Neural Computing and Applications
https://doi.org/10.1007/s00521-018-3701-0

ORIGINAL ARTICLE

Adversarial neural networks for playing hide-and-search board game Scotland Yard

Tirtharaj Dash[1] · Sahith N. Dambekodi[2] · Preetham N. Reddy[2] · Ajith Abraham[3]

Received: 22 November 2017 / Accepted: 10 September 2018
© The Natural Computing Applications Forum 2018

Abstract
This paper investigates the design of game playing agents, which should automatically play an asymmetric hide-and-search-based board game with imperfect information, called Scotland Yard. Neural network approaches have been developed to make the agents behave human-like in the sense that they would assess the game environment in a way a human would assess it. Specifically, a thorough investigation has been conducted on the application of adversarial neural network combined with Q-learning for designing the game playing agents in the game. The searchers, called detectives and the hider, called Mister X (Mr. X) have been modeled as neural network agents, which play the game of Scotland Yard. Though it is a type of two-player (or, two-sided) game, all the five detectives must cooperate to capture the hider to win the game. A special kind of feature space has been designed for both detectives and Mr. X that would aid the process of cooperation among the detectives. Rigorous experiments have been conducted, and the performance in each experiment has been noted. The evidence from the obtained results demonstrates that the designed neural agents could show promising performance in terms of learning the game, cooperating, and making efforts to win the game.

Keywords Deep reinforcement learning · Multi-agent system · Intelligent game agents · Adversarial models · Expert systems

1 Introduction

Game is defined as any activity that has the four traits: goals, rules, feedback system, and voluntary participation [1]. Board games such as ‘Senet,’ ‘Mancala’ and ‘Go’ were invented in the 3000 BC as a form of recreation, and the ability to play a board game well is regarded as the sign of intelligence and learning. Based on the balance of the game [2] during the game play, games can be classified into two different groups: symmetric and asymmetric. In a symmetric game, options are the same for each side of the players. Because of the similar options to play the game, each side of the players continues to apply their part of the strategy chosen from the available options to win the game as soon as possible. This fact makes the symmetric games short, e.g., Ludo [3]. Asymmetric board games are those games in which the players do not stand on the equal ground. Availability of different options provides different advantages and disadvantages to each player. In asymmetric games, the players do not focus more on the short-term strategies rather on the long-term strategy to win the game. Availability of different options and varying strategies makes the asymmetric games more complex, yet interesting to play. However, it should be noted that the asymmetric games can still have statistical balance, even if the options available for each individual side are not balanced against each other.

Sahith N. Dambekodi and Preetham N. Reddy have contributed equally to this work.

✉ Tirtharaj Dash
tirtharaj@goa.bits-pilani.ac.in

Ajith Abraham
ajith.abraham@ieee.org

[1] Department of Computer Science and Information Systems, Birla Institute of Technology and Science Pilani, K.K. Birla Goa Campus, Sancoale 403726, Goa, India

[2] Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science Pilani, K.K. Birla Goa Campus, Sancoale, Goa 403726, India

[3] Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, Washington 98071-2259, USA

A subclass of asymmetric games with imperfect information is the class of hide-and-search games which are played by two parties called hider(s) and seeker(s). The goal of the seeker(s) is to capture the hider(s) in finite time. Many different versions of the hide-and-search game exist which are based on the number of hiders and number of seekers. Available literature suggests that present research in the area of hide-and-search games focuses more on mathematical techniques for finding optimal strategies for the seekers to capture the hider(s) in finite time [4, 5]. In our present research, we investigate adversarial neural network learning approach in playing a hide-and-search board game called Scotland Yard.

Scotland Yard is a popular asymmetric board game which won the prestigious *Spiel des Jahres* (Game of the Year) award in the year 1983. This game uses a hide-and-search mechanism during the game [6] in which the seekers, called detectives, have to determine the location of the mobile hider based on the limited amount of information. The detectives should cooperate in a coalition to solve a variant of the pursuit–evasion problem [7] to capture the hider, called Mister X (Mr. X). Scotland Yard has three properties [8] that make it a very challenging problem to solve using adversarial learning mechanisms: (1) The seekers (detectives) perform public moves while Mr. X can perform both public and private moves. Therefore, imperfect information is available to the detectives. (2) There are five detectives and it requires that all of them should cooperate in some fashion to catch Mr. X. (3) The game is asymmetric, and hence, the game is imbalanced in its goals that make automatic adaptation to the situations harder.

Machines can play the board games either using search-based heuristics or learning-based heuristics. Searching can be carried out in a game tree or a graph, application of which depends on the properties of the game itself [9–11]. Learning-based heuristics have been applied in board games [12] and one of the most successful applications of learning is the application of deep neural network to master the game of Go that also employs tree search for enumerating the possible moves [13]. Other researches suggest that the tree search could be reinforced to make the search more efficient and applicable in reality [14]. Recently, Scotland Yard has been modeled as a search game by using heuristic search algorithms. One of such techniques was attempted in [8, 15] in which Monte Carlo Tree Search (MCTS) technique was used as a heuristic search algorithm. In a different development over the study of this game, a complexity analysis has been performed by Sevenster [16], where it has been shown that the generalized version of the game is PSPACE-complete. Further, in this work, it has also been shown that if the detectives are ignorant of the whereabouts of the Mr. X, the Scotland Yard game is NP-complete. Therefore, it is indeed hard to solve the game of Scotland Yard using traditional search-based heuristics in polynomial time and the problems remain open, thereby making it an interesting problem to study. In a recent survey, it has been discussed that neuro-evolution can be a better alternative to other procedural search-based heuristics [17]. Furthermore, hyper-heuristic, which is possibly a combination of smaller heuristics, could solve the game problems in more efficient way [18]. In this work, we attempted to model this complex and open problem of Scotland Yard by essentially learning the pattern that Mr. X follows during its hiding in the board (map).

It is trivial to state that it is a two-player game (essentially, two-sided) game in which each side attempts to win by applying its part of strategy. The strategy of one side becomes an adversary for another side and vice-versa. If both the sides are modeled using mathematical approximators such as neural networks, the decisive strategy constructed by neural network(s) at one side is adversarial to the decisive strategy constructed by the other side. Such a combination of neural network model could be referred as an adversarial neural network that learns through reinforcement, and therefore, the methodology could be called as deep reinforcement learning. In our present research, we investigate the application of adversarial neural network that also incorporates Q-learning technique to play the Scotland Yard board game. We hypothesize that neural networks with various depths would result in different degree of learning in the behavior of the hider (Mr. X) and the seekers in the Scotland Yard game. Furthermore, we also incorporate the cooperative strategies in designing the adversarial neural networks for the hider (Mr. X) as well as the seekers (detectives) to play the game in finite time. Rigorous experiments have been conducted to obtain the results in the form of win-to-loss ratios. Conceptual inferences have been made from these obtained results.

1.1 Summary of contributions

Our contribution in the present work can be summarized as follows:

- We propose and investigate the applicability of adversarial neural networks for playing a hide-and-search board game called Scotland Yard.
- Two different models of the adversarial neural networks have been developed and rigorously experimented to learn the behavior of Mr. X in the game and further to incorporate the coalition among the detectives.
- Conceptual inferences have been made from the obtained results of the developed models.

- This investigative study could aid in the present research in the applied field of machine learning and hide-and-search game.

The paper has been organized as follows: Sect. 1 introduces the problem and our major contribution in this research, Sect. 2 presents a brief explanation of the Scotland Yard game, the rules of the game, and fundamental motivation behind the use of neural network to design the game agents, Sect. 3 elaborates the methodology proposed and implemented in this research, and Sect. 4 describes the experimental setup and the experimental evaluation of our proposed methods. The work is summarized in Sect. 5.

2 Background and motivation

2.1 Scotland Yard board game

The game of Scotland Yard was introduced in 1983, and the original version was first published by Ravensburger, and the English language version of the game was marketed by Milton Bradley [19]. The Scotland Yard board game is played by 6 players: 5 searchers called detectives and 1 hider, called Mr. X. The game is essentially a two-player game in which the five detectives work as one team to capture Mr. X in a region which has many different routes presented on a map. A graph of a portion of the map used in the Scotland Yard game is shown in Fig. 1.

2.1.1 Rules of the game

There are 199 vertices (nodes) numbered from 1 thorough to 199. The vertices are connected by 4 different kinds of edges that represent different transportation modes: bus, taxi, underground tunnel, and boat. Each vertex can hold at most one player, and each player can occupy at most one vertex at one time. The vertex occupied by a player at a time is called the location of the player at that time. There are 18 predefined vertices where all the six players can start at by randomly choosing one of the 18 vertices. Each detective can have maximum 10 taxi tickets, 8 bus tickets, and 4 underground tickets. Mr. X can have 4 taxi tickets, 3 bus tickets, and 3 underground tickets. The movement of the players is alternating starting with Mr. X. A sequence of moves by all the six players (5 detectives and Mr. X) is called a 'round.' There are maximum 24 rounds in Scotland Yard game.

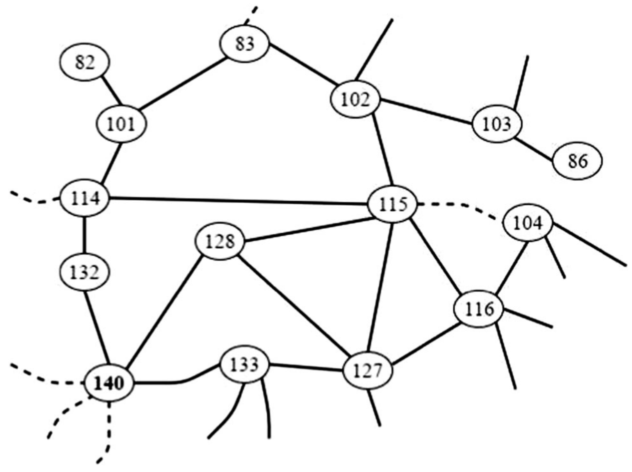

Fig. 1 A graph of a portion of the map used in Scotland Yard board game

Each player looses a ticket based on the type of the chosen edge in moving from one vertex to another. The interesting aspect is that when a detective looses a ticket, it gets added to the ticket of Mr. X. However, vice-versa is not true. When Mr. X plays a ticket, the ticket is permanently removed from the game. This aspect makes this game a (pseudo) zero-sum game in which detectives would want to minimize their expenditure on the tickets and Mr. X would want to maximize the chance of getting additional tickets. It is alright to call this game as a pseudo zero-sum game because when Mr. X plays a ticket, the ticket is permanently lost, whereas this is not the case with the detectives. In addition, Mr. X also has a black fare ticket that allows him to move along any type of edge, including the boat ride. Sometimes, Mr. X may also play one of his double-move tickets, and at that round of play, detectives have to skip their turn.

During the game play, Mr. X keeps his location a secret, and only after moving on rounds 3, 8, 13, 18, and 24, he announces his location. Detectives are well informed about the ticket used by Mr. X in his movement from one vertex to another. Mr. X is captured when any one of the detectives moves onto the node which Mr. X is occupying. Mr. X's goal is to avoid being captured by the detectives until no detective has any ticket left or none of them can perform a move. It should be noted that a detective cannot perform the move when it does not have a valid ticket to move from its presently occupied node.

2.2 Motivation behind the use of neural networks

It has been mentioned earlier that Scotland Yard game has imperfect information. The information that is available is imperfect because of the fact that the detectives are unaware of the location of Mr. X and they are allowed to know the location at some specific times. In our present work, the very first job was to transform this problem of imperfect information into a problem with pseudo-perfect or near-perfect information. Though it is a challenge at the first place, it is quite motivating to investigate the way a learning model perceives this aspect of the game. However, it should be noted that the transformation-based modeling would affect the way a learning model learns during the game play. We tried to imitate the fact that a living human player would play the Scotland Yard game in a different sense which depends on the locations of the players, tokens left and from his own meta-intelligence. A living human being could be biased by his own judgments during the game, and modeling this aspect completely in the machine is very complex to undertake as a task [20–22] in our present work. However, our present version of the modeling does take care of some kind of judgments and cooperations among human beings to make the agents possible to adapt to game dynamics and possibly win the game.

The idea behind the cooperation among the agents in Scotland Yard further fueled after the cryptography experiment [23] conducted by Google, in which Google used 3 neural networks, namely Alice, Bob, and Eve. The experiment included cooperations among the neural networks for encryption–decryption experiments and also demonstrated promising results. Scotland Yard game has the perfect scenario to investigate this fact. The five detectives must communicate and cooperate among themselves to capture the nearly invisible Mr. X.

The Scotland Yard game is incredibly complex, and manually coding all the possibilities would be near impossible, and factually, it would take exponential time as shown in a complexity study of the game [16]. Converting the problem to neuro-solvable or neuro-computable would make the underlying topology more adaptable and flexible in which the game agents have continuous interaction with the present scenario of the environment of the game. As has been demonstrated in the past, the capability of the artificial intelligent agents in a game, a neural net can be incredibly powerful in this aspect, and furthermore, it does not require any human intervention [24–26].

Neural networks have a successful history of being well-studied and well-adapted method for many different real-world applications [27–31]. The neural network model designed here is called ‘adversarial’ because there are two different types of agents, one for Mr. X and other for the detectives. These players—the neural agents—play the Scotland Yard game in an automated setting. They improve their knowledge about the system during the play and continue to improve based on their experiences. Since the adaptation of the neural networks is continuous and there is continual feedback mechanism governing the experience, the learning is recurrent and deeply reinforced. The following section elaborates on the methodology developed in this research.

3 Methodology

In this section, we present the architectural topology of the developed model, the scoring mechanism, the feature space used for the neural agents, and the learning steps.

3.1 Architectural topology

The input to the neural agent is a set of features which represent essentially the presently assessed environment of the game. The output of the neural agent is a move (or, a set of moves) that would be determined from the available inputs. It also includes the essential hidden computation layers which are used to stabilize the learning and adaptation in the neural network. An abstract design of a neural agent designed in this work is shown in Fig. 2.

We have used a Q-learning methodology [32] that also includes an additional querying mechanism. The principal reason behind the use of the querying mechanism is that the final layer of the intended neural network cannot be fixed before hand. Some neurons in the final layer might need to output 2 different valid moves for the player in the question (that is the player for whom the move is to be decided), and other might need to output as high as 8 valid moves. The way to fix this issue was to make the output layer large enough (in terms of the number of output neurons) to have one entry for each valid move possible from every node. Then, one could take a dot product with the list of valid moves for the player in the question and then take its maximum. Instead, we chose to append the contemplated move to the observation and query them together to the model function. The output of the model function would be saved. Such a query would be performed for each valid move for the player in the question. The maximum value of

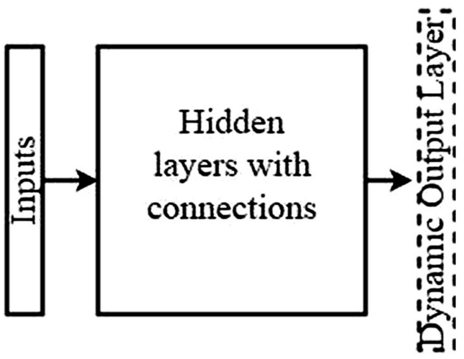

Fig. 2 An abstract design of a neural agent for the Scotland Yard game (‘dynamic’ mentioned for the output layer represents for the dynamic activation of the neurons in the output layer based on the presented features for a specific player. The output layer can be visualized as a dynamically activating layer which would represent the moves)

the output would determine which action the player would take. In this way, we could fix the size of the output layer.

3.2 Action space in the game

The action space is the moving of the unit (Mr. X or detectives) from one node to a node that is immediately accessible (exactly one node jump away) using one of the three methods of transport.

- Subway allows the node jump to cover multiple nodes in between but is only accessible from a smaller selection of predefined nodes.
- Bus can cover a smaller number of nodes per node jump, but has a larger selection of nodes from which it can be accessed.
- Taxi allows the unit to travel exactly through exactly one node per jump, but can be accessed from every single node on the map.

3.3 Scoring mechanism (rewards) in Q-learning

The rewards added to each agent were based on the outcome of the game and the performance of the agent in that particular game. The scoring scheme used is shown in Eq. (1).

$$\text{reward} = \begin{cases} +100 & \text{for a win} \\ -100 & \text{for a loss} \end{cases} \tag{1}$$

We initially started out with a constant reward system. This means that if any detective catches Mr. X, all the detectives are rewarded. However, such a reward system might deter the effectiveness of learning for the detectives during learning the game, because, if a detective wins when it is not chasing Mr. X, he might misinterpret the reward affecting its own learning. Considering this aspect of learning, we modified the scoring system in such a way that the detectives would be more aggressive. In the modified scoring system, a detective is rewarded based on how far he is from Mr. X in terms of number of vertices which follows the proportionality relation,

$$\text{reward} \propto f(\text{dist})^{-1} \tag{2}$$

More specifically, if the distance is more, the reward is exponentially lower when the detectives win. That means, if a detective i is closer to Mr. X than another detective j, it should get higher reward than detective j. When the detectives lose, the absolute value of the negative reward is lower when the detective is closer to Mr. X at the end of the game. The equation used for computing the corrected reward from the final reward is given in (3).

$$\text{reward}_i = \frac{\Delta_{\max} - \Delta_{d_i,X}}{\Delta_{\max}} \times \text{reward}_i \tag{3}$$

where $\Delta_{\max}$ is the maximum path length from Mr. X to the detective i, $\Delta_{d_i,X}$ is the actual distance from the detective i to Mr. X, and reward_i is the reward for the detective i.

3.4 Designing the feature space

In learning the neural agents, the feature space plays the most contributing part in modeling the imperfect information as perfect information. If the feature space is adequate, the model could be robust and highly reliable in terms of its decisive capability [33]. To transform the game with imperfect information to a game with almost perfect information, the following parameters have been considered: (a) number of rounds to the next announcement of the location by Mr. X, (b) the used tokens, and (c) the last known location. All these considered parameters could be monitored even by a human agent, and our detective agent would mimic the same. Any more information than the mentioned would make a biased detective agent, and the model cannot be said to be fully reliable. The detective must learn the game and the behavior of Mr. X given sufficient training simulation and continual recurrent learning.

In our present development on the automatic agents, the location of the hider (Mr. X) is revealed every fifth round and hidden for the rest. Our designed feature space completely reflects the game in a realistic scenario without disclosing any additional details that would make any one of the parties (hider or searchers) stronger than the counterpart. The vertices have been one-hot-encoded and then added to the feature space. Since the vertices are numbered from 1 to 199, each vertex should get a unique feature vector. The feature space used by both Mr. X and the detectives has been depicted as follows.

3.4.1 Feature space for Mr. X neural agent

The neural agent for Mr. X uses the following essential features where the length of the features are given inside the brace. The overall length of the feature vector is $199 + 3 + 995 + 15 + 1 = 1213$.

1. One-hot encoded location of Mr. X (199)
2. Number of tokens (taxi, bus, and underground) left to use by Mr. X (3)
3. One-hot encoded locations of each detective ($199 \times 5 = 995$)
4. Number of tokens (taxi, bus, and underground) still left with each detective ($3 \times 5 = 15$)
5. Round number (1)

3.4.2 Feature space used by the detective neural agents

The detectives have a little more unconventional feature space as shown below.

1. One-hot encoded last revealed position of Mr. X. If Mr. X has not revealed his location yet, a zero vector would be appended. (199)
2. The number of rounds until Mr. X reveals his position next (1)
3. One-hot encoded location of all the detectives ($199 \times 5 = 995$)
4. Tokens still left with each detective ($3 \times 5 = 15$)
5. Tokens used by Mr. X in its last 5 turns ($3 \times 5 = 15$)
6. An additional feature vector based on which kind of model is being investigated.

In this work, we investigated the cooperation among the detectives using two different neural agent architectures: (a) single neural agent for all the detectives behaving as the controlling head of the team and directing the moves of all the detectives (refer Fig. 3) and (b) individual neural agent for each detective (refer Fig. 3). For the former design of the neural agent for the detectives, a one-hot encoded feature vector of length 5 has been appended which essentially denotes the detective number. So, the length of the feature vector for a single neural agent is $1225 + 5 = 1230$ and for individual neural agents, the length is 1225 ($199 + 1 + 995 + 15 + 15$).

3.5 Underlying algorithm

In our present work, the core algorithm is a combination of Q-learning and multilayered neural networks making the model a reinforced learner which continuously finetunes itself by improving with experience during the game play. Q-learning is a model-free reinforcement learning technique [34] which is basically used to find an optimal selection policy for any given finite Markov decision process (MDP) [35]. It operates by leaning an action–value pair function that ultimately gives the expected outcome of taking an action in a given state and by following the optimal policy thereafter. Q-learning is a greedy learner and has been proved to be optimal in terms of the maximum reward returned after all successive steps [32] in the game.

The mathematical model of the problem consists of an agent, a set of states $\mathbf{S}$, and a set of actions in each state denoted as $\mathbf{A}$. Let us denote an action as a ($\in \mathbf{A}$), the agent can move from a state $s_i (\in \mathbf{S})$ to another state $s_j (\in \mathbf{S})$. This action rewards the agent at the state s_i. A reward can be a numerical score. The goal of the agent is to optimize (either maximize or minimize, depending on the problem at hand) the total rewards. To solve this optimization problem, Q-learning offers the agent to choose the optimal action at each state that it reaches. Such an optimal policy would definitely aid to achieve the long-term award in the game and suitable for the Scotland Yard game. The reward is a weighted sum of the expected values of the rewards to be achieved in all future steps starting from the current state s_i, where the weight for a step from a state s_i and Δt steps into the future is computed as $\gamma^{\Delta t}$. The parameter γ is a called the discount factor, which is responsible for a trade-off between sooner and later rewards. γ lies in the range [0, 1] and can be understood as the likelihood to succeed at every future step Δt.

So, algorithmically the state pair transformation function, denoted as Q can be written as

$$Q : \mathbf{S} \times \mathbf{A} \rightarrow \mathbb{R}, \tag{4}$$

where Q generates a real number from the state–action pair. Initially, Q function returns a predefined fixed value. Afterward, each time the agent takes an action a from the state s_i to move to state s_j and the Q is updated to fit more into the realistic scenario which is greedy in some sense. The iterative update equation for Q is given in Eq. (5).

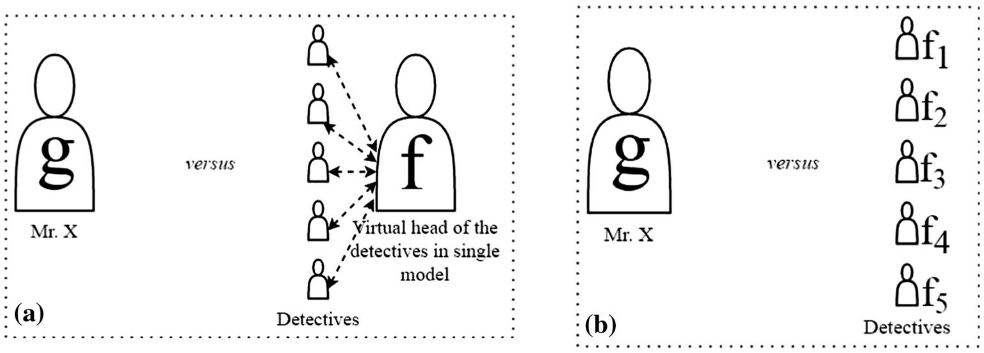

Fig. 3 Modeling of the neural agents in single model and multiple model; a model is nothing but a function approximator (f, g, or f_i) that produces the outcome based on the input features at any given time. **a** Single neural agent for all the detectives virtually behaving as the head of the detectives who instructs them and **b** each detective is a neural agent cooperating with each other

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t.\{r_{t+1} + \gamma.\max_a Q(s_{t+1}, a) - Q(s_t, a_t)\}. \tag{5}$$

Here α_t is the learning rate ($\alpha_t \in (0, 1]$), and r_{t+1} is the reward achieved after performing an action a_t in the state s_t. An episode of the algorithm finishes when the state s_{t+1} becomes the final state where the value of Q is set to 0.

A neural network is used as the function approximation model, which is combined with Q-learning to create the final learning algorithm for training the detectives and Mr. X based on their own feature space and game theoretic perspective. At each step of the game, there is a particular move that a player can make to maximize its chances of winning. In this way, for every possible state of a discrete step game, there will be a certain move. A list of these moves can be tabulated by extensive simulations, forming a lookup table. However, in a game having a very large number of states (such as Scotland Yard), such an approach is computationally not feasible. So, the neural network has been used to approximate the state space. The neural network approximates the optimum action space during the game play. In this approach, the neural network could be trained by a smaller number of iterations (simulations) as it could approximate the optimum actions of previously unseen states, by already learning the states that are similar to the present set of states.

The rewards obtained throughout the game were bootstrapped together and added only at the end of every game. This sped up the learning rate as compared to when the agents were trying to learn at every round. The reward in the present round can be written as

$$\text{reward}[\text{round}_{\text{curr}}] = \text{Reward}[\text{round}_{\text{curr}}] + 0.9^{\beta} \times \text{game\_reward}, \tag{6}$$

where β is the multiplicative factor which can be obtained from the total number of rounds played (N_{round}) and current round ($round_{\text{curr}}$). The game_reward is the score that is based on the outcome of the game.

$$\beta = N_{\text{rounds}} - \text{round}_{\text{curr}} \tag{7}$$

$$\text{game\_reward} = \begin{cases} +100 & \text{for a win} \\ -100 & \text{for a loss} \end{cases} \tag{8}$$

Equation (6) is responsible for reducing the cumulative impact of the moves which have occurred at the beginning of the game on the reward function while simultaneously increasing the rewards at the end of the game. This is done as the moves done at the later stages will be more responsible for deciding the outcome of the game rather than the moves done at the beginning of the game.

A process flow diagram demonstrating the working process of the model is shown in Fig. 4.

4 Experimental evaluation

In this section, the experimental setup and the obtained results have been elaborated.

4.1 Experiment environment

All the simulations have been conducted in High-Performance Computing Cluster system with 14 independent processors, and each machine has 32 GB of main memory. Python along with its relevant packages (Tensorflow [36], pandas [37]) has been used for writing the software. In this experiment, in addition to the discussed methodology, we also investigated various other kinds of game playing agents to play the Scotland Yard board game. A game playing agent in Scotland Yard could be either random move generator or learner (function approximation model). Both these kinds of agents can explore the search space (the map) to take an optimal decision on their movement from one location to another. To activate this characteristic of the agents, we added a probabilistic exploration parameter which ranges from 0 to 1. Since the agents have to learn to play the game, the whole process of the simulation is divided into (a) training phase and (b) testing phase. The testing phase is completely independent, and the agents lack any kind of support (in terms of feedback) during this phase. They explore the map only during the training phase.

In the exploration phase of learning, there are two possible ways the agent can make a decision. The first way is by taking what the agent believes to be the optimal path based on previous experiences. The second way is by taking a random path or ‘exploring’ the map. Both these modes need to be used so that the agent does not only try to win the game, but also at the same time tries to explore the map and figure out relatively newer strategies that were not seen earlier. These modes are selected based on a random probability between 0 and 1. At the start of the simulations, the probability for exploration is set to 1. As the number of games played increases, this probability decreases and finally becomes 0, where the agent is trying to play optimally, thereby making itself fit for the testing phase. During the testing phase, there is no exploration and the model is purely trying to act optimally.

As mentioned earlier, we have used two different kinds of agents such as random move generators and learning-based move generators. The random move generators are the agents that generate random moves given a set of inputs, though there is no relation between a move and the inputs. The learning-based generators or the learners use the function approximation models to generate the moves based on their inputs. Further, the function approximation

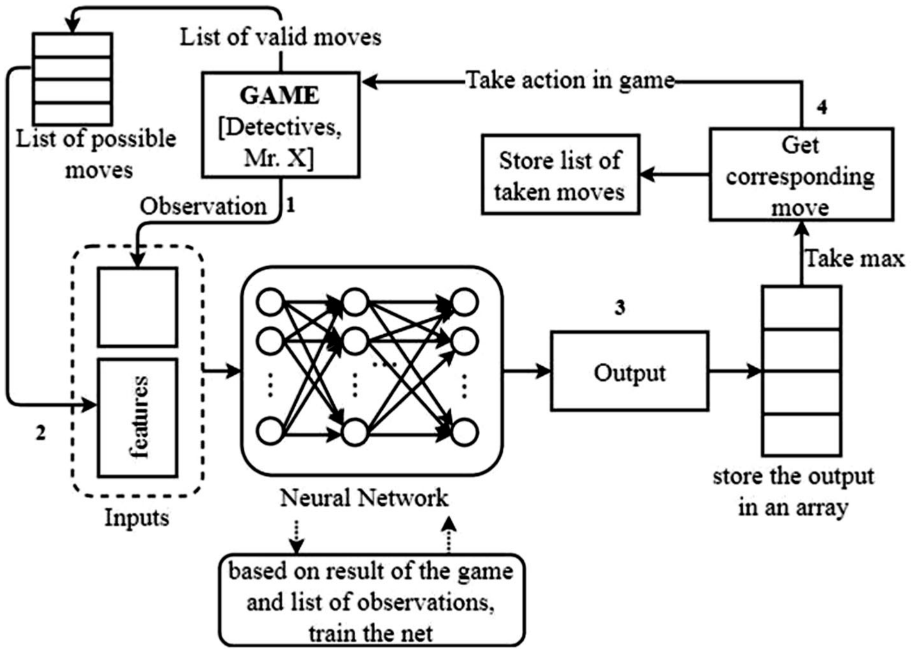

Fig. 4 A process flow diagram of the model, and the numbers are the important milestones: (1) The game decides whose turn it is and selects the particular neural network. The game consists of agents D1–D5 and Mr. X. The game gives out the general observation along with a list of valid moves. (2) Each move is independently appended to the general observation and queried through the neural network. (3, 4) From the list, the maximum action (optimum) is chosen, and the corresponding move is communicated back to the game to make the move. The list is stored as well, to learn from it once the final result is out

model could be single model or multiple models (as shown in Fig. 3). In the case of multiple model-based move generator, the functions f_is are the random move generator. All possible types of gaming agents have been simulated and investigated. In what follows, we describe the purpose of our experiments conducted in this work.

4.2 Purpose of our experiments

We had two possible approaches to design the automatic adversarial game from the viewpoint of the detectives. We needed to figure out a way for them to communicate with each other. With regard to this situation and idea, the architecture could be built as a central hive mind or a collection of neural networks learning to work with each other. The detectives could also be made to move according to a random move generator. Mr. X could either be trained along with the detectives in an adversarial fashion, or the moves of Mr. X could be a preset pattern. The complexity of each network was also varied between each experiment. By combining each of these separate aspects between the detectives and Mr. X, we wished to see which would bring the highest possible improvement. Our experiment purpose was also to see the behavior of the opposition parties given their agent architectures. Each of the seven experiments reflects a different set of parameters to test the built networks. This would help us in understanding the solution better and open new perspective from the machine learning point of view. The results for each of the seven experiments have been depicted in the following subsection.

4.3 Results

The results show the performance of the game during the training and the testing phase. We measure the performance using the parameter given in Eq. (9). ‘Game number’ in the X-axis in testing phase denotes the number of games that have been played so far between the different models. As the number of games increases, the probability for exploration as is lowered and the probability of exploitation is increased. Experiments 1 to 5 are simulated for 25,000 times (0–25,000). Experiment 6 is simulated for 100,000 times, and experiment 7 is simulated for 200,000 times.

$$\begin{aligned}\text{Performance} = {} & \text{Number of games the detectives win} \\ & - \text{Number of game Mr. X wins}\end{aligned} \tag{9}$$

4.3.1 Hyperparameters setting

The hyperparameters related to neural networks and the Q-learning are set as follows. The neural networks are fully connected (i.e., multilayer perceptron) that uses ReLU (rectified linear unit) activation model [38]. Adam (a stochastic gradient descent technique) [39] optimizer was used for training the neural networks with learning rate being set to 0.001. The neural network architectures for Mr. X and the detectives are provided below. Further, the architectural hyperparameters related to the neural networks are explained in each experiment as and when required.

The experiments 1 to 4 have the following neural architecture. For Mr. X, size of the input layer (= feature space of Mr. X + observations from the game = $1213 + (199 + 3) = 1415$); number of hidden layers = 4 ($l_1 : 708, l_2 : 708, l_3 : 354, l_4 : 354$);[1] number of output = 1. For detectives, the input layer size and the hidden layer information are as follows:

- Single model – input layer (feature space + observation from the game = feature space + possible next state + token used) = $1230 + 199 + 3 = 1432$; number of hidden layers = 4 ($l_1 : 716, l_2 : 716, l_3 : 358, l_4 : 358$)
- Multiple models – input layer (feature space + possible next state + token used) = $1225 + 199 + 3 = 1427$; number of hidden layers = 4 ($l_1 : 714, l_2 : 714, l_3 : 357, l_4 : 357$.)

For Q-learning, learning rate (α_t) is set to 1, the discount factor is set to 0.9, and the intermediate reward has been set to 0. The intermediate rewards have been calculated by using the final reward and then extending the calculations for rewards backward through the game.

4.3.2 Experiment 1

In the first experiment, each detective uses its own random move generator; Mr. X uses a single model that learns continuously during the game and generates a move. As mentioned earlier, the learning agent for Mr. X is essentially a fully connected neural network that is combined with Q-learning. However, Mr. X explores the map initially during the training phase and gradually stops exploring and exploits more to get the optimal moves. The number of simulations for experiment 1 is 25,001. This experiment was conducted to test the effectiveness of the random behavior that the gaming agent possesses during the game play. The game performance during training and testing is shown in Fig. 5. It can be observed that detectives are winning more games at the start, Mr. X is exploring the map and both the sides, in essence, making the random moves during their play. However, once Mr. X starts lowering its exploration and increasing its exploitation of the map space, the detectives start losing since they are untrained about the game.

4.3.3 Experiment 2

In this experiment, the detectives use a single model that generates the random moves for the detectives. However, since the moves are randomized, there is no theoretical and computational difference between a random multi-model (Experiment 1) and random single model. The characteristics of Mr. X in this experiment remain identical as in experiment 1 that uses a single learning model combining both neural network and Q-learning. The game performance during training and testing is shown in Fig. 6.

[1] l_i denotes the number of neurons in the layer i.

4.3.4 Experiment 3

In this experiment, the detectives use a single shared model for that learns the game along with exploring the map during the training phase. The architecture of the learner is identical to the one used by Mr. X in our experiment 1 and experiment 2. Similar to experiment 2, the detectives were restricted to only exploit the map space toward the end of their training. Mr. X was allowed to use random moves using a single model. The game performance during training and testing is shown in Fig. 7.

4.3.5 Experiment 4

In this experiment, the detectives use multiple separate models of learning with identical architecture mentioned in our experiment 3 along with exploring the map during the training phase. They were also restricted to only exploit the map space toward the end of their training. Mr. X was allowed to use random moves using a single model. The game performance during training and testing is shown in Fig. 8.

In our randomness experiments, we observed that the detectives were relatively better when they were using a single model for generating their moves. However, this would make the game as if there is a central player controlling all the detectives and moving the pawns. There would be no scope for coordination and communication between the five detectives playing the game. Also, the performance plot demonstrates that, given more simulations, the detectives would start coordinating among each other. Losing games does not directly imply that communication cannot be established. Hence, we decided to still stick with the multi-model variant for detectives during the adversarial neural network experiments. The multi-model agents would provide turn specializations to the players. Furthermore, it was believed that neural network architecture with more number of layers for the detectives would result in better game performance. Hence, the following three experiments were performed using the neural networks. During the training phase, these neural networks would be learning the game, and in the testing phase, they are left to play the game alone without any environmental supports.

4.3.6 Experiment 5

In this experiment, each detective is a neural agent with the fully connected network architecture. The neural networks for the detectives are identical with regard to their

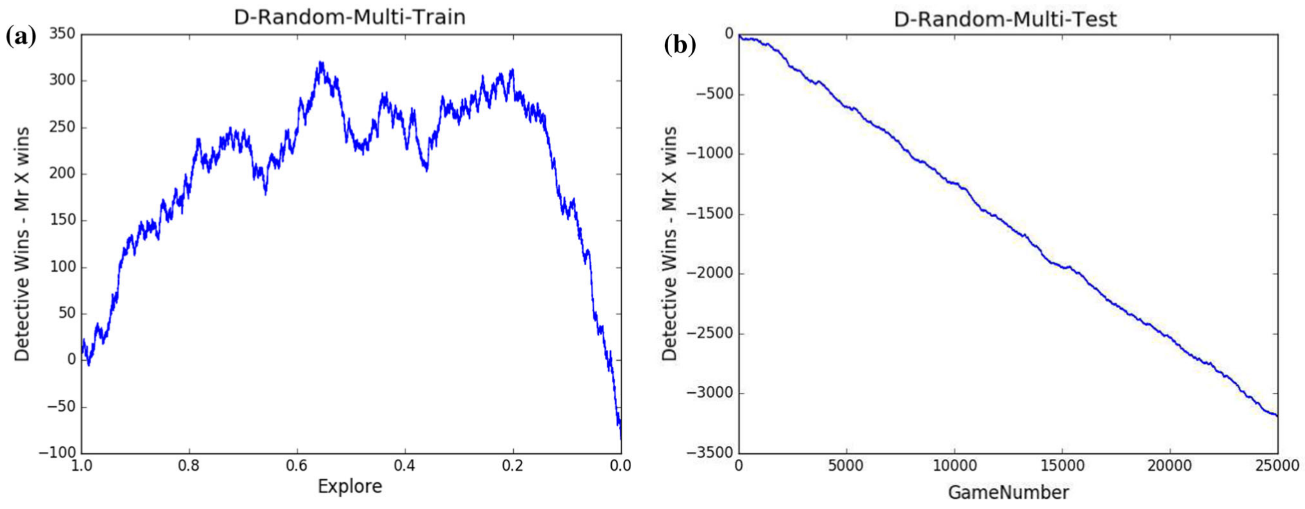

Fig. 5 Game performance for Experiment 1. **a** Experiment 1—training. **b** Experiment 1—testing

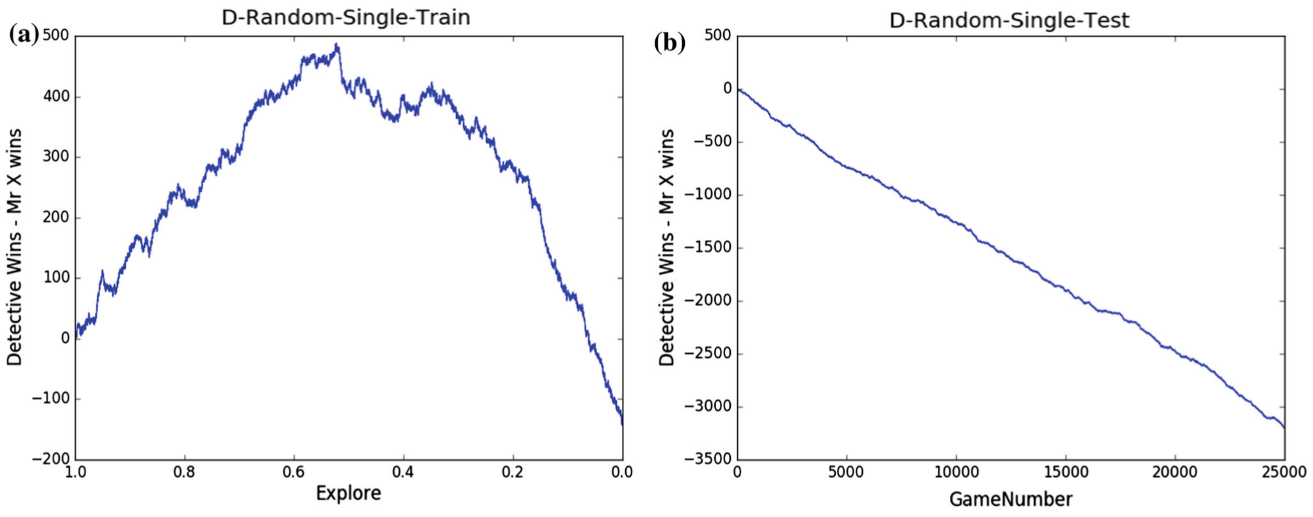

Fig. 6 Game performance for Experiment 2. **a** Experiment 2—training. **b** Experiment 2—testing

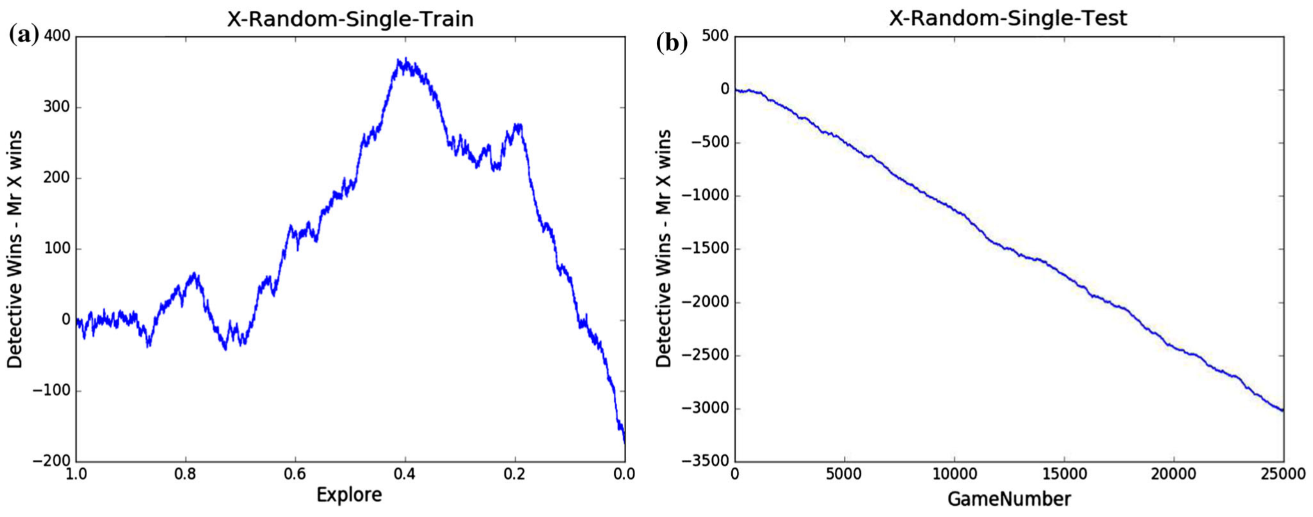

Fig. 7 Game performance for Experiment 3. **a** Experiment 3—training. **b** Experiment 3—testing

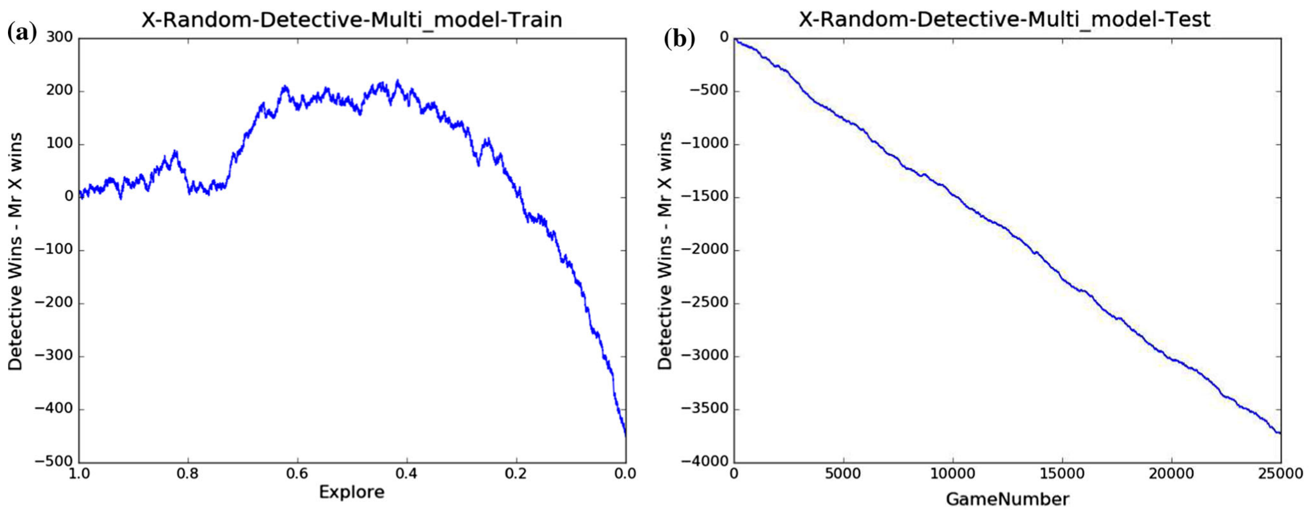

Fig. 8 Game performance for Experiment 4. **a** Experiment 4—training. **b** Experiment 4—testing

architecture which is as follows: input layer size = 1427; number of hidden layers = 8 ($l_{1-4}: 714, l_5: 357, l_6: 357, l_7: 179, l_8: 179$). Further, each detective explores the map at the beginning of the training and gradually reducing their exploration and focusing on exploitation performance. Mr. X was designed with a neural network architecture, Input layer size = 1415; Number of hidden layers = 1 ($l_1: 708$). Mr. X also included exploration of the map and was continuously learning during the training phase. The game performance during training and testing is shown in Fig. 9.

The peak of the training graph exhibits a very interesting characteristic. A positive slope in the plot means at that vicinity the detectives are winning more games than losing, and a negative slope means otherwise. That peak denotes the exploration until which the detectives' policy along with random exploration was better than that of Mr. X. However, in our previous experiments, at exploration set to 0, the detectives were losing as the experience of Mr. X had already increased, which made it stronger. This got translated into a monotonically losing run for the detectives as can be noticed in the previous graph.

4.3.7 Experiment 6

We hypothesized that if the number of simulation is increased and the detectives are equipped with more complex network architecture, they would learn more. Hence, in this experiment, each detective used a deep fully connected neural network architecture with the following architecture: input layer size = 1427; number of hidden layers = 20 ($l_{1-4}: 714, l_{5-8}: 357, l_{9-12}: 179, l_{13-16}: 90, l_{17-20}: 45$). The architecture for Mr. X was identical as mentioned in experiment 5. The number of simulation was increased to 100,000. It was observed that the detectives still had the advantage till exploration = 0.2.

4.3.8 Experiment 7

We conducted an identical experiment like experiment 6 to access the performance of the adversarial neural agents with more number of simulations. Each detective used the identical 22 layered fully connected neural network architecture as explained in experiment 6. The architecture for Mr. X was identical as mentioned in the former experiment. The number of simulation was doubled to 200,000. The testing phase performance is observed to be quite interesting in this experiment (please see Fig. 11b). The figure suggests that the testing is improved with more number of simulations giving the neural agents for detectives a chance to make themselves smarter with time, because of the fact that the exploitation performance is better than the one seen in experiment 6. Observing the training performance toward the exploration of between 0.2 and 0 in Figs. 10a and 11a, it can be said that the increased simulation number does improve the quality of learning in the detective agents.

4.4 Summary of experimental results

A summary of experimental results and the performance of the game playing agents during training and testing phase are depicted in Table 1. Further, the relative game score for each experiment during training and testing is viewed in Fig. 12. The comparative plot shows the improvement in the neural agents with the increase in the number of simulations. However, the performance during training is better than the performance observed during testing. The experiment 5 to 6 could demonstrate promising

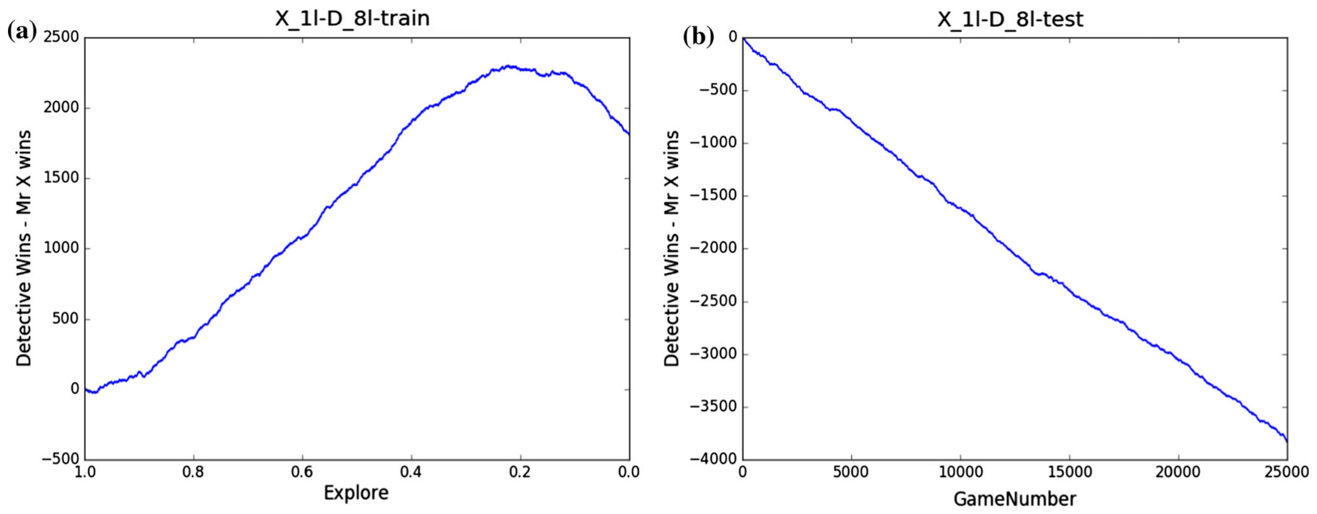

Fig. 9 Game performance for Experiment 5. **a** Experiment 5—training. **b** Experiment 5—testing

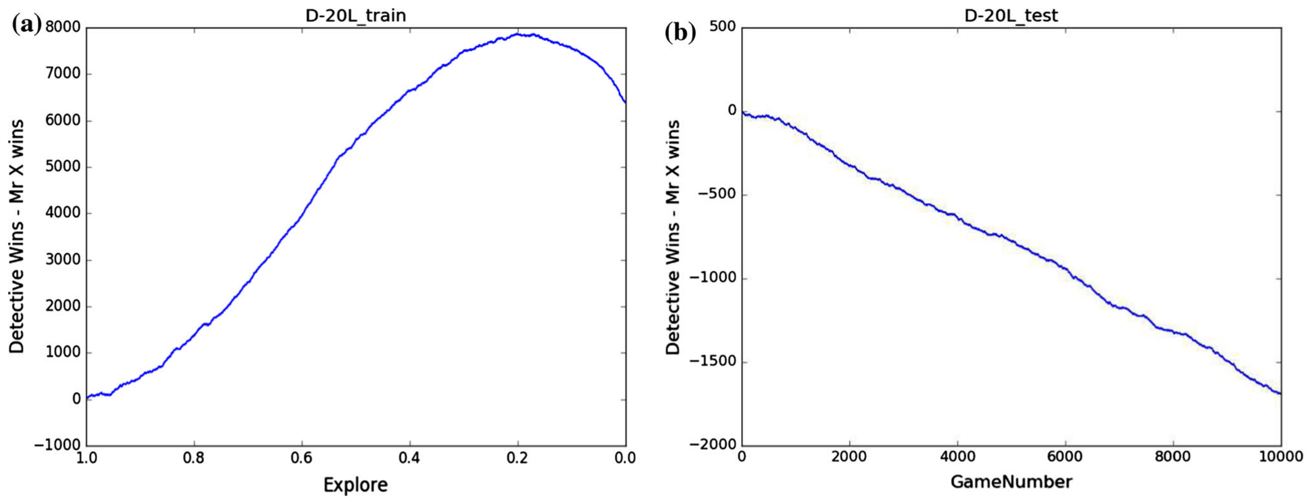

Fig. 10 Game performance for Experiment 6. **a** Experiment 6—training. **b** Experiment 6—testing

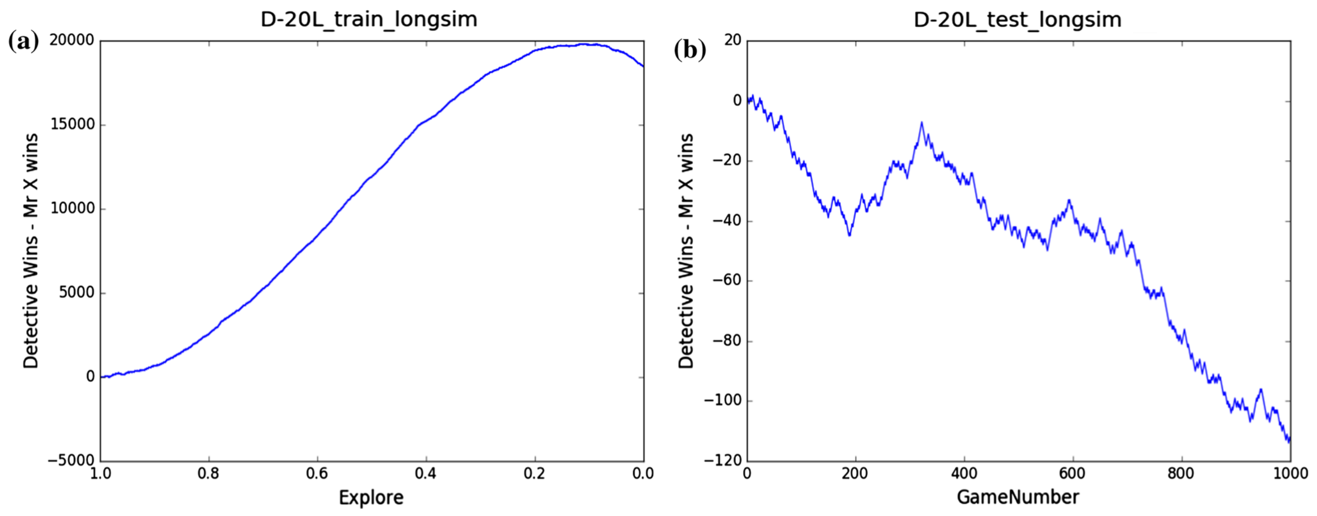

Fig. 11 Game performance for Experiment 7. **a** Experiment 7—training. **b** Experiment 7—testing

Table 1 Summary of experimental results with scores for detectives and Mr. X and the performance (score difference) in each experiment. '+' game score represents that the detectives are better than Mr. X, and '−' game score represents that Mr. X is better than the detectives

Experiment	Training phase scores			Testing phase scores		
	Detectives (searchers)	Mr. X (hider)	Score difference	Detectives (searchers)	Mr. X (hider)	Score difference
Experiment 1	12,458	12,543	− 85	10,906	14,095	− 3189
Experiment 2	12,429	12,572	− 143	10,899	14,102	− 3203
Experiment 2	12,414	12,587	− 173	10,993	14,008	− 3015
Experiment 4	12,275	12,726	− 451	10,642	14,359	− 3717
Experiment 5	13,404	11,597	+ 1807	10,584	14,417	− 3833
Experiment 6	53,194	46,807	+ 6387	41,621	58,380	− 16,759
Experiment 7	109,235	90,766	+ 18,469	444	557	− 113

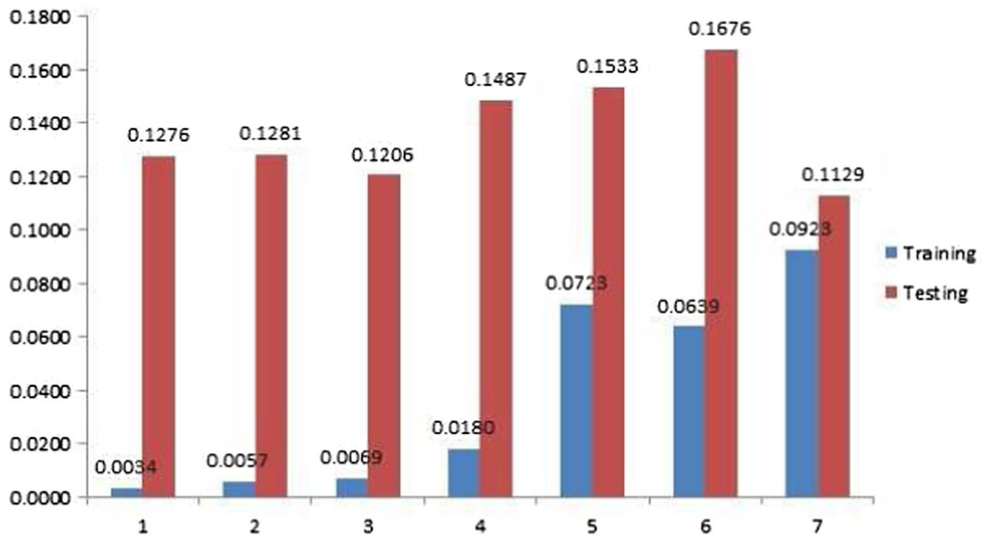

Fig. 12 Relative game performance with regard to the number of simulation during training and testing (*X*-axis: experiment number, *Y*-axis: game score/number of simulations; legends: first bar—training, second bar—testing)

improvement in the relative game performance during the training and the testing phase. Few of the experiments such as experiment 2 and 3 use a shared detective model that in essence makes it a single (detective) player playing five different moves—one for each detective. The rest of the experiments uses multiple detectives (multiple opponents), and via episodic learning these players optimize their set of move-based various environmental inputs during the game playing. One way to understand this setting is to view a multiple opponent game in which the opponents are communicating among themselves in search for a solution in the complex domain. However, this property is automatically realized by the neural network models for the detectives. Since this is a two-sided game and strategy of each player influences the strategy of another while the game is being played, there should exist Nash equilibria during this game play.

Furthermore, the following inferences could be made from the experimental results that are presented in the figures. The training curve for all the experiments conducted in this research followed a similar pattern. Detectives always started out by winning larger number of games when the exploration rate was high. This eventually saturates by the time game progresses till the exploration rate is 0.2 and then starts decreasing. However, this saturation point was much later when there were more number of layers for the detectives' neural networks and also when Mr. X was not making random moves as these are difficult for the detectives to predict, which is highly adversarial even for human experts. The training was ultimately not stable for every case involving random Mr. X as the detectives would not be able to completely learn the random moves without a deeper and complex enough network architecture and would ultimately started failing in training much earlier. When Mr. X has a neural network, the training is much more stable as the detectives are able to better learn the pseudo-random movements of Mr. X by adapting to the predictions proposed by the Mr. X's neural engine. However, even with a simpler network, Mr. X has too large of an advantage due to its movements being hidden. This is why the detectives will always start losing when they take optimum movements, that is, when the exploration rate becomes low. However, as we increase the number of layers and the complexity of the detective networks, the saturation point where detectives start losing more than they win becomes closer to exploration rate 0. We can infer with more layers the detectives become better at making optimum movements to catch Mr.X.

5 Conclusion and future directions

In this work, a thorough investigation has been carried out to test and validate the design of game playing agents, which should automatically play a hide-and-search-based board game called Scotland Yard. The agents are nothing but a hyper-heuristic combined with deep neural network and Q-learning strategy. Rigorous experimental evaluation has been done and the obtained results suggest the following important observations:

- Given more simulations, the detectives can gain an advantage. However, in our present experimental setup, it was not possible to increase the number of simulation or the depth of the neural network beyond the mentioned values because of the limitations in computational resources. This aids to the weakness of our present research by limiting us to do the statistical validation.
- However, with a 22 layered deep neural network with 200,000 simulations was able to exhibit remarkable performance in which the detectives' win is comparably far better than Mr. X's wins. This result further increases interest in exploring the possibility of deep adversarial learners for the game and tests whether our hypothesis in the aforementioned point could be validated.
- The number of simulations required cannot be predecided, but should be sufficiently large given the state space of the problem.
- It is difficult to exactly identify the communication between the detectives. It was observed as if the detectives are predicting each other's moves. However, this is not entirely true.
- With the shared reward system designed in this work, the detectives can learn each other's tendencies and change their own behavior to have a bigger impact on the team's success.
- This shared reward system is the channel through which the detectives communicate.

There are very limited research on the automated solution for the Scotland Yard game, and a direct comparison of our present work with the available literature (in [8, 15]) would not be relevant and meaningful.

In future, to enhance the performance, a better communication system between the detectives should be designed or the presently designed model should be improved in terms of the model complexity along with attempts made to reduce the game learning time (number of simulations). Again, statistical validation should be attempted to compute an estimate of the game performance for various number of simulations. Further, this problem could also be modeled as a solution for traffic problems, with more detectives as motor vehicles; one can define conditions for traffic congestions and optimal control.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interest in relation to the publication of this article.

Appendix 1: Some additional details

Scotland Yard: the implemented version

In our present work, we implement the original Scotland Yard board game. This might bear some similarities to a game with same name that is played in Tokyo (Scotland Yard Tokyo), and in Switzerland (Scotland Yard - Swiss Edition). For additional clarification, we again provide information on the version that is being implemented in this work. There are two teams in this game. The two teams are of Mr. X. (alone) and 5 Detectives out to catch him. Mr. X. moves covertly and his position is revealed only 5 times throughout the entire game. The detectives must cleverly corner and capture Mr. X by the end of their game. Each location on the map is a node, numbered from 1 to 199. To travel between nodes, a player can use one of 3 transportation mechanisms: taxi, bus and subway, each of which has different connectivity across the map. Each of these services can be availed using tokens given to the players at the start of the game. We have not used double turns and waterways in our implementation of the game. What makes the game more interesting is that after each move by a detective, the detective would hand its used token used to Mr. X, who can use it himself in his covert movements. The overall result is more overlap between the two parties, and the detectives being able to affect the game in more than one way.

Q-learning: configuration

We have used the following configuration for the Q-Network. The detectives and Mr. X move using a static network until the turn is done. Each of their moves is stored in the replay memory. At the end of the game, the appropriate reward is given according to which a team won (either Mr. X or detective side). After the game is done and during the learning phase, the reward is back-propagated through each move made during the entire game. As we move backwards through the memory, the reward is diminished exponentially. In this way, greater weight is given to the late game turns.

Exploration is linearly decreased from 1 to 0 (as shown in above figures) depending on the total number of turns in the game. Exploration is defined as the probability that the agent will take a random move instead using the network.

The reward system is $+100$ for a win, -100 for a loss. For the detectives, the reward is calculated dynamically based on the distance of the detective from Mr. X when the game is over. The closer a detective is to Mr. X, during a win it will have a higher fraction of the $+100$ reward, and for a loss it will have a lower fraction of the -100 reward.

References

1. McGonigal J (2011) Reality is broken: why games make us better and how they can change the world. Penguin, London
2. Peled N, Kraus S (2015) A study of computational and human strategies in revelation games. Auton Agents Multi Agent Syst 29(1):73–97
3. Kroll M, Geiß R (2007) A ludo board game for the agtive 2007 tool contest. http://www.informatik.uni-marburg.de/~swt/agtive-contest/Ludo-Karlsruhe.pdf. Accessed 23 June 2017
4. Adler M, Räcke H, Sivadasan N, Sohler C, Vöcking B (2003) Randomized pursuit-evasion in graphs. Comb Probab Comput 12(03):225–244
5. Sujit PB, Ghose D (2004) Multiple agent search of an unknown environment using game theoretical models. In: American control conference, 2004. Proceedings of the 2004, vol 6. IEEE, pp 5564–5569
6. Subelman EJ (1981) A hide-search game. J Appl Probab 18(03):628–640
7. Isaacs R (1999) Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization. Courier Corporation, North Chelmsford
8. Nijssen JAM, Winands MHM (2011) Monte-carlo tree search for the game of Scotland Yard. In: IEEE conference on computational intelligence and games (CIG). IEEE, pp 158–165
9. Coulom R (2006) Efficient selectivity and backup operators in Monte-Carlo tree search. In: International conference on computers and games. Springer, pp 72–83
10. Luckhart C, Irani KB (1986) An algorithmic solution of n-person games. In :AAAI, vol 86, pp 158–162
11. Sturtevant NR, Korf RE (2000) On pruning techniques for multi-player games. AAAI/IAAI 49:201–207
12. Thrun S (1995) Learning to play the game of chess. In: Advances in neural information processing systems, pp. 1069–1076
13. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489
14. Kocsis L, Szepesvári C (2006) Bandit based Monte-Carlo planning. In: European conference on machine learning. Springer, pp 282–293
15. Nijssen P, Winands MHM (2012) Monte carlo tree search for the hide-and-seek game scotland yard. IEEE Trans Comput Intell AI Games 4(4):282–294
16. Sevenster M (2006) The complexity of scotland yard. In: van Benthem J, Löwe B, Gabbay D (eds) Interactive logic, pp 209–246
17. Risi S, Togelius J (2015) Neuroevolution in games: state of the art and open challenges. IEEE Trans Comput Intell AI Games 9:25–41
18. Li J, Kendall G (2015) A hyper-heuristic methodology to generate adaptivestrategies for games. IEEE Trans Computat Intell AI Games 9:1–10
19. Scotland Yard (2016). https://www.boardgamegeek.com/boardgame/ 438/scotland-yard. Online; accessed 3 Nov 2016
20. Turing AM (1956) Can a machine think. World Math 4:2099–2123
21. Wang D, Subagdja B, Tan A-H, Ng G-W (2009) Creating human-like autonomous players in real-time first person shooter computer games. In: Proceedings, twenty-first annual conference on innovative applications of artificial intelligence, pp 173–178
22. Schrum J, Karpov IV, Miikkulainen R (2011) Ut$\hat{2}$: human-like behavior via neuroevolution of combat behavior and replay of human traces. In: IEEE conference on computational intelligence and games (CIG). IEEE, pp 329–336
23. Abadi M, Andersen DG (2016) Learning to protect communications with adversarial neural cryptography. arXiv preprint arXiv:1610.06918
24. Richards N, Moriarty DE, Miikkulainen R (1998) Evolving neural networks to play go. Appl Intell 8(1):85–96
25. Lin W, Baldi P (2008) Learning to play go using recursive neural networks. Neural Netw 21(9):1392–1400
26. Clark C, Storkey A (2015) Training deep convolutional neural networks to play go. In: International conference on machine learning, pp 1766–1774
27. Jetal Hunt K, Sbarbaro D, Żbikowski R, Gawthrop PJ (1992) Neural networks for control systems—a survey. Automatica 28(6):1083–1112
28. López de Lacalle LN, Lamikiz A, Sánchez JA, Arana JL (2002) Improving the surface finish in high speed milling of stamping dies. J Mater Process Technol 123(2):292–302
29. Ho W-H, Tsai J-T, Lin B-T, Chou J-H (2009) Adaptive network-based fuzzy inference system for prediction of surface roughness in end milling process using hybrid taguchi-genetic learning algorithm. Expert Syst Appl 36(2):3216–3222
30. Arnaiz-González Á, Fernández-Valdivielso A, Bustillo A, López de Lacalle LN (2016) Using artificial neural networks for the prediction of dimensional error on inclined surfaces manufactured by ball-end milling. Int J Adv Manuf Technol 83(5–8):847–859
31. Swain RR, Khilar PM, Dash T (2018) Neural network based automated detection of link failures in wireless sensor networks and extension to a study on the detection of disjoint nodes. J Ambient Intell Hum Comput. https://doi.org/10.1007/s12652-018-0709-3
32. Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8(3—-4):279–292
33. Reddy PN, Dambekodi SN, Dash T (2017) Towards continuous monitoring of environment under uncertainty: a fuzzy granular decision tree approach
34. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction, vol 1. MIT press, Cambridge

35. Puterman Martin L (2014) Markov decision processes: discrete stochastic dynamic programming. Wiley, Hoboken
36. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M et al. (2016) Tensorflow: large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467
37. McKinney W (2011) pandas: a foundational python library for data analysis and statistics. In: Python for high performance and scientific computing, pp. 1–9
38. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp 807–814
39. Kingma D, Adam JB (2014) A method for stochastic optimization. arXiv preprint arXiv:1412.6980