Discovering of Game AIs' Characters Using a Neural Network based AI Imitator for AI Clustering

Zhou, Yushan
School of Electronics Engineering and Computer Science
Peking University
Beijing, China
zysls@pku.edu.cn

Li, Wenxin
School of Electronics Engineering and Computer Science
Peking University
Beijing, China
lwx@pku.edu.cn

***Abstract*—In game AI research, most work aims at building a more powerful AI but few are reported on explaining AI's behavior or revealing the characters of AI. Previous efforts in training a human-like AI via style learning implies that AI may behave with human characteristics. However, the early work treated human playstyle as a whole instead of identifying the difference among various AIs. In this paper, we focus on finding out manlike characters of individual AIs, and clustering AIs according to their characters. We propose a Neural Network based game AI imitator to imitate AIs' behavior and find that some AIs are easier to imitate than others. Based on this observation we define the term *imitability* to describe the difficulty of imitation and cluster the AIs into two categories according to their imitability. Through statically analyzing, we find that AIs with lower imitability are generally farseeing with a global perspective while the other group are nearsighted and narrow-minded. The AIs hard to imitate also perform better when fighting with others. Upon the above semantic analysis of the clustering results, we conclude that the imitability can be used to identify AIs' character.**

***Index Terms*—game AI, explainable AI, characters, style learning, neural network**

I. Introduction

With inspirational development of game AI like AlphaGo and Suphx, they greatly attract public attention because of their superhuman performance [1]–[3]. While outperforming humans significantly, these AI are black boxes, which confuses us which playing style we should classify it as. What's worse, humans always fear the unknown especially those without emotion resonance. To break the barriers between humans and AI programs, it's of significant importance to explore the character, i.e., playing style, that game AI behaves in game playing and make the game AI easy, intuitive, and vivid to understand.

Concretely, exploring game AI's characters helps discover various ways of game playing. AIs with distinct search algorithms differs in building search tree, which contributes to diverse behaviors that can be captured as AIs' characters. The calculation process of AI resembles thinking in human, e.g., AI who only uses the current information is too short-sighted to consider the future, and some farseeing one chooses action with most feedback through searching game tree. Previous efforts in style learning of AI mostly focused on training a human-like AI, which implies that AI may behave with human characters. It also benefits the gameplay video generation [4]–[6], that customizing spectator-specific gameplay based on persona research reduces difficulty and saves time in developing.

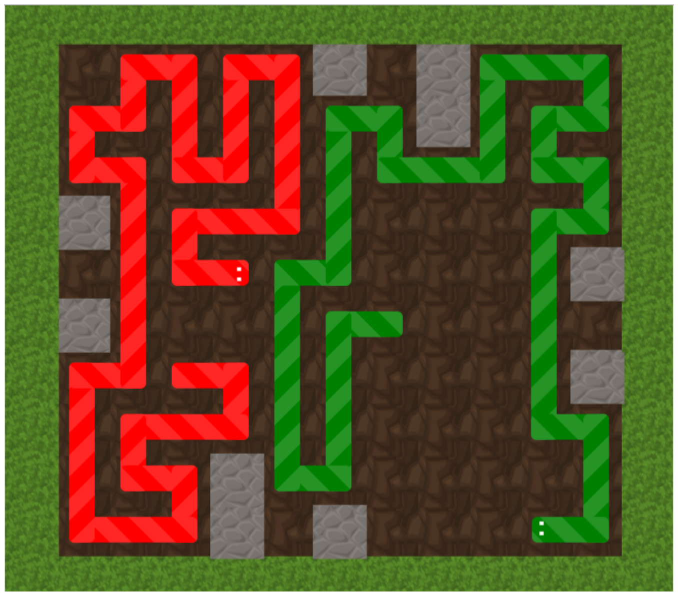
Fig. 1. A screenshot of *Snake* on the online multi-agent platform, Botzone. The grids surrounding the field are walls. The greys are stones, the same as obstacles in logs. The green connected snake and the red one are controlled by the players, whose heads are specially marked with two dots. Other grids are blank.

In this paper, the term *bot* is used to refer to an AI program. An AI imitator is a neural network mimicking AI program's decisions. The game we use in this paper is a variant of traditional game called *Snake*. As there's exactly one game for each match in the *Snake* game, we use the term *match* in places of *game*. We define the term *IMI* (imitability) to quantitatively describe the difficulty of imitation and cluster the AIs into two categories according to their imitability.

For most game AI, searching on a game tree is a common technique to find the optimal solution. The size of game tree is typically vastly larger than the state space of the problem, and each node in the tree is a reachable state, which means that exhaustive search is almost impossible within time and space limit when the size of tree is enormous. Some search techniques are used to narrow down the search to several branches and find an approximate solution. Common search algorithms in game AI differ in range and order of search. Apart from the algorithm structure, another essential component of the game AI is value function, which evaluates each state and determines whether the move is good or bad.

Different search order in algorithms and diverse information utilized in value function may account for diverse playing style in game AI. Learning about bots' search order and pre-defined weights in value function, i.e., their preferences, is a heuristic approach to analyze bots' characters. With the limitation of time, bots may have strong preferences in their search order, which is reflected in the generation of the unbalanced game tree. The pre-defined weights in the evaluation function also influence the final decision making. However, without access to their source codes, we can hardly build a clear cognition of how these bots are organized even if we know the algorithm structures. For example, the parameters in AlphaGo, Suphx and Deep Blue remain undisclosed. It is difficult to discover the AI's characters in lack of intuitive methods of studying AI behaviors. Therefore, it's natural to come up with the idea of imitating the thinking and calculating process of AI. More directly, we can train an AI-style bot to clone the AI's behaviors in different situations.

In the field of style learning, music and text attract our attention. Different music pieces have distinct characters. We can distinguish them by comparing the elements of music (such as pitch, tonality, and rhythm, etc.) at a low level, and recognize the style at a high level. It is the same in text, we can use the low-level characters (like the use of rhetoric) to summarize the high-level style. We are convinced that behavior characters of AI in game playing can also be captured to classify the playing style. Especially, the characters of game AI discussed in this paper refers to the objective behavior but not the subjective feelings or emotions. It is not easy for us to discover whether the bot will be anxious and make random or aggressive decisions like human players when it is about to lose. Also, it is difficult for the programmers to create a game AI to guess the opponent's thoughts and change its strategy. However, we can capture the behavioral characters like taking a flanking tactic to surround the enemy, snatching areas as more as possible, and taking aggressive/conservative strategy when both sides encounter. A reasonable assumption is that bots with a distinctive playing style differ in behavioral characters because of different algorithms and diverse parameters.

Under this assumption, we conducted experiments based on the *Snake* game on Botzone, which is a designed to evaluate implementations of different game AI by automatically scheduling matches for them under the ELO rating system. Their execution environment has limits in memory and time, enforcing users to write time-efficient and space-efficient programs to balance exploration and exploitation.

We selected 22 bots from the ELO rank list according to the algorithms they used. There are 194 bots in total. We selected 2 NE (Nash Equilibrium) bots, 3 MCTS (Monte Carlo Tree Search) bots, 4 MC (Monte Carlo) bots, 5 Alpha-Beta bots, 4 DFS (Depth First Search) bots and 4 ES (Expert System) bots randomly from the list. Especially, the MC bots and the MCTS bots expand the successors randomly in game tree, while the others not. None of the bots makes random decisions. We downloaded the public matches from the platform and focused only on the selected bots, then we recovered the gameplay states to produce the datasets used to train an AI imitator cloning the AI's decisions.

It is necessary to mention that the *Snake* on the platform is adapted into a two-player turn-based game, with a map ten grids long and eleven grids wide showed in Fig. 1. Ten stones are placed symmetrically and their coordinates will be generated randomly before the match begins. The players make decisions simultaneously, with an action set consisting of North, East, South, and West, encoded from 0 to 3. Different from the traditional snake game, snakes grow on certain turns only. In the beginning, both snakes have a body length of one grid and grow every turn. After ten turns, the frequency decreases to once every 3 turns. The player loses once the snake collides with obstacle or any snake' body, or the bot performs an invalid action. The other one will be the winner in this match. Although the size of action set is 4, the actual number of valid actions for each state is no more than 3. The snake can't head back to bite itself, and its head is connected to the body. Considering that there is no action as standing still, i.e., not moving, a natural strategy to win is surrounding the opponent to make invalid action, e.g., biting the snake body, the wall, or the stones. In Table I, we show the *Snake's* state-space complexity and game-tree complexity along with other games summarized in [7]. Especially, *Snake* is a simultaneous decision making game, and we calculate its game-tree complexity by treating its branch factor as the number of action pair.

TABLE I
STATE-SPACE COMPLEXITIES AND GAME-TREE COMPLEXITIES OF VARIOUS GAMES

Game	State-space compl.	Game-tree compl.
Checkers	10^{21}	10^{31}
Othello	10^{28}	10^{58}
Chess	10^{46}	10^{123}
Chinese Chess	10^{48}	10^{150}
Snake	10^{54}	10^{122}
Hex	10^{57}	10^{98}
Shogi	10^{71}	10^{226}
Go(19×19)	10^{172}	10^{360}

In the following section, we present the related work, and compare their targets and methods with ours. In Section III, we elaborate a method of clustering the AI programs by evaluating their imitability using universal AI imitators. In Section IV, we give the concrete experiment settings and conduct static analysis on the bots to demonstrate the feasibility of clustering. We read bots' programs to give the cluster semantic characters to get stronger evidence. Finally, in Section V, we summarize the work, analyzing the advantages as well as shortcomings, and discuss the future work.

II. Related work

A. Behavioral Cloning in Imitation Learning

Imitation learning focuses on learning to utilize experts' demonstrations to complete a certain task. Torabi et al. [8] split the research in imitation learning into two main categories, (1) behavioral cloning and (2) inverse reinforcement learning. The former directly maps the states to the actions in supervised learning methods, while the latter involves reinforcement learning [9]. In our work, we use the behavior cloning methods to train an AI imitator to exploit characters of bots' playing style, and the bots chosen are not all expert-level.

The early work of imitation in game AI mainly concentrated on how to make the AI behave like a human, or how to produce human-style non-player characters (NPCs) as opponents or mates in games [10]–[13]. Ortega et al. [10] compared hard-coded method and three types of neural networks by mimicking the human players' gameplay across levels in Super Mario Bros . Their objective was training a human-like, believable and enjoyable AI in game playing. Devlin et al. [12] combined the traditional Monte Carlo Tree Search algorithm with human gameplay data, adding weights to the rollout decision, making AI more human-like while maintaining good performance as before. They made a strong but unrealistic assumption that there was only one human playstyle in the game, then they amalgamated the whole gameplay data into one model. In the extended experiment, they clustered individual players' behavior and discovered four distinct clusters, but failed to define the clusters qualitatively. Ishii et al. [5] introduced a new evaluation function into a variant of MCTS called Puppet-Master MCTS to let AI behave humanity. Holmgård et al. [11] modeled player decision making styles based on persona and clone methods.

Our work is different from theirs on the objective, for we investigate the imitability of AI programs and try to recognize the playing style of distinct AI with a semantic description, but not human-like or human-style learning. We address the imitation learning problem through supervised learning methods, using neural networks to imitate the AI program's responses. Besides, there is seldom clear classification of AI playing style in previous work, and it is innovative to give a semantic meaning to the clustering results of the AI programs.

B. Music Style Learning

Music style learning is one of the mainstream in music information retrieval researches. The similarity between the work using unsupervised methods and ours is that we both try to give a semantic description to the clustering results. Because of the nature of music, the style is influenced by the custom, the tone, etc., which inspires the researches in discovering the relationship between the music clusters and genres, and recognizing the music's style.

To cluster the music recordings, the similarity (or distance) must be defined. Tsai and Bao [14] represented music recordings in the same genre as Gaussian mixture model, and computed what is the likelihood that a music recording belongs to the genre. Cilibrasi et al. [15] used the compression program bzip2 to approximate the distance between music pieces and distinguished them between various musical genres (classical, jazz, rock). Kopiez et al. [16] and Cilibrasi et al. [17] both used the best informative compressors to categorize musical style and found the cluster method helps authorship detection. They claimed that the best compression rate of a data sequence was related to the self-similarity of the sequence and then to its complexity.

Compared to the methods above, our work differs in the measurements of similarity but is similar in the evaluation of specific properties of the objects. Music style learning calculates entropy on music pieces while we perform the evaluation of the AI's imitability based on match log.

C. Cluster-based Text Sentiment Analysis

Text Sentiment Analysis is widely used in public opinion collections. Feng et al. [18] extracted blogs on common emotions with fine-grained sentiment clustering, extending the coarse level with a classification of "positive", "negative" and "natural". Eyben et al. [19] clustered the combination of the text and speech to improving expressiveness for speech synthesis systems.

Similar to music, text reflects the creators' characters (strict or lively) and stimulates emotions in the audience through rhetoric (like parallelism, irony, etc.). The text style is more like a sentiment classification like "happy", "sad", etc. Different from that, the playing style of game AI we discuss in this paper is not related to the emotions but the behavioral characters when making decisions, e.g., taking a flanking tactic to surround the opponent's snake body, or changing its strategy to a conservative one when two snakes' heads encounter.

III. Proposed method

In this section, we elaborate the method of clustering the bots on their imitability measured by the performance of the neural-network-based AI imitators. To cluster the bots into different styles, there are several questions to be addressed: (1) the form of data on which we prepare to cluster, (2) the linguistic meanings that the clustering results can be explained, and (3) the clustering approach and the evaluation of the clustering results.

To answer the first question, we did not cluster on the matches data directly, but on the performance of the imitators. Considering the need for quantitative evaluation of imitability in clustering, we defined the term IMI (imitability) to denote the performance of the networks which reflects the degree of imitation, particularly the validation accuracy.

The second and the third question both need the verification proof on the clustering results. The clustering of music style by genres inspires us that the algorithms and the information utilized in the value function probably correlate strongly with the clustering results. We conducted static analysis on the bots' code, then we tabulated the search depth and the information used in the value function (see Table II). The search depth and the value function will not be the input to the networks, but we will use them to analyze bots' characters.

A. Data Collection and Preprocess

We selected various bots using different algorithms from the ELO rank list of *Snake* game on the online multi-agent platform, Botzone. The platform uses an ELO system to rate the relative ability of AI programs. We downloaded the public matches from the platform and focused only on the selected bots, then we recovered the gameplay states to produce the datasets used to train an AI imitator cloning the AI's decisions.

We used N_b to represent the number of bots, and ranked them by ELO score with sequence number from 1 to N_b. For each bot, we replayed the match from logs to generate state-action pairs, which resulted in N_b datasets. A sample of match log is shown below. Finally, we sampled pairs randomly in each bot's dataset, to keep the sizes of N_b dataset the same.

```
{
  _id: ObjectId, // match ID
  initdata: {
    width: Number, // map's width
    height: Number, // map's height
    0: { x: Number, y: Number }, // player 0's coordinate
    1: { x: Number, y: Number }, // player 1's coordinate
    obstacle:[
      { x: Number, y: Number },
      { x: Number, y: Number },
      ...
    ] // obstacles' coordinates
  },
  players: [
    { _id: ObjectId },
    { _id: ObjectId }
  ], // players' ID
  log: [
    {
      output: {
        content: {
          0: {...}, // initdata above sent to player 0
          1: {...}  // initdata above sent to player 1
        },
        display: {...} // map information to display
      }
      time: Number // millisecond
    }, // Judge's output
    {
      0: {
        time: Number, // millisecond
        response: {
          direction: Number // player 0's action
        }
      },
      1: {
        time: Number, // millisecond
        response: {
          direction: Number
        }
      }
    }, // bots' actions for the first turn
    {
      output: {...},
      time: Number
    }, // Judge's output
    {
      0: {...},
      1: {...}
    }, // bots' actions for the second turn
    ...
  ],
  scores: [
    Number, // player 0's score
    Number  // player 1's score
  ]
}
```

B. Data Preprocess and Analysis

The gameplay state is presented as a zero-one triple-dimensional array with a shape of (10, 11, 4). The map is 10 grids high and 11 grids wide, and the number of channels is 4. The four channels represent: (1) our snake's head, (2) our snake's body (including the head), (3) the opponent snake's body (including the head), and (4) the stones (see Fig. 2). The mask is a 1×4 vector where the element at the index i is 1 if the action i is valid, 0 otherwise. We multiplied the output before softmax with a mask to make sure that the action does not violate the rules.

The count of valid actions varies from 0 to 3. The samples with 0 or 1 valid action should be deleted from the datasets, as these states are trivial. Finally, we removed the duplicated states to avoid overlapping occurring in training set and validation set.

C. Evaluation by Imitation and Clustering

We designed networks with different structures as the imitator. Then we used k-fold cross-validation to evaluate the performance of the neural networks. More concretely, we shuffled the dataset randomly, then splitted the dataset into k groups. For each unique group, we took it as validation set and the rest as training set. Finally, we summarized the performance, the mean validation accuracy, to evaluate the imitability of the bot.

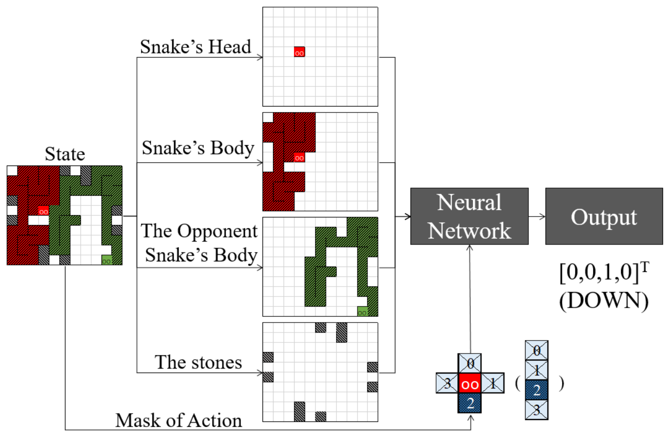

Fig. 2. Data formulation. We regulated the state of the situation as a zero-one triple-dimensional array with a shape of (10,11,4), and the four channels mean: (1) our snake's head, (2) our snake's body, (3) the opponent snake's body, and (4) the stones. The mask of action can be calculated according to the static situation.

The number of networks is N_n. The $j-th$ network evaluates the imitability of $i-th$ bot as a score IMI_i^j. For the $i-th$ dataset, we could get N_n evaluation scores $IMI_i^1, IMI_i^2, \cdots, IMI_i^{N_n}$ from the networks. Reduction like max, min or mean makes the object easier to cluster.

Similar to the idea in [16] and [17], we considered that the best performance of neural-network-based imitators is related to the self-similarity of the matches data. We used j^* to refer to the best network with the largest mean value.

$$j^* = \arg\max_{j=1}^{N_n} \frac{1}{N_b} \times \sum_{i=1}^{N_b} IMI_i^j \tag{1}$$

We use the median in $\{IMI_i^{j^*}\}_{i=1}^{N_b}$ to classify the bots into two categories. The bots with higher IMI is easier to imitate. The objective of classification is to give a semantic meaning to the bots with distinct behaviors, which is measured by the imitators. In this paper, we concentrated on the semantic-level analysis on the cluster results and explore the feasibility of using the method to cluster the bots with different playing styles.

D. Semantic-level Analysis

During the imitation process, the input only contains the state of situation without information of the algorithm or value function. To demonstrate whether the clustering results have a semantic meaning, we conducted static analysis on the selected bots' code. Then we described the information utilized in the value function, the search depth as well as search optimization to explain what and why the bot behaves a certain playing character in game playing.

IV. Experiments

A. Datasets

We collected 22 bots from the ELO rank list on the platform ($N_b = 22$). Simple descriptions of algorithms used in these bots are summarized in Table II. Public match logs[1] from May 2015 to January 2020 were used for our data.

The logs we downloaded from the platform have uniform JSON format, which includes the initial state and the bots' actions for each turn. We replayed the matches to generate state-action pairs as datasets. The gameplay state is presented as a zero-one triple-dimensional array with a shape of (10,11,4) (see Fig. 2). The deduplication of repeated data and the removal of states with zero or one valid action were done in the process.

TABLE II
Bots with different algorithms.

BID	Rank	Algo	SD	Prune	Timelimit
1	1	MCTS	inf	Yes	Yes
2	2	NE	ID	Yes	No
3	3	MCTS	inf	Yes	Yes
4	6	Alpha-Beta	ID	Yes	No
5	9	NE	ID	Yes	No
6	13	MCTS	ID	Yes	No
7	15	Alpha-Beta	ID	Yes	No
8	17	Alpha-Beta	ID	Yes	No
9	19	Alpha-Beta	11	Yes	No
10	20	MC	inf	Yes	Yes
11	31	DFS	8	Yes	No
12	32	DFS	8	Yes	No
13	34	Alpha-Beta	10	Yes	No
14	49	DFS	inf	Yes	No
15	54	MC	inf	No	Yes
16	56	MC	inf	No	Yes
17	77	ES	N/A	No	No
18	87	ES	1	No	No
19	89	DFS	inf	Yes	No
20	91	MC	1	No	No
21	149	ES	1	No	No
22	156	ES	1	No	No

[a]BID is the serial number of bots. Rank is the bot's rank in the ELO raking list. Algo is the algorithm the bot uses. SD means the search depth, and the ID means iterative deepening. Prune indicates whether the bot uses pruning. Timelimit indicates whether the bot balances the search time in different actions. The algorithms in the column of Algo are (1) MCTS, Monte Carlo Tree Search, (2) NE, Nash Equilibrium algorithms, (3) Alpha-Beta, AlphaBeta pruning with Minimax algorithm, (4) DFS, depth first search, (5) MC, Monte Carlo, (6) ES, Expert System.

Every bot's matches were dealt as a dataset respectively. For each dataset, we randomly sampled matches to remove the influence from significant difference in dataset size. We tested on 4 AIs to choose the best number of matches per dataset. 1000 is the best among other values (see Fig. 3). Particularly, we calculated the baseline accuracy as $((N_2/2)+(N_3/3))/(N_2+N_3)$, where the N_2 and N_3 are the counts of states with two and three valid actions respectively. We have analyzed that there are at most 3 valid actions in Section one. The states with zero valid action means that the player will lose in the following turn, and those with one valid action could be predicted all the time using the game rules. So we don't take the states with zero or one valid action into consideration. The results range from 40.0% to 47.3%.

[1]https://extra.botzone.org.cn/matchpacks/

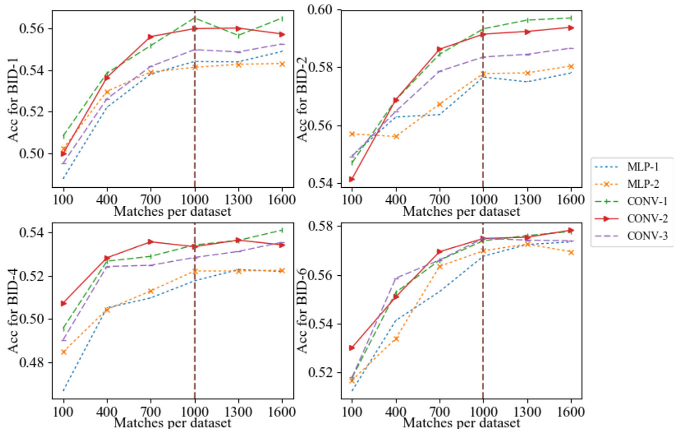

Fig. 3. Validation accuracies of AIs with different numbers of matches as datasets.

B. Networks

We designed 5 networks by two main structures of neural networks ($N_n = 5$), MLP and CNN, to imitate AI's decisions. NN1 and NN2 are MLP, and the others are CNN. We used MLP-1, MLP-2, CONV-1 to CONV-3 to refer those networks in the following. MLP-1 is a two-layer fully connected with 128 hidden units, and MLP-2 is three-layer with 80 and 256 hidden units in the adjacent layers. The sizes of convolutional kernel used in CONV-1 to CONV-3 are 2×2 and 3×3, and the max stride is 2. The number of parameters of the networks are shown in Table III.

TABLE III
THE NUMBER OF PARAMETERS OF THE NETWORKS

Network	Parameters
MLP-1	56964
MLP-2	57044
CONV-1	58052
CONV-2	61724
CONV-3	59452

The network takes a 4-channel input (see Fig. 2), and the output is a 4-element vector. Optimization is by Adam [20] using mini-batches of 32 samples in 300 epochs. Batch-normalization is used in both MLP and CNN networks. All the models are regularised using weight decay, starting with a learning rate of 0.02 and are trained on a GPU cluster for 80 to 100 hours in total. Before feeding the output to the softmax layer, we multiplied the output value with the mask of action to remove the invalid actions.

C. Cluster

We used 5-fold (k) cross-validation to evaluate the performance of the neural networks. For each network, we got 22 (N_b) evaluation scores of the datasets, and then we calculate the mean score. The best network with the highest mean score was chosen. Clustering was based on the best network's evaluation scores to find distinct playing styles in bots. The median dichotomizes the bots into two classes evenly, and a semantic meaning will be attached to the classes. To verify the reliability, we analyzed the codes of the selected bots and classify them by the algorithms and some implementation details, like the search depth.

D. Static Analysis of AI programs

We gave simple statistics of static analysis on the bots' codes in Table II. Three kinds of differences in bots raised our attention. First, the search depth determines whether the bot is farseeing or short-viewed. Second, some bots prune the game tree to accelerate the searching speed, and justify the time cost in expansion dynamically which helps search deeper to see further. Those short-viewed bots don't adopt the pruning method. Third, the bots that take an exhaustive search don't limit time in searching part of branches, which makes them farsighted but also leads to the less exploration on the other branches.

Because of the Monte Carlo method's principle, the MC bots and the MCTS bots use random action selection. The common steps in MC and MCTS algorithm are selection, expansion, and rollout/simulation, while the latter has one more step called backpropagation and keeps records at each node on the game tree: (1) the number of wins, (2) the number of simulations, (3) the total number of simulations run by the parent node [21]. The MC bots use uniform random action selection at each node. On the contrast, the MCTS bots select action by a probability that is proportional to the value of corresponding child node, which is calculated by UCB using the records [22]. The difference leads to the diversity in the game trees, like depth, width, scale, and balance, etc. The MCTS bots do well in balance the depth and width, which makes the bots far-seeing. Although the MC bots also concentrate on the width, they could't know a complete subtree which means that it makes a bias estimation of the actions. The MC bots perform worse than the MCTS bots.

The NE (Nash Equilibrium) bots evaluate each action and select the most promising one through finding a Nash Equilibrium in the payoff matrix [23]. The payoff matrix is a table in which the bot's valid actions are listed in rows and the opponent's in columns (or vice versa). The cells in the matrix record a pair of rewards to the rivals under the choice of the actions pair, and the rewards are evaluated using the dynamic information of the blank connected girds after several steps. The Nash Equilibrium is such a steady state that no one can change his action to be better when the other remains. Instead of selecting action randomly as MC bots and MCTS bots, the NE bots expand the node which is reached by the best action in the Nash Equilibrium solution. If the payoff matrix can precisely describe the situation, the NE bots will make the perfect choice. It's worth noting that the NE bots adopt pure strategy but not the mixed strategies, and they don't make choices randomly. However, the bots can't fill the payoff matrix in the beginning but update the matrix by the backpropagation of the result calculated in evaluation function from the leaf node to the parent node. It makes the NE bots close to the MCTS bots.

The search depth of Alpha-Beta bots is related to the search order. A good bot sorts the nodes to be expanded and can search deeper. The DFS bots have fixed search order and behave the narrow-minded characters, for they don't explore more to find a better solution. The DFS bots' overall performance is worse than Alpha-Beta bots'.

The bots with Expert Systems behave both short-sighted and narrow-minded.

Generally speaking, the NE bots and the MCTS bots are far-seeing and have a perspective of the general situation. The bots with Expert Systems behave both short-sighted and narrow-minded. The MC bots are far-sighted but have a bias estimation of the future. The Alpha-Beta bots are far-seeing and comprehensive if they with the nodes well sorted. The DFS bots are narrow-minded.

E. Results

The average accuracy curves of the five networks on the 22 datasets are shown in Fig. 4 and Fig. 5.

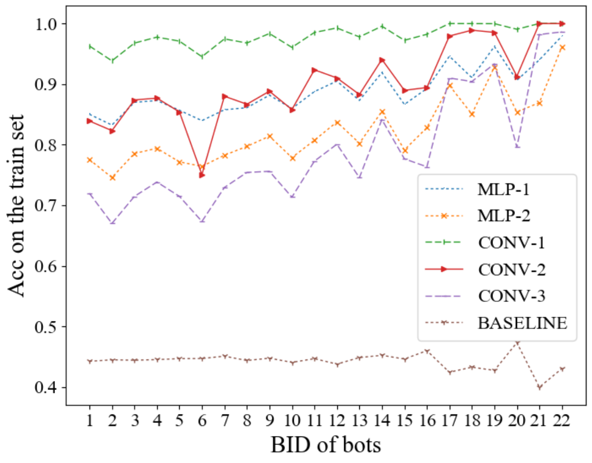

Fig. 4. The accuracy curves of five networks on train set after 300 epochs. The bottom curve is the baseline as random choosing.

Comparing the curves in the right subfigure, we found that CONV-2, the four-layer convolutional neural network (CNN) performs better than the other networks.

Using the validation accuracies of CONV-2, and get the median as 0.60, on which we cluster the bots into two classes. The first class contains the bots with id 1, 2, 3, 4, 5, 6, 7, 10, 13, 15, 16, and the second 8, 9, 11, 12, 14, 17, 18, 19, 20, 21, 22. We can see that the MCTS bots and the NE bots fall into the first class, while the DFS bots and the ES bots fall into the second class. The Alpha-Beta bots and MC bots fall into both classes.

The results reveal that the search preference, as well as information the bots utilizes, is relevant to the imitability. As mentioned in the static analysis of the codes, the unbalanced game tree will lead to bot's preference behavior in the actual match. The MCTS, MC, and NE bots utilize the information of

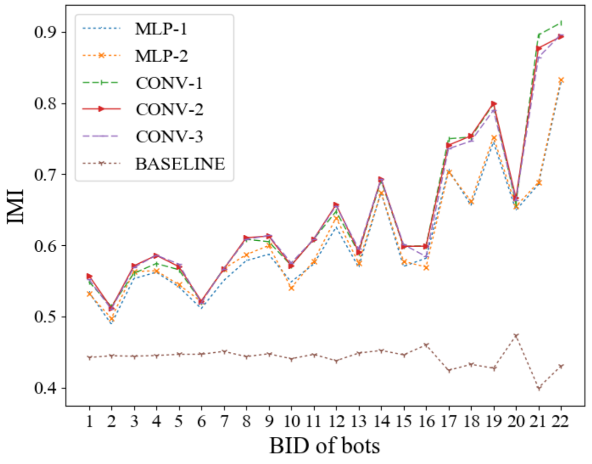

Fig. 5. The accuracy curve of five networks on validation set. The IMI is defined in Section 3 to denote the performance of the networks which reflects the degree of imitation. The bottom curve is the baseline as random choosing. The bot's imitability is measured by the accuracy on the validation set.

the grids state in the following, which reinforces their ability of grasping the general situation. The MCTS and the MC bots also tend to build a balanced game tree, while the Alpha-Beta bots and the DFS bots are affected by the search order. The difference makes the latter two kinds of bots spend too much time in searching the first available action and they fail to explore the other directions, which is narrow-minded as humans. The ES bots copy the thinking of the programmers. However, the human experience doesn't work on this game, for the state complexity is too large to build a decision tree with good strategies.

The bots with lower IMI in the first class behave the circumspect and farseeing characters. They are harder to imitate but perform better when competing with others than the bots with higher IMI. Algorithms that the first-class bots use are tend to build a balanced game tree and allocate more computing resources to the most promising nodes and those unvisited. The evaluation functions utilize dynamic information to endow the bots a perspective of the general situation. While the bots of the second class are nearsighted and narrow-minded, for they search partial branches of the game tree according to their preferences and only use the static evaluation of the current state.

We get the results through modelling the making-decision as a one-step process, which may lead to the bad imitation results of these far-seeing bots.

V. Conclusion and Future Work

We propose a novel method to cluster the AI programs by evaluating their imitability using NN-based imitators. We discover that the search preferences and information utilize in evaluation function are relevant to the AI's imitability. It means that imitability can be used to identify AIs' characters. An unbalanced game tree may account for the AI's behavior characters with preferences in the actual match. The AI who is difficult to imitate tends to build a balanced game tree and utilizes the information as far as possible, and it behaves farseeing characters. The easy-to-mimic AI is short seeing for they have preferences on certain regions/directions and evaluates the situation within few steps. The cluster method helps us conjecture how the AI builds its game tree and the information it uses in the evaluation function without prior knowledge of the AI, and we will give a semantic meaning to the cluster results.

Apart from the game *Snake*, we also did experiments on *Gomoku* and *Reversi*. The results were similar except the ES bots. We suspected that the humans did well in solving *Gomoku* and *Reversi* as many josekis were found. However, few researches were done on strategies of *Snake*, which forced the coders to search better action instead of using human experience.

There are still some aspects in need of further investigation in this paper. As claimed in [17] that the best compression rate is related to self-similarity property of sequence, we are going to experiment more network structures (like RNN, ResNet, etc.) to find the optimal prediction accuracy. It is necessary to do the comparative experiments on the bots with all combinations of algorithms and evaluation functions. Experiments on more games should be considered. We have conducted the imitation experiment on a bot that uses stochastic strategy but found that it was hard to imitate. It's easy to understand that we can't predict the next move of a clueless human, and it's impossible to imitate a completely making-random-decision bot. In addition to the clustering results with a semantic meaning of being farseeing and narrow-minded, there are many human-like playing styles to find. Further research will focus on revealing more fine-grained classification of AI programs' characters to make AI easy, intuitive, and vivid to understand.

Acknowledgment

This work is supported by Science and Technology Innovation 2030 "The New Generation of Artificial Intelligence" Major Project No.2018AAA0100901, China.

We would like to thank Haoyu Zhou and Haifeng Zhang for their contributions to the Botzone platform. We also would like to acknowledge the users who provide their AI programs.

References

[1] J. E. Laird and M. van Lent, "Human-level ais killer application: Interactive computer games," *AI Magazine*, vol. 22, no. 2, p. 15, Jun. 2001.

[2] J. E. Laird, "Research in human-level ai using computer games," *Commun. ACM*, vol. 45, no. 1, p. 3235, Jan. 2002.

[3] V. M. Petrovi, "Artificial intelligence and virtual worlds toward human-level ai agents," *IEEE Access*, vol. 6, pp. 39 976–39 988, 2018.

[4] R. Thawonmas and T. Harada, "Ai for game spectators: Rise of ppg," in *Proc. AAAI 2017 Workshop on Whats next for AI in games*, 2017, pp. 1032–1033.

[5] R. Ishii, S. Ito, M. Ishihara, T. Harada, and R. Thawonmas, "Monte-carlo tree search implementation of fighting game ais having personas," in *Proc. of IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.

[6] R. Ishii, S. Ito, R. Thawonmas, and T. Harada, "An analysis of fighting game ais having a persona," in *Proc. IEEE 7th Global Conference on Consumer Electronics (GCCE)*. IEEE, 2018, pp. 590–591.

[7] H. J. Van Den Herik, J. W. Uiterwijk, and J. Van Rijswijck, "Games solved: Now and in the future," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 277–311, 2002.

[8] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," Aug. 2019, pp. 6325–6331.

[9] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 2, no. 4.

[10] J. Ortega, N. Shaker, J. Togelius, and G. N. Yannakakis, "Imitating human playing styles in super mario bros," *Entertainment Computing*, vol. 4, no. 2, pp. 93–104, 2013.

[11] C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis, "Personas versus clones for player decision modeling," in *Proc. of International Conference on Entertainment Computing*. Springer, 2014, pp. 159–166.

[12] S. Devlin, A. Anspoka, N. Sephton, P. I. Cowling, and J. Rollason, "Combining gameplay data with monte carlo tree search to emulate human play," in *Proc. of Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[13] R. Thawonmas, M. Kurashige, K. Iizuka, and M. Kantardzic, "Clustering of online game users based on their trails using self-organizing map," in *Proc. of International Conference on Entertainment Computing*. Springer, 2006, pp. 366–369.

[14] W.-H. Tsai and D.-F. Bao, "Clustering music recordings based on genres," in *Proc. of International Conference on Information Science and Applications*. IEEE, 2010, pp. 1–5.

[15] R. Cilibrasi, P. Vitanyi, and R. De Wolf, "Algorithmic clustering of music," in *Proc. of the Fourth International Conference onWeb Delivering of Music, 2004. EDELMUSIC 2004.* IEEE, 2004, pp. 110–117.

[16] R. Kopiez, A. Lehmann, I. Wolther, and C. Wolf, "Musical style and authorship categorization by informative compressors," 2003.

[17] R. Cilibrasi, P. Vitányi, and R. d. Wolf, "Algorithmic clustering of music based on string compression," *Computer Music Journal*, vol. 28, no. 4, pp. 49–67, 2004.

[18] S. Feng, D. Wang, G. Yu, W. Gao, and K.-F. Wong, "Extracting common emotions from blogs based on fine-grained sentiment clustering," *Knowledge and information systems*, vol. 27, no. 2, pp. 281–302, 2011.

[19] F. Eyben, S. Buchholz, N. Braunschweiler, J. Latorre, V. Wan, M. J. Gales, and K. Knill, "Unsupervised clustering of emotion and voice styles for expressive tts," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 4009–4012.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *Proc. of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008, p. 216217.

[22] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proc. of European conference on machine learning*. Springer, 2006, pp. 282–293.

[23] M. J. Osborne and A. Rubinstein, *A course in game theory*. MIT press Cambridge, 1994.