Multi-Agent Differential Testing for the Game of Go

Jiaxue Song
Beijing University of Chemical Technology
Beijing, China
songjiaxue000816@163.com

Xiao-Yi Zhang
University of Science and Technology Beijing
Beijing, China
xiaoyi@ustb.edu.cn

Paolo Arcaini
National Institute of Informatics
Tokyo, Japan
arcaini@nii.ac.jp

Fuyuki Ishikawa
National Institute of Informatics
Tokyo, Japan
f-ishikawa@nii.ac.jp

Yong Liu
Beijing University of Chemical Technology
Beijing, China
lyong@mail.buct.edu.cn

Bin Du
Beijing University of Chemical Technology
Beijing, China
dubin@mail.buct.edu.cn

Abstract

Artificial intelligent (AI) techniques have been successfully applied in various domains, especially in complex games like Go. Indeed, although Go has simple rules, it has countless variations of possible positions, making it an extremely difficult game. Therefore, it has been taken as a benchmark to assess the abilities of AI agents (called *Go Agents*). However, assessing the performance of the decision-making of a Go agent is challenging; indeed, since Go agents play much better than humans, it is difficult to establish intuitive rules or to rely on human experts to determine whether a Go agent's decision is correct or not. While professional players can do it, they can not evaluate too many moves. To tackle these issues, in this paper we propose a differential testing approach, called *Multi-Agent Differential testing For the game of Go* (MAD4Go), to identify interesting tests in which a Go agent may perform a non-optimal move. Specifically, we play different Go agents over the same tests and we check whether they agree with each other. In case of disagreement, we assess the *level of disagreement*. If two agents *strongly disagree* with each other, it is more likely that at least one agent made a wrong decision. We conducted experiments by evaluating Go agents from Leela Zero, using as tests different Go positions obtained from real Go games of professional players. Results revealed diverse disagreements among agents, showing that MAD4Go can identify valuable tests.

CCS Concepts

• **Theory of computation → Reinforcement learning**; • **Software and its engineering → Software testing and debugging**.

Keywords

differential testing, Go game, AI agents

FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3730553

ACM Reference Format:
Jiaxue Song, Xiao-Yi Zhang, Paolo Arcaini, Fuyuki Ishikawa, Yong Liu, and Bin Du. 2025. Multi-Agent Differential Testing for the Game of Go. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3696630.3730553

1 Introduction

The game of *Go* is a discrete and zero-sum game in which two players alternatively place white and black stones on a square board to occupy larger areas of the board. Despite its simple rules, it is an extremely complex game. First, it has around 10^{172} *positions* (i.e., placements of stones), making it impossible to find a perfect strategy to win the game. Secondly, playing the game requires not only the calculation of future moves but also strategies such as judging the concepts of *thickness* and *balance* between outside influence and territory.

Due to its complexity, Go has been taken as benchmark to assess the reasoning capabilities of AI. In this field, AI techniques have obtained notable successes by the development of *agents* (*Go agents* in the following) that rely on the combined use of deep neural networks (DNNs) and Monte Carlo Tree Search (MCTS). In 2015, DeepMind proposed AlphaGo [12] that won 4:1 against Sedol Lee, one of the top Go players at the time. AlphaGo was later extended in the even more powerful version AlphaGo Zero [13].

Assessing the quality of a Go agent and identifying some mistakes in its decision logic is a challenge for two main reasons:

- the environment of Go is huge, and the decision of each move involves the consideration of several complicated logical chains. Consequently, it is difficult to establish some definitive rules or metrics based on domain knowledge to determine whether a move at a specific position is *good* (i.e., resulting in a winning situation or maximizing the probability of winning) or *bad* (i.e., resulting in the loss of the game or reducing the possibility of winning);[1]
- the performance of a Go agent often significantly surpasses that of human players. Even when the decisions of Go agents are not optimal, they are still better than those of humans. Therefore, using human players to judge the quality of the moves of Go agents is not possible. While top professional

[1]We avoid using terminology as "correct" and "wrong", as, in general, there is not a single correct move.

players are in principle able to assess these moves, they need to spend a considerable amount of time for doing so; thus, they can only evaluate very few moves.

In this paper, we propose an approach to select some *tests* (defined as Go board positions) in which the moves of a Go agent are possibly not optimal, and could lead to the loss of the game. The goal is to select very few tests for further evaluation by professional players, so reducing the cost of testing while enhancing its effectiveness. Such tests can be later used for understanding and further improvement (e.g., fine tuning) of the agents.

For the reasons described above, it is difficult to assess the decisions of a single Go agent. On the other hand, comparing the decisions of different Go agents seems more promising; the intuition is that, in the cases in which two agents disagree, the decision of one agent may be not good. By following this intuition, we propose *Multi-Agent Differential testing For the game of Go* (MAD4Go), to automatically assess different tests and select those that are more likely to expose bad decisions of some Go agents.

Specifically, we execute different Go agents over the same tests and we check whether they *agree* (i.e., they play the same move), or they *disagree* (i.e., they play a different move). However, a simple disagreement between two agents (as computed by previous works [21]) is not necessarily a sign of a bad move; indeed, two agents may play different moves because they adopt different playing strategies, but they may still recognize the move played by the other agent as valuable. Therefore, we introduce three levels of disagreement: *low*, *medium*, and *strong*. In *low disagreement*, both agents recognize the move played by the other player as good. In *medium disagreement*, only one of the players recognizes the move of the other player as good. In *strong disagreement*, both agents do not recognize the move of the other player as good; this is the most interesting case, as it identifies situations in which two agents have totally different opinions and, probably, one is at fault.

To summarize, the main contributions of this paper are:

- a way to compare the moves of different Go agents, that checks whether the agents agree or disagree. In case of disagreement, we define three levels of disagreement.
- a differential testing process (MAD4Go) based on the comparison of different Go agents. The approach takes as input a set of tests, i.e., a set of Go positions. The agents are asked to play a move from each tests, and to evaluate each other's moves. In this way, we assess their agreements and disagreements. From the results, we select as most interesting those tests that show the highest level of disagreement.

We experimented with MAD4Go using Go games selected from a database of real games played by professional human players. From the games, we generated over 15,000 tests by selecting Go positions at different stages of the games. As Go agents, we selected six agents of varying strengths, derived from different weight networks of Leela Zero. Experimental results show that MAD4Go can identify tests that show *strong disagreement* among different Go agents; these tests are likely to show some bad decisions of some Go agents.

According to the SIGSOFT Empirical Standards[2], this paper provides an "Engineering Research" contribution.

[2] https://www2.sigsoft.org/EmpiricalStandards/docs/standards

Paper structure. Sect. 2 introduces background concepts related to testing Go agents, and Sect. 3 presents the proposed approach MAD4Go. Then, Sect. 4 describes the design of the experiments we conducted, and Sect. 5 discusses experimental results. Finally, Sect. 6 discusses threats that may affect the validity of the approach, Sect. 7 reviews related work, and Sect. 8 concludes the paper.

2 Preliminaries

In this section, we introduce basic definitions related to the game of Go, Go agents, and testing of Go agents.

Definition 1 (Go position and game)**.** A *Go position* p is given by the placement of white and black stones in the intersections of the 19×19 grid of a board. A *Go game* g is a sequence of Go positions (i.e., $g = [p_0, \ldots, p_n]$) obtained by alternatively applying valid moves of the *Black* and *White* players, where p_0 indicates the initial position in which the board is empty. A *move* mv is the action of placing a stone on the board. By playing a move mv on position p, we obtain the successive position, i.e., $p_{i+1} = mv(p_i)$. Thus, a Go game can also be represented as a sequence of moves $[mv_1, \ldots, mv_n]$.

Definition 2 (Go agent)**.** A *Go agent* ag is an AI-based software that can play Go. A classical type of Go agent is given by the integrated use of Monte Carlo Tree Search (MCTS) and of a deep neural network (DNN). The *structure* and the *weights* of the DNN determine a specific agent. Given a Go position p, agent ag produces a list of *candidate moves* $CM = [cm_1, \ldots, cm_k]$, where each candidate is identified by the pair $cm_i = \langle mv_i, rew_i \rangle$: mv_i is a possible move considered by ag in position p, and $rew_i \in [0, 100]\%$ is the *reward* for mv_i that assesses the benefit expected in the long term by playing mv_i, i.e., the percentage of game wins in repeated continuations of the game after mv_i. Note that CM is sorted based on the *rew* values, from higher to lower. Thus, the move mv_1 of the first candidate cm_1 in CM is the next move to perform.

Since the reward *rew* of a move indicates the percentage of wins in repeated continuations of the game after that move, when the reward score exceeds 50%, this indicates that the Go agent estimates that the current player has an advantageous position and it is statistically more likely to ultimately win the game.

Note that, due to time limitations, agents are usually given a time budget to calculate the candidate moves CM. So, in general, not all the possible moves are evaluated, and it can happen that an agent does not consider a good move.

Testing Go agents means evaluating their performance from different situations. We define a test as follows.

Definition 3 (Test)**.** A *test* t is given by the pair $\langle p, S \rangle$, where p is the Go position from which the game should continue and S indicates which player (Black or White) must play. In our work, tests are derived from real games without any handicaps, in which the Black and White take turns to place stones in sequence, with Black moving first. Thus, we assume the turn in a test has already been known, and the test t can be simplified as the Go position p.

Tests are generated by recording game positions from real Go games. For a specific game g, we first capture all board positions throughout the game sequence: $[p_0, \ldots, p_n]$. Each Go position p_k can be used as a test.

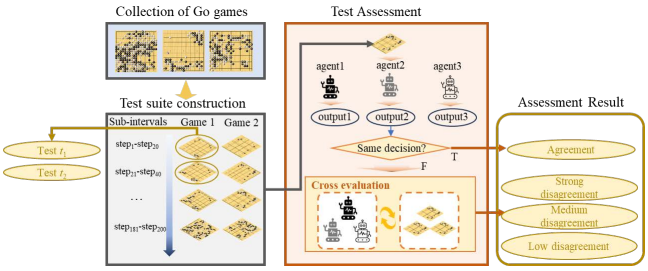


Figure 1: MAD4Go – Multi-Agent Differential Testing for the Game of Go

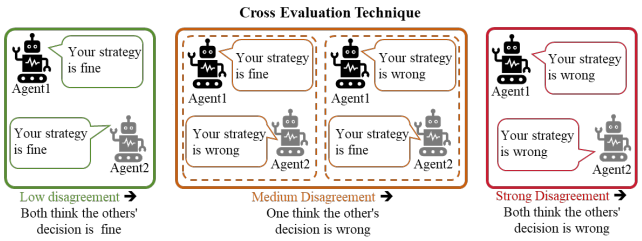


Figure 2: Different levels of disagreements

Executing a test t means using an agent ag to perform a test t. The output of test execution is the tuple:

$$O^t_{ag} = \langle CM, \widetilde{mv}, \widetilde{rew} \rangle \quad (1)$$

where:

- CM is the ranked list of candidate moves provided by agent ag (see Def. 2);
- $\widetilde{mv}$ is the move that ranks the first and it is chosen by the agent;
- $\widetilde{rew}$ is the reward of $\widetilde{mv}$.

The overall goal of Go testing is to find Go positions in which a Go agent makes bad moves that will likely lead to lose the game.

3 MAD4Go – Multi-Agent Differential testing For the game of Go

Due to the huge number of possible Go positions and the complexity of the game, it is challenging to establish criteria to automatically assess the quality of moves generated by Go agents. Moreover, most Go agents have capabilities that surpass those of a large part of human Go players, and so it is not easy to manually assess the moves of Go agents. This can be done only by professional Go players; however, in order to do this, they need to conduct rigorous analysis, which is very time-consuming. Thus, it is infeasible to assess too many tests in this manner.

To address these issues, we propose *Multi-Agent Differential testing For the game of Go* (MAD4Go), an approach that automatically selects few tests that have high probability of showing bad moves of a Go agent. These few tests can be shown to professional Go players for manual assessment; in this way, the effort required for manual assessment is minimized. The overall description of MAD4Go is shown in Fig. 1 and described in the following sections.

3.1 Test suite construction and execution

The approach takes in input a collection of real Go games played by professional players and builds a test suite TS by treating some Go positions p_i of each game $g = [p_0, p_1, \ldots, p_n]$ as a test t.

The approach also takes in input a set of Go agents $\mathcal{A} = \{ag_1, \ldots, ag_m\}$ that we want to test.

Then, for each agent $ag \in \mathcal{A}$ and test $t \in TS$, the approach plays ag over t, obtaining the test result $O^t_{ag} = \langle CM, \widetilde{mv}, \widetilde{rew} \rangle$.

3.2 Test assessment

After the execution of all the tests in TS by all the Go agents $\mathcal{A}$, the approach checks the level of agreement and disagreement between the agents.

Specifically, for each test $t \in TS$ and pair of agents $\langle ag_i, ag_j \rangle$, it collects their test outputs $O^t_{ag_i}$ and $O^t_{ag_j}$ (see Eq. 1).

If $O^t_{ag_i}.\widetilde{mv} = O^t_{ag_j}.\widetilde{mv}$ (i.e., the agents played the same move), we say that the agents *agree* and we add the test to the set TS^{agree}_{ij} that collects tests in which the two agents agree.

Instead, if $O^t_{ag_i}.\widetilde{mv} \neq O^t_{ag_j}.\widetilde{mv}$, this indicates a *disagreement* in their decision-making. However, a disagreement alone does not necessarily imply a fundamental opposition between the agents' strategies. To systematically analyse these differences, we categorize disagreement into three levels (as shown in Fig. 2):

- **Low Disagreement.** Although the two agents play a different move, they recognize that the move played by the other agent is still good, and that the final result of the game will not change by applying one of the two moves. This typically occurs when the player that must move is in an irreversible strategic position (either it has a clear dominance of the board or it has an irreversible disadvantage), such that any next move can not alter the predetermined outcome. We define TS^{Low}_{ij} as the set of tests in which agents ag_i and ag_j have a low disagreement.
- **Medium Disagreement.** In this case, one agent acknowledges that the move of the other agent is good, while the other agent thinks that the move of the other agent is not good. Specifically, the first agent believes that the final outcome (win or loss) of the game will remain unchanged regardless of which moved is played; instead, the second agent thinks that adopting its own decision has a higher chance of leading the current player to victory, whereas following the other agent's strategy there is a higher chance of loss. We identify with TS^{Medium}_{ij} the set of tests in which agents ag_i and ag_j have a *medium disagreement*.
- **Strong Disagreement.** In this case, the two agents believe that the other agent's move is not good. Each agent is convinced that adopting its own decision has higher chances to lead the current player to victory, whereas following the other agent's strategy has higher chances of loss. We identify with TS^{Strong}_{ij} the set of tests in which agents ag_i and ag_j have a *strong disagreement*.

While the previous description gives of an intuition of the different levels of disagreement, it does not provide a way to automatically classify the tests in the three disagreement levels. Therefore, we propose a metric that allows to do so, as described in the following.

3.2.1 Disagreement level computation. We define an evaluation function $eval(t, mv, ag)$ to assess the "opinion" of agent ag regarding a move mv to be played in test t; the opinion is assessed by the reward assigned by ag to that move.

Using the evaluation function, given a test t, two agents ag_i and ag_j, and their test results $O^t_{ag_i}$ and $O^t_{ag_j}$, we can ask the opinion of each agent about the other agent's move. For example, in order to ask the opinion of agent ag_i regarding the move played by agent ag_j in test t (i.e., $O^t_{ag_j}.\widetilde{mv}$) we proceed as follows:

- if the moved played by agent ag_j is among those considered by agent ag_i in $O^t_{ag_i}.CM$, the evaluation is given the reward computed by ag_i for that move, i.e.,

$$\begin{aligned} &EV^t_{ij} = eval(t, O^t_{ag_j}.\widetilde{mv}, ag_i) = cm^*.rew \\ &\text{with} \\ &cm^* \in O^t_{ag_i}.CM \wedge cm^*.mv = O^t_{ag_j}.\widetilde{mv} \end{aligned} \tag{2}$$

- if the moved played by agent ag_j is not among those considered by agent ag_i in $O^t_{ag_i}.CM$, we do not have access to the assessment of agent ag_i for the move. On the other hand, we can assess the opinion of ag_i on the "consequences" of playing the move; the intuition is: if playing the move $O^t_{ag_j}.\widetilde{mv}$ leads to a Go position t' where the opponent can play a very good move, the move $O^t_{ag_j}.\widetilde{mv}$ is not good. By following this intuition, we proceed as follows:
  - we play the move $O^t_{ag_j}.\widetilde{mv}$ under evaluation, obtaining a new Go position t' to be used as test, i.e.,

$$t' = O^t_{ag_j}.\widetilde{mv}(t) \tag{3}$$

  - we then execute agent ag_i from t', obtaining the test result $O^{t'}_{ag_i}$. The reward $O^{t'}_{ag_i}.\widetilde{rew}$ of the best move to be played in t' tells how much the previous move $O^t_{ag_j}.\widetilde{mv}$ has been able to create a challenging position for the opponent player: the lower $O^{t'}_{ag_i}.\widetilde{rew}$ is, the better $O^t_{ag_j}.\widetilde{mv}$ has been. By following this intuition, the evaluation of $O^t_{ag_j}.\widetilde{mv}$ is given by the complement of the reward $O^{t'}_{ag_i}.\widetilde{rew}$, i.e.,:

$$EV^t_{ij} = eval(t, O^t_{ag_j}.\widetilde{mv}, ag_i) = 100\% - O^{t'}_{ag_i}.\widetilde{rew} \tag{4}$$

In the following, for consistency, we also identify as $EV^t_{ii} = O^t_{ag_i}.\widetilde{rew}$ the reward assigned to an agent ag_i to the chosen move.

An evaluation value EV tells us about the confidence of winning or losing the game with that move. We can classify the value in three categories as follows:

$$Cat = \begin{cases} \texttt{G} & \text{if} \quad EV > 50\% + \delta \\ \texttt{B} & \text{if} \quad EV < 50\% - \delta \\ \texttt{N} & \text{otherwise} \end{cases} \tag{5}$$

where:

- Cat = G indicates that the agent thinks that the move is Good (possibly leading to a win);
- Cat = B indicates that the agent thinks that the move is Bad (possibly leading to a loss);
- Cat = N indicates cases in which the agent is Neutral, i.e., it does not have a strong opinion about the possibility of the move of leading to a win or a loss.

In Eq. 5, 50% indicates a neutral winning probability and δ is a hyperparameter of the approach that determines the minimum required value for determining a good or bad move.

Finally, given a pair of agents $\langle ag_i, ag_j \rangle$, a test t, and the evaluations EV^t_{ii}, EV^t_{jj}, EV^t_{ij}, and EV^t_{ji} (with their corresponding categories Cat^t_{ii}, Cat^t_{jj}, Cat^t_{ij}, and Cat^t_{ji}), we can assess the disagreement level in this way:

- if the category of one of the assessments (i.e., Cat^t_{ii}, Cat^t_{jj}, Cat^t_{ij}, and Cat^t_{ji}) is N, we discard the test t, as it is not useful to compare the opinions of the agents; otherwise, we check the disagreement level as in the following points;
- there is *low disagreement* if it holds:

$$(Cat^t_{ij} = Cat^t_{jj}) \wedge (Cat^t_{ji} = Cat^t_{ii}) \tag{6}$$

  It means that the two agents, although they play differently, agree on the quality of the moves played by each other;
- there is *medium disagreement* if it holds:

$$\begin{aligned} &((Cat^t_{ii} = Cat^t_{jj} = \texttt{G}) \wedge (Cat^t_{ij} \neq Cat^t_{ji})) \vee \\ &((Cat^t_{ii} \neq Cat^t_{jj}) \wedge (Cat^t_{ij} = Cat^t_{ji} = \texttt{B})) \end{aligned} \tag{7}$$

  It considers two cases: (i) each agent thinks that its move is good, and one of the two agents thinks that also the other agent's move is good; (ii) only one of the two agents thinks that its own move is good, but both agents think that the other agent's move is bad;
- there is *strong disagreement* if it holds:

$$(Cat^t_{ii} = Cat^t_{jj} = \texttt{G}) \wedge (Cat^t_{ij} = Cat^t_{ji} = \texttt{B}) \tag{8}$$

  It means that each agent thinks that its own move is good, and the other agent's move is bad.

In summary, the three levels of disagreement can be categorized using the cases shown in Table 1.[3]

4 Experimental design

In this section, we report the design of the preliminary experiments we conducted to assess MAD4Go. All test suites and full experimental results are reported online at [14].

4.1 Research questions

We identified the following research questions (RQs) to guide the experiments to assess MAD4Go.

RQ1 (Overall (dis)agreement) *What is the percentage of agreement and disagreement (at the three levels) between all agent pairs?*

Through RQ1, we aim to examine to what extent agents agree

[3]There are other possible cases in which an agent thinks that its own move is bad and the other agent's move is good. These cases are due to a not sufficient evaluation of the possible moves by the agents. However, these cases are extremely rare (only 10 cases out of more than 15,000 tests) and so they will be excluded from the results.

Table 1: Disagreement level computation

Disagreement level	Category			
	Cat^t_{ii}	Cat^t_{jj}	Cat^t_{ij}	Cat^t_{ji}
Strong	G	G	B	B
Medium	G	G	G	B
	G	G	B	G
	G	B	B	B
	B	G	B	B
Low	G	G	G	G
	B	B	B	B
	G	B	B	G
	B	G	G	B

and disagree, and, in case of disagreement, at what level. This allows to assess the ability of MAD4Go in discriminating and prioritizing tests based on disagreement levels.

RQ2 (**Evolution of (dis)agreement**) *How do the agreement and various disagreement levels evolve throughout the different phases of the game?*
Through RQ2, we assess agreement and disagreement at different stages of the game, to identify whether there are stages where the agents are more likely to disagree with each other.

RQ3 (**Qualitative evaluation**) *What does characterize the varying levels of disagreement among agents?*
For RQ3, we seek to conduct a qualitative evaluation to assess whether the tests selected by MAD4Go—those that cause strong disagreement between agents–can indeed reveal potential bad decisions of the agents.

4.2 Experimental settings

MAD4Go requires in input a test suite TS of Go positions. When designing tests, a critical factor is obtaining realistic positions; in order to do so, we consider a repository of real games [5] played by professional Go players across eight skill levels (*1d* to *8d*). For each level, we sample 100 games, for a total of 800 games. Each game $g = [p_1, \ldots, p_n]$ is partitioned into sub-intervals of 10 consecutive moves (i.e., $[p_1, \ldots, p_{10}]$, $[p_{11}, \ldots, p_{20}]$, ...). This division reflects the different phases of Go gameplay, such as *fuseki* (opening), *chuban* (middle game), and *endgame*, in which agents may exhibit varying strategic capabilities. For each sub-interval, we randomly select one position p and the associated stone S (Black or White) to play next based on the game's move sequence. A test is then generated as the tuple $t = \langle p, S \rangle$. So, in total, we generate a test suite TS of 15,564 tests.

In order to answer RQ2, we generate subsets of TS corresponding to specific move intervals, denoted as TS_I, where $I \in \{[1, 20], [21, 40], \ldots [241, 260], [261, 500]\}$); all intervals have a size of 20 steps, except for the last interval that collects the few cases that have more than 261 steps. Each TS_I is analysed separately to assess Go agents' behaviours across various phases.

MAD4Go aims to evaluate and select tests that expose disagreements among different Go agents. The website [8] provides multiple Go agents. For each agent, it specifies the DNN architecture and weights (See Def. 2) alongside its *Elo rating* [3] (a method to assess the skills of players in zero-sum games). The Elo difference between two agents predicts their expected win ratio; for example, an agent with a 100-point Elo advantage over another one is expected to win 64% of games. We selected six agents as reported in Table 2. The table reports, for each agent, a unique ID, the DNN architecture and weight identifiers recorded in [8], and the Elo score derived from repeated game evaluations. We selected the agents from [8] by following this criterion: two *top*-ranked agents (with Elo score higher than 16,700), two *middle*-ranked agents (with Elo score around 11,200) and two *low*-ranked agents (with Elo score around 4,600).

Table 2: Go agents selected for the experiments

Agent ID	ID of DNN and weights in [8]	Elo score in [8]
Top1	0e9ea880	16,726
Top2	e6e2e99b	16,688
Mid1	057adf2c	11,252
Mid2	521b0868	11,142
Low1	30f49ac5	4,855
Low2	6140dc2e	4,475

To analyse the generated tests and selected Go agents using the MAD4Go framework, we compute agreement/disagreement metrics between each agent pair for each test $t \in TS$. This requires executing all tests with every agent, requiring a total of $15,564 \times 6 = 93,384$ test executions. Note that, when calculating disagreement levels, if one agent's chosen move does not appear in the other's candidate list, this requires an additional test execution (see Sect. 3.2.1). Since each test execution takes approximately 5 seconds, the overall experiment took about 6 days of computation.

The assessment of an evaluation value EV uses a margin δ (over and below 50%) to declare a move as Good or Bad (see Eq. 5). In the experiments, we set the conservative value of $\delta = 5\%$.

5 Experimental results

In this section, we answer the RQs reported in Sect. 4.1

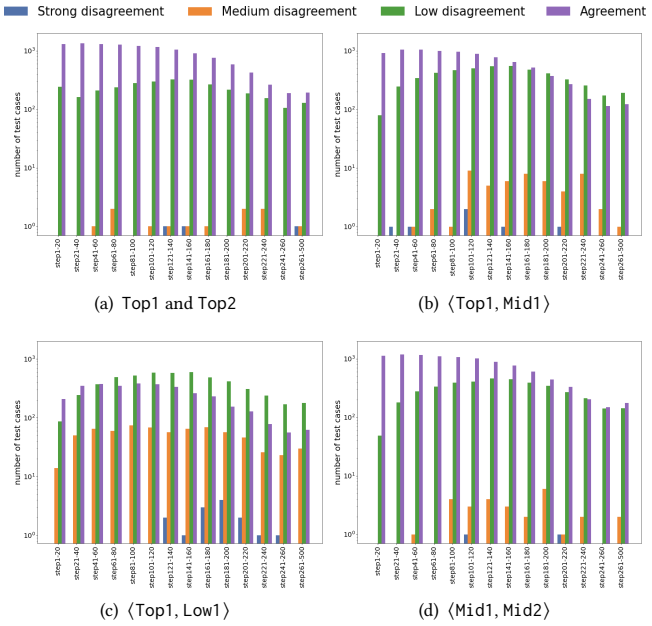
RQ1 – Overall (dis)agreement. In this RQ, we want to assess to what extent different types of agents agree and disagree. Table 3 reports, for each pair of agents, the number of agreement cases, and the number of disagreement cases for each disagreement level (strong, medium, and low). We report in parentheses the percentage relative to the total number of tests.[4]

The results show a significant number of agreement cases. Among all the 233,460 comparisons, the agent pairs reach an agreement in 90,072 of the cases (38.6%). The total disagreement cases amount to 80,471 (34.5%), with 74,408 classified as low-level disagreements. Combined agreement and low-disagreement cases account for around 70% of all comparisons. Thus, we can infer that the agents generally exhibit reasonable reasoning capabilities and can reach consensus or maintain minor differences in most cases.

[4]Note that the sum for each row can be lower than the number 15,564 of total tests, as the tests that do not obtain a clear classification as Good or Bad are not considered (see Sect. 3.2.1).

Table 3: RQ1 – Agreement and disagreement cases. Number in parentheses is the percentage

Agent pairs		Disagreement level			Agreement
		Strong	Medium	Low	
Top1	Top2	3 (0.02)	12 (0.08)	3146 (20)	12,025 (77)
Top1	Mid1	6 (0.04)	53 (0.34)	5,006 (32)	8,854 (57)
Top1	Mid2	5 (0.03)	71 (0.46)	4,996 (32)	8,750 (56)
Top1	Low1	14 (0.09)	704 (4.5)	5,279 (34)	3,344 (21)
Top1	Low2	8 (0.05)	827 (5.3)	6,065 (39)	3,382 (22)
Top2	Mid1	8 (0.05)	48 (0.31)	4967 (32)	8,977 (58)
Top2	Mid2	6 (0.04)	65 (0.42)	4,935 (32)	8,810 (57)
Top2	Low1	15 (0.10)	700 (4.5)	5,223 (34)	3,381 (22)
Top2	Low2	6 (0.04)	833 (5.4)	6,000 (39)	3,419 (22)
Mid1	Mid2	2 (0.01)	28 (0.18)	4,070 (26)	10,288 (66)
Mid1	Low1	13 (0.08)	610 (3.9)	4,988 (32)	3,316 (21)
Mid1	Low2	5 (0.03)	679 (4.4)	5,287 (34)	3,329 (21)
Mid2	Low1	20 (0.13)	591 (3.8)	4,881 (31)	3,429 (22)
Mid2	Low2	10 (0.06)	691 (4.4)	5,156 (32)	3,470 (22)
Low1	Low2	0 (0.0)	20 (0.13)	4,409 (28)	5,298 (34)
Total		121	5,932	74,408	90,072



(a) Top1 and Top2 (b) ⟨Top1, Mid1⟩

(c) ⟨Top1, Low1⟩ (d) ⟨Mid1, Mid2⟩

Figure 3: RQ2 – Evaluation at different phases of the games

Considering the disagreement cases, the number of *medium* disagreements is 5,932 cases (2.5% of total tests), and the number of *strong* disagreement comparisons is only 121 (0.05%). These few strong disagreement cases are the most interesting ones, as they most probably identify wrong moves of one of the two agents.

Moreover, it can be observed that different disagreement patterns exist across different agent pairs. Agents with higher abilities (i.e., Top1 and Top2) exhibit higher consensus rates, achieving agreement in 12,025 cases (77.2% of their pairwise tests), suggesting their capability to identify appropriate moves in most game positions. Still, our approach detected 15 strong and medium disagreement cases for the pair ⟨Top1, Top2⟩. The associated tests could be valuable for exposing potential decision errors in advanced Go agents.

In addition, Mid1 and Mid2 also exhibit similar consensus patterns: they reach an agreement in 10,288 of 15,564 comparisons (66.2%). This indicates that agents with comparable strength levels tend to share similar decision-making patterns. Conversely, agent pairs with significant skill gaps (e.g., Top1 and Low1) yield more frequent and severe disagreements. However, these disagreements often reflect the weaker agent's limitations rather than revealing critical insights about the advanced agents, thus carrying less value compared to disagreements between high-performing agents.

RQ2 – Evolution of (dis)agreement. In this RQ, we are interested in assessing whether agreement and disagreement change during the different phases of the game. Given the high number of agent pairs, we select four representative pairs: the comparison between the top-performing Go agent Top1 and three other agents (Top2, Mid1, and Low1), and the comparison between Mid1 and Mid2. Fig. 3 reports the results for the four comparison (note the logarithmic scale). Each sub-figure contains four bar charts per game interval (each of 20 steps, except for the last one that considers all steps above 261), quantifying the number of tests for which there is an agreement, and those for which there is a low, medium, and strong disagreement.

From the figures, we observe that the agents tend to reveal agreement on the played moves during the opening phases (typically the first 50 moves). This can be due to the prevalence of established opening patterns (e.g., *Joseki*).

After the moves of the opening phase, the complexity of the game increases, and each move requires deep evaluation, thereby increasing the performance gaps between agents with different skill levels and different playing strategies. Therefore, mid-game phases (i.e., moves from 100 to 200) exhibit several low disagreement instances across all four agent pairs.

The medium and strong disagreements demonstrate different distributions for the different pairs. For the pair ⟨Top1, Low1⟩, medium and strong disagreement cases occur across all game phases (predominantly in mid/late-game); this is expected, as Top1 usually produces much better moves than Low1. For pairs ⟨Top1, Top2⟩ and ⟨Mid1, Mid2⟩, instead, medium and strong disagreement cases are less frequent but still tend to distributed in the phase of mid-game; as observed in RQ1, the fact that the number of medium and strong disagreement is low is due to the fact that the agents are quite good, and tend to generate reasonable moves; the fact that the disagreement is focused in the mid-game could be due to the fact that, in this part of the game, the different strategies of the agents are more relevant.

Considering the endgame stage (known as *Yose*, typically beginning after 260 moves), disagreements between agents tend to be less frequent. This reduction occurs mainly because, at this phase, the game outcome is already determined.

Generally speaking, these findings indicate high strategic consistency among agents during opening and endgame phases, while

(a) Strong disagreement – Top1's move (b) Strong disagreement – Top2's move

(c) Low disagreement – Top1's move (d) Low disagreement – Top2's move

Figure 4: RQ3 – Qualitative evaluation

disagreement increases in the mid-game phase. Consequently, our test selection may prioritize mid/late-game tests, in which we can find more tests leading to strong or medium disagreement among the agent pairs.

RQ3 – Qualitative evaluation. To perform a preliminary validation on whether MAD4Go can effectively identify valuable tests that reveal potential vulnerabilities in Go agents, we conducted a qualitative analysis. We examined two cases with different disagreement levels identified through MAD4Go for the two top-performing agents Top1 and Top2. Fig. 4 illustrates the tests in which the two agents exhibit strong disagreement and low disagreement levels. For analysis, we employed KataGo (one of the strongest Go AIs) to evaluate Go moves for a long period of time and combined its response with professional domain knowledge of the Go game, in order to assess the correctness of the agents' decision.[5] The blue numbers in Fig. 4 indicate predicted score margins; for example, in KataGo's scoring system, *B1.3* indicates that Black currently leads by 1.3 points, while *W0.7* denotes White leading by 0.7 points.

Figs. 4(a) and 4(b) illustrate a test in which Top1 and Top2 strongly disagree with each other; the two figures show the different moves played by the agents. The current game between Black and White is extremely close, as the score difference is less than 1 point. In this position, Top1 recommends move S5 (with reward $\widetilde{rew} = 74.67$) while Top2 recommends R4 (with reward $\widetilde{rew} = 71.39$). When they evaluate the moves provided by the other, Top1 estimates a 43.68% winning probability for Top2's R4 move (i.e., $EV^t_{\text{Top1Top2}} = 43.68$), and Top2 estimates a 37.49% winning probability for Top1's S5 (i.e., $EV^t_{\text{Top2Top1}} = 37.49$). The analysis conducted by an experienced player[6] reveals that S5 creates a "half-eye" through a "ko" in the bottom-right corner. Although it leads to the sacrifice of a stone, this deliberate exchange proves to be strategically worthwhile. Instead, R4 allows Black to extend liberties, ultimately leading to a "seki" (dual life) and loss of White's territory. KataGo calculations confirm that S5 maintains White's 0.6-point lead versus Black's 1.3-point advantage with R4. In other words, Top2's decision incurs approximately 2-point loss, which could be critical in a close game.

Figs. 4(c) and 4(d) display a low-disagreement test in which both moves maintain White's 20-point advantage, as Black's upper stones are already dead. The two agents, although they play a different move, recognize that also the other move is good. This type of test is less valuable, as it does not show any deficiency of the agents.

In general, we can see that comprehensive evaluation of Go agents' performance requires professional game analysis and the computation of lots of variations, which is resource-consuming. MAD4Go can successfully identify critical disagreement cases between top agents that reveal meaningful decision-making limitations, particularly in complex positions where bad moves may determine game outcomes.

6 Threats to validity

Construct validity. The goal of MAD4Go is to find cases in which agents disagree. Therefore, RQ1 and RQ2 investigate the agreement and disagreement levels between all the pairs of agents, overall and for different stages of the game. The assumption of MAD4Go is that tests showing strong disagreement could be particularly interesting, and show some really bad move; in RQ3, we did a preliminary validation in which we used a very good AI agent for a long time to assess two tests showing strong and low disagreement.

External validity. It could be that the results do not generalize to different types of agents. To mitigate this threat, we considered different types of agents (with different skill levels) from a popular repository of Go agents.

7 Related work

The goal of MAD4Go is to assess and select tests for Go agents. Different AI techniques have been employed to develop such agents. First approaches used techniques like Alpha-Beta Pruning [17]; later, Monte Carlo Tree Search (MCTS) was adopted for enhanced search-space exploration [2]. The breakthrough came with AlphaGo [12], the first AI to defeat professional players by combining deep neural networks (DNNs) trained on human games with MCTS guidance. AlphaGo Zero [13] provided further improvement by using pure reinforcement learning based on ResNet-based DNNs. In our work, we utilize Leela Zero (the leading open-source implementation) to select Go agents [7, 8] and KataGo (one of the best state-of-the-art AIs) [20] for the qualitative evaluation in RQ3.

Our approach is inspired by differential testing [4], a testing approach that compares different versions of a system under test using the same test suite; test results are usually compared to check for differences [6]. For example, Prochnow and Yang [11] proposed

[5] Note that this type of evaluation can only be done for few tests, but it is not feasible to adopt in practice for all the tests as it is too computational expensive.

[6] One of the authors is an advanced Go player with an experience of 20 years.

DiffWatch, a differential testing approach to detect changes of deep learning libraries. Tan et al. [15] proposed an automated approach to quantify the quality of knowledge graphs via differential testing. Misse-Chanabier et al. [9] proposed a differential testing method to find bugs in Virtual Machine (VM) generators. Regarding games, Castellano et al. [1], proposed a differential testing approach whose goal is to discover differences in agents that play in a Game-as-a-Service (GaaS) [19] type of game; the approach in [1] does not compare the decisions of the agents (as done in our work), but the features of the game state. In our previous work [21], we conducted a very preliminary exploration to examine when Go agents perform different moves. However, we only considered agreement and disagreement, and we did not distinguish among the different levels of disagreement; therefore, the previous approach was not able to identify valuable tests.

MCTS, one of the main elements of Go agents (MCTS) relies on reinforcement learning (RL). The testing of RL has been considered in the literature. For example, Trujillo et al. [18] experimented with different coverage criteria with the goal of testing applications based on RL. Zolfagharian et al. [22] used a search-based approach to test RL agents. A similar approach was employed by Tappler et al. [16] to assess the safety and performance of deep RL agents. Nikanjam et al. [10], instead, proposed an approach to detect bugs in RL applications like OpenAI Gym.

8 Conclusions

AI agents used in games like Go (Go agents) are much stronger than most human players. Only professional Go players can judge their moves; however, this is very time consuming and can only be done for very few tests. In this paper, we propose `MAD4Go`, a differential testing approach that compares the decisions of different Go agents over different tests, and prioritize those tests for which the agents have the highest disagreement.

As future work, we plan to conduct a large empirical evaluation showing that the selected tests are indeed the most interesting ones; this would require to conduct long evaluations of each test using the most advanced Go agents, as done in RQ3. Moreover, we plan to apply our proposed categorization of disagreement levels also to AI agents used in other types of games, like those available in OpenAI Gym. The application to different games will need to identify the best metric for scoring moves.

Acknowledgments

Xiao-Yi Zhang is the corresponding author. X. Zhang is supported by Youth Fund of the National Natural Science Foundation of China (No. 62302035). P. Arcaini is supported by JSPS KAKENHI Grant Number JP23K11062. F. Ishikawa is supported by Engineerable AI Techniques for Practical Applications of High-Quality Machine Learning-based Systems Project (Grant Number JPMJMI20B8), JST-Mirai.

References

[1] Ezequiel Castellano, Xiao-Yi Zhang, Paolo Arcaini, Toru Takisaka, Fuyuki Ishikawa, Nozomu Ikehata, and Kosuke Iwakura. 2023. Explaining the Behaviour of Game Agents Using Differential Comparison. In *37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE22)*. Association for Computing Machinery, New York, NY, USA, Article 210, 8 pages. doi:10.1145/3551349.3560503

[2] Adrien Couëtoux, Martin Müller, and Olivier Teytaud. 2013. Monte Carlo Tree Search in Go. (2013). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=06ff771c690a34981eb8b8576bd9e202def23358 unpublished.

[3] A.E. Elo. 1966. *The USCF Rating System: Its Development, Theory, and Applications.* United States Chess Federation. https://books.google.co.jp/books?id=onUazQEACAAJ

[4] Robert B. Evans and Alberto Savoia. 2007. Differential Testing: A New Approach to Change Detection. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Dubrovnik, Croatia) *(ESEC-FSE '07)*. Association for Computing Machinery, New York, NY, USA, 549–552. doi:10.1145/1287624.1287707

[5] FoxGoDataset [n. d.]. Fox Go Dataset. https://github.com/featurecat/go-dataset.

[6] Muhammad Ali Gulzar, Yongkang Zhu, and Xiaofeng Han. 2019. Perception and practices of differential testing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 71–80. doi:10.1109/ICSE-SEIP.2019.00016

[7] LeelaZero 2023. Leela Zero. https://github.com/leela-zero/leela-zero.

[8] LeelaZeroagents 2021. Leela Zero agents. https://zero.sjeng.org/.

[9] Pierre Misse-chanabier, Guillermo Polito, Noury Bouraqadi, Stéphane Ducasse, Luc Fabresse, and Pablo Tesone. 2022. Differential Testing of Simulation-Based Virtual Machine Generators. In *Reuse and Software Quality*. Springer International Publishing, Cham, 103–119.

[10] Amin Nikanjam, Mohammad Mehdi Morovati, Foutse Khomh, and Houssem Ben Braiek. 2022. Faults in deep reinforcement learning programs: a taxonomy and a detection approach. *Automated Software Engineering* 29, 1 (2022), 1–32.

[11] Alexander Prochnow and Jinqiu Yang. 2022. DiffWatch: watch out for the evolving differential testing in deep learning libraries. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 46–50. doi:10.1145/3510454.3516835

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

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nat.* 550, 7676 (2017), 354–359.

[14] Jiaxue Song, Xiao-Yi Zhang, Paolo Arcaini, Fuyuki Ishikawa, Yong Liu, and Bin Du. 2025. Supplementary Material for the paper "Multi-Agent Differential Testing for the Game of Go". https://github.com/NeNeSakura/MAD4Go.

[15] Jiajun Tan, Dong Wang, Jingyu Sun, Zixi Liu, Xiaoruo Li, and Yang Feng. 2024. Towards assessing the quality of knowledge graphs via differential testing. *Information and Software Technology* 174 (2024), 107521. doi:10.1016/j.infsof.2024.107521

[16] Martin Tappler, Filip Cano Córdoba, Bernhard K. Aichernig, and Bettina Könighofer. 2022. Search-Based Testing of Reinforcement Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 503–510. doi:10.24963/ijcai.2022/72

[17] Thomas Thomsen. 2000. Lambda-search in game trees–with application to Go. *ICGA journal* 23, 4 (2000), 203–217.

[18] Miller Trujillo, Mario Linares-Vásquez, Camilo Escobar-Velásquez, Ivana Dusparic, and Nicolás Cardozo. 2020. Does Neuron Coverage Matter for Deep Reinforcement Learning? A Preliminary Study. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 215–220. doi:10.1145/3387940.3391462

[19] Ulf Wilhelmsson, Wei Wang, Ran Zhang, and Marcus Toftedahl. 2022. Shift from game-as-a-product to game-as-a-service research trends. *Service Oriented Computing and Applications* 16 (2022), 79–81.

[20] David J Wu. 2019. Accelerating Self-Play Learning in Go. *CoRR* abs/1902.10565 (2019). arXiv:1902.10565 http://arxiv.org/abs/1902.10565

[21] Mingyue Zhang, Xiao-Yi Zhang, Paolo Arcaini, and Fuyuki Ishikawa. 2023. An Investigation of the Behaviours of Machine Learning Agents Used in the Game of Go. In *2023 10th International Conference on Dependable Systems and Their Applications (DSA)*. 734–742. doi:10.1109/DSA59317.2023.00105

[22] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. 2023. A Search-Based Testing Approach for Deep Reinforcement Learning Agents. *IEEE Transactions on Software Engineering* 49, 7 (2023), 3715–3735. doi:10.1109/TSE.2023.3269804