Container-based Virtual Environment for Battle Execution of Round-robin in Applied Java Programming Exercise with Game Strategy and Contest Style

Naoki Hanakawa and Hiroyuki Tominaga
Kagawa University
2217-20, Hayashi-machi, Takamatsu City, Kagawa Pref., Japan
s16g464@stu.kagawa-u.ac.jp, tominaga@eng.kagawa-u.ac.jp

ABSTRACT

We have proposed an applied Java programming exercise with board-game strategy for learning problem solving. During the implementation of Gogo game methods, students learn concepts such as algorithms and revision using trial and error by execution results. We have developed the support system “WinG”. WinG-LA is a local review tool. It offers a game execution library as Java API’s, and contains four modules for examination of strategies. WinG-CS is a contest management server. It executes many games among the students' uploaded programs and maintains the preliminary and final periods for the battle league. Students’ scores are decided by the result of round-robin matching. We performed an educational practice in 2011. In this paper, we describe the improvement of WinG-CS and exercise practice, and introduce a virtual environment into the server for efficiency. We reconfigure the database. We also propose a program execution platform using container based virtual environment. This realizes inexpensive, easy, and suitable load distribution for the educational site.

KEYWORDS

Java programming exercise, board game strategy, continuous integration, round-robin contest style, rating index by standing, virtual environment

1 INTRODUCTION

1.1 The ultimate goal of programming education

In information engineering colleges, programming exercises are regarded as a required subject. After an introductory exercise in a junior class, an advanced exercise in a senior class considers object-oriented programming with C++/Java languages.

We consider that two aspects of the process and product are important toward the ultimate goal of programming education. One is the algorithmic approach, which refers to programming skills regarded as an effective tool for problem solving. The other is the desirable attitude of an engineer. This means continuous integration of programming according to software development methods. However, if the subjects of the field are far from the student's interest, it is difficult to illustrate concrete images and pursue the goal of programming. Moreover, without interest, a student does not feel any attachment to a software product that he creates and must maintain.

A teacher often selects applied problems related to specific IT fields. He also treats simple projects on information systems based on actual works. However, a student often does not understand the requisite mathematical concepts, or does not understand a concrete image because of a lack of social experience. Therefore, he tends to stumble on a clue of the problem condition, and gives up solving by programming.

1.2 Programming exercise of game strategy with a contest style

In the field of knowledge information processing, game strategy programming is an attractive exercise subject. It includes several learning items, such as formulation of move operation, condition of application, pattern matching of the board situation, intellectual searching by forecast, and simulation trial.

It includes the competitive learning approach, adopting a contest style between students’ strategies. A student considers this work a voluntary attempt rather than a teacher's imperative task. The approach is expected to improve motivation and to bring about educational effects. Moreover, the best
strategy, which is considered the most correct answer, is not clear. Thus, learners are required to continuously integrate new information through competitions with opponents.

2 OUTLINE OF OUR EXERCISE

2.1 Applied Java exercise of board game strategy

We have proposed an applied programming exercise in board game strategy using a contest style. We adopt the board game "Gogo," which is a variant of "Ninuki-Renju," like "Pente." We have developed the support system “WinG” [1][2]. We have carried out several educational practices since 2005 in an applied programming lesson. This exercise became a required subject, and was positioned as a more serious problem in 2011.

2.2 Rules and features of board game Gogo

Japanese traditional Gobang (Gomoku-Narabe) is a very famous board game, and is a simplified version of "Renju." Each player alternately puts a stone with his color (white or black) in a square mesh board. The player aims to create a connected run with his five stones in a straight or diagonal line. "Ninuki-Renju" is not so popular, but is a very interesting game based on Gomoku. It has additional rules about removing and capturing the sandwiched opponent's stones. "Pente" is known as a variant of the game in Germany and Poland [3]. We adopt "Gogo," which is also a simple version of Ninuki-Renju. The board has 169 cells in a 13×13 square. We adopt an arrangement of the rules for a reasonable subject of the programming exercise (Fig. 1). There are two winning conditions: creating a "steady" five-run or capturing five pairs (ten stones). As an "unsteady" five-run has removable pairs, the run can be broken in the next turn, and the game does not end. Moreover, if the fifth pair is obtained in that situation, victory can be achieved out of a disadvantage. Runs greater than five are not applied in the winning condition. Making two three-runs (San-San) by placing a stone is a mismove and a losing condition, whereas making San-San by removing a pair is allowed. Gogo is profound in that removing stones causes a significant change in the board situation, and surprising turnover of a game aspect. When forming a strategy, two aspects must be considered. One is the tendency to aim for connecting or capturing, and the other is a tendency to choose attack or defense. The policy of strategy has many variations with various evaluation functions. Though the rules are simple, a beginner shows his individuality with the preference and attitude. As documents about rotes of Ninuki-Renju are not enough, you must search good strategies by yourself.

2.3 Previous system and our research

We developed the first version of WinG in 2005. It consisted of a local execution environment and contest-management server WinG-CS [1]. The environment offers game execution between strategies or human players, and battle record functions for review. By doing this, we support the students' creation of strategies. With the initial edition of WinG-CS, we devised a mechanism by which students submit strategies and execute them on the server.

The environment changes in a local development tool WinG-LA as the next version [2]. WinG-LA consists of 4 modules: "game execution," "battle record replay," "board situation generation," and "hand trial test." We introduced SWG and the first version of WWG (weighted winning grade), which is a simple rating index [4].

2.4 Making process of the strategy program

We offer a board game execution library for Gogo as a Java API. The hand method calc_hand() is overridden in the subclass inheriting the Computer Player class by student. The method receives the current state as an argument. It returns the next hand. The state, as an instance of the State class, consists of the board situation for stone placement and two pockets for captured stones. An instance of
the Master class manages the game progress according to the rules. We present a strategy prototype to students, which includes the necessary processing as comments. Most strategy codes have 300–500 lines.

The overview of the strategy design in Gogo is shown in Fig. 2. In the first step, the outline of a strategy idea is considered, and a tactics policy is determined. Each tactic is mostly described by if-then rules in terms of knowledge and information processing. In the second step, the left-hand side of a rule is pattern recognition of the stone placement on the board. Various matching algorithms of stone placement are realized, such as a four-run, double runs, and multiple capturing. More detailed patterns may be refined, and specific patterns may be determined for the winning process. The right-hand side is the assignment of an evaluation value for each cell by a heuristic function. Table 1 shows examples of production rules. The evaluation value must be revised by trial and error through game execution. Students consider the global board situation and an adjustment of the priority of the rules. In the last step, a cell with the maximum value is selected as the hand. These steps are repeated to continually improve the strategy.

Not only does pattern matching improve, but forecast reading is also adopted in the game-tree search. If the strategy is used, strategy evaluation must be used, such as α-β pruning. Students may also introduce strategies based on probability, like the Monte-Carlo Tree Search, as a more advanced challenge. However, in such complex strategies, they need to consider the time limit per hand and make the appropriate pruning steps.

2.5 Progress support of strategy programming by WinG-LA

The local review package WinG-LA offers several functions for the examination of strategy and debugging. It consists of four modules, as shown in Fig. 3. They are implemented as a Java standalone application. The package also contains a reference manual for the game execution library and contest rules.

The strategy execution module performs a battle between computer players with implemented strategies as the execution environment (Fig. 4a). The record replay module exhibits a game by a battle record for strategy analysis (Fig. 4b). The state edit module makes initial game states for test cases of hand debugging (Fig. 4c). The hand trial module shows results of some board state with one hand operation by a strategy program as a simple test (Fig. 4d). To aid in strategy consideration, WinG-LA also prepares many samples of game strategies, initial states, and battle records.

2.6 Preliminary and final league in game contest

In our exercise, we have set a tournament period of several weeks. We also carry out a preliminary league and a final league in a battle style. Learners can submit their strategy many times during about five weeks in the preliminary league. The submitted strategy competes with other strategies on the contest management server. The battle result is given by weight winning grade considering rating of each strategies. In the preliminary league, the server receives ~1000 submissions from 40 learners. This means that the server performs hundreds of thousands of battles in a round-robin battle, which requires a long execution time. Therefore, we introduce a thinning round-robin battle in consideration of a winning degree.

The contest management server WinG-CS publishes all battle results and the rankings of the submitted strategies. Learners can check steps and the cause of victory in their battle results, and replay battle records while excluding the opponent strategy codes. We introduce index strategies in three strength levels as the strength index. This information is effective feedback for revising strategies. In this way, we motivate the learners to revise their own program continuously by providing the opportunity to constantly evaluate their own strategies.

After finished preliminary league, students comprehensively judge and select their own best strategy, which will participate in the final league. In the final league, the server performs a round-robin battle with the strategies provided. An evaluation of each student is
decided by the score and a summary report, in which he/she analyzes the process and result.

![Figure 1. The rules of Gogo and board states](image1)

![Figure 2. The overview of strategy design in Gogo](image2)

![Figure 3. The module configuration in WinG-LA](image3)

![Figure 4. GUI of the 4 modules in WinG-LA](image4)

Table 1. Examples of production rule

<table>
<thead>
<tr>
<th>Pattern</th>
<th>A/D</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroy: 5-run</td>
<td>Run</td>
<td>Defense</td>
</tr>
<tr>
<td>Realize: 5-pairs</td>
<td>Pair</td>
<td>Attack</td>
</tr>
<tr>
<td>Realize: 5-run</td>
<td>Run</td>
<td>Attack</td>
</tr>
<tr>
<td>Block: 5-pairs</td>
<td>Pair</td>
<td>Defense</td>
</tr>
<tr>
<td>Block: 5-run</td>
<td>Run</td>
<td>Defense</td>
</tr>
<tr>
<td>Attack: 4-run</td>
<td>Run</td>
<td>Attack</td>
</tr>
<tr>
<td>Block: 3-run</td>
<td>Run</td>
<td>Defense</td>
</tr>
<tr>
<td>Attack: 1-pair</td>
<td>Pair</td>
<td>Attack</td>
</tr>
<tr>
<td>Mismove</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Others</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

To play games by the strategy with opponent player
To decide the best hand
To compare alternatives
To assign evaluation values
To recognize stone pattern
To consider outline of strategy
Connected Run Capturing Pairs
Four-run Double Three-runs
Attack Defense

Guang-Hong Huang, Hsing-Kuei Lin, Yu-Chih Huang, and Yu-Hsiang Chou

The International Journal of E-Learning and Educational Technologies in the Digital Media (IJEETDM) 3(2): 83-98
The Society of Digital Information and Wireless Communications (SDIWC), 2017 ISSN: 2410-0439 (Online)
3 Purpose and position of our exercise

3.1 Game strategy programming exercises in our curriculum

We carried out this exercise in an experiment for the third-year students in an information and electronics engineering college. Students had already studied data structures and algorithms in C and the basic syntax of Java in another class. We have also carried out an exercise with card game strategy in C [5]. Its subject matter is a poker game. The students aim for high points, and they are ranked by points. They learn the design of data structures, such as decks and hands, implementation of pattern matching based on a production rule, and the judgement of poker hands. After completing the poker exercise, we started this exercise.

We described the differences between the card game and board game exercises. These exercises complement each other. In the card game, we used incomplete information game. In addition, the card game exercise used scoring, rather than competition. Thus, the students’ efforts directly lead to results. In contrast, in this exercise, we use a game with perfect information, and this exercise is in a battle style. Therefore, it is difficult for students to improve results only by their own efforts.

3.2 Learning items and educational purpose

Our educational goals in this exercise consist of two aspects: analysis and design, and implementation and verification. On analysis and design, we make the students examine the application conditions of the hand and priority of processing based on search algorithms in knowledge engineering. Such tasks include formulating the hand application condition, extracting the pattern of a characteristic state participating in a win or lose condition, such as San-San, and implementing heuristic knowledge obtained from actual experience.

On implementation and verification, we require the students to practice the spiral development method based on object-oriented programming. In other words, we stimulate the students to improve their evaluations of states or hands from the battle result. This includes steps such as localizing points to fix the evaluation parameters using modularization, verifying the validity of method of hand using the sample state as a test case, and improving the state analysis by replaying battle records.

3.3 Related works

Here, we describe related competitions. IPSJ have held SamurAI Coding, which is a game AI programming contest, since 2012 [6]. IEEE CIG holds many game AI competitions annually [7]. MIT has also held Battlencode, which is a battle game AI contest [8]. In general corporations, CodinGame has held the CodinGame Contest, which is an open competition [9]. The purpose is the enhancement of programming skills through the creation of game AI. Thus, many strategy programming contests are held worldwide.

There are also related researches on education applications. Canada aims to improve student knowledge about AI through an international online contest [10]. They verified the educational efforts of participating in the Google AI Challenge as part of the class. It improved motivation and results by using contests in the class. Yoon teaches basic concepts about games and AI using Angry Birds [11]. Other researchers provide practice classes using computer games with the contest style [12][13]. In our research, the purpose is not only creating strong strategy, but also learning appropriate techniques and good practices in software development. This is the most distinct aspect of our research.

In recent years, the development of game AI by machine learning is remarkable. In Go, DeepMind’s AlphaGo broke the world’s top talent for the first time using deep reinforcement learning [14]. AlphaGo Zero, using reinforcement learning through only self-games, is showing a higher Elo rating than AlphaGo [15]. Open distribution of libraries for machine learning, including TensorFlow and Chainer, is also increasing [16][17]. Now, this exercise targets examination and realization of classical strategy. In the future, we would like to respond to such advanced approaches.
4 SUMMARY OF CONTEST MANAGEMENT SERVER WinG-CS

4.1 Support system WinG and execution environment

To realize the exercise, we have developed the support environment WinG (Fig. 5). Learners download WinG-LA, which is a support tool for developing strategies, from a shared folder on our server. They implement their strategies on their own PC. We offer a game execution library as Java APIs, and students use the development environment of their choice. WinG-LA provides some modules, such as battle execution and efficient debugging. It also provides some samples strategies and board situations for examination of their strategies.

Server-side WinG-CS, which supports contest management, manages some processes on the server side in the preliminary and final leagues. A league corresponds to each exercise in each year. The system makes the submitted strategies meet in a battle game, and publishes the battle results and records (Fig. 6). The system saves the battle results and records to a database on the server. Learners can browse these data. They can also download battle records, and replay the battles on WinG-LA.

4.2 Functions and configuration of WinG-CS

In this paper, we describe features and enhancement of WinG-CS in 2016 CS. From 2011 to 2015, we had carried out educational practices with previous version of WinG-CS. The system was implemented in Ruby 1.9, and the database is saved by XML. New WinG-CS, which was enhanced in 2016, is implemented in Ruby 2.3 and Ruby on Rails 4.2. Rails is based on a Model-View-Controller architecture, which allows us to reduce the development cost in coordination with the database and GUI. Our contest server mounts an Intel Core i7 CPU and DDR3 32GB RAM. The server OS is Linux series CentOS 6.8, but we updated the Linux kernel to 3.10 to use the virtual environment Docker.

4.3 Internal processing of WinG-CS

WinG-CS accepts the source codes of learners' strategies during the preliminary league. WinG-CS performs acceptance and battle processing as internal processing at all such times. Acceptance processing and battle processing proceed seamlessly after submission.

Acceptance processing performs a static check of the synchronous processing and a dynamic check of the asynchronous processing. WinG-CS registers the submission file to the submission DB, and also registers the acceptance result. In the static check, the system checks the file type and file size, and compiles the submission file (Fig. 7). When the learner submits a binary file or a potentially malicious file, the system excludes this file. Dangerous files include some functions, such as OS command execution or input from the keyboard. The students are notified of the result of the static check is notified regardless of success or failure. In the dynamic check, the system performs a checking battle to test compiled strategy binary by a battle with a sample strategy. The system excludes some strategies, such as stopping by error or causing an endless loop. If checking battle finishes successfully, it registers the strategy DB as a strategy which can participate in the preliminary league.

Subsequently, in battle processing, the submission strategy performs battles with all submitted strategies and is calculated the weighted winning grade (WWG) (Fig. 8). First, the system obtains strategies in the league, and creates a tournament chart with them. Next, the system performs battles by the tournament chart. During this time, the battle results and game records are saved as temporary files. After finishing battles, it registers the opponent ID and battle result to the battle DB. The game records are XML files, so the system saves a file path to the battle DB. Then, it calculates a temporary WWG using the battle results.

4.4 Introduction of time of suspension of submission

Battle processing takes a considerable amount of time as the number of submissions increases.
In addition, the WWG is affected by the WWG of other strategies, which it also affects. Therefore, processing becomes complicated with successive submission of strategies. WWG also changes frequently. All of this sometimes results in system faults or significant delays in processing on the legacy system. For the reason, we deal with a thinning round-robin series to reduce the amount of processing.

In 2016, we decided to stop strategy submission from 24:00 to 6:00. During this period, we finish the battle processing for the day. The system calculates the battle results and WWG at that time. These results are reflected in the ranking table, and published to the Web in the morning. In the new system, battle processing and database management were improved, so system faults and significant delays no longer occurred. However, there is still room for improvement in the immediacy of result reflections.

4.5 Reconstitution of database

In the legacy system, we use XML as the database. However, when the system registers the battle results of about 1000 strategies, it takes a long time to load from database. We therefore reconstitute the database using RDBMS (Relational database management system) to accelerate. We use PostgreSQL 9.6 as the DBMS (Database management system). Table 2 shows a list of tables that comprise the database. The submission table "Submission" registers all the submitted files. It is a submission management log for the learners. The strategy table "Strategy" registers only strategies which pass acceptance processing and participate in the preliminary league. ID is given to each strategy with a serial number for each learner. It is used to make the tournament chart.

The battle result table “Battle” saves all the actual battles. It registers the self and opponent IDs, battle result, and path of the battle record file. Battle record files are still in XML format for use in the record replay module in WinG-LA. In the future, we will consider migrating to saving these files in JSON format.

4.6 Use of Docker in battle environment

We describe using Docker on WinG-CS. On WinG-CS, we register the Docker image as a template of the execution container in advance [18][19]. This image includes openJDK, and contains an execution library and commands. In executions of individual battles, WinG-CS creates an execution container based on the image. The execution container includes the tournament chart (Fig. 9).

Furthermore, the execution container mounts strategy-saving directories and game record directories on the host machine. As a result, we can directly read and write data on the host machine from container. First, the container begins battle processing according to the execution commands. After the battles, the container transfers all battle records to the host machine, and any databases are updated. Finally, the container is discarded. Such an approach also facilitates load balancing using multiple machines.

4.7 Improvement of GUI in the student side

We describe the GUI of the improved system. The top page of the league shows tables of strategy ranking and individual submission history by tab. Learners can remain anonymous by using player nicknames. The general ranking table in Fig. 10a shows the ranking of all submitted strategies so far. Rows of index strategies and their own strategies are shown in different colors, and are emphasized from other learner’s strategies. The submission history tab in Fig. 10b shows only the student’s own strategies. Submitted files are shown in a time series. If the user clicks a strategy name, they move to the strategy detail page.

The strategy detail page shows detailed information about the individual strategy in various tabs. The strategy summary tab in Fig. 10c shows conventional information, such as the submission date, battle result information, and strategy code. Learners select their own best strategy for the final league. The records list tab shows all the results of the strategy. Learners can download battle records, and replay them on WinG-LA. They can also download all battle records at once.
Table 2. Database tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Student ID, Login information</td>
</tr>
<tr>
<td>League</td>
<td>Subject’s name, Contest period</td>
</tr>
<tr>
<td>Player</td>
<td>Player name, Role, Best strategy ID</td>
</tr>
<tr>
<td>Submission</td>
<td>Strategy name, Check status</td>
</tr>
<tr>
<td>Strategy</td>
<td>Submission ID, WWG, File path</td>
</tr>
<tr>
<td>Battle</td>
<td>Strategy ID, Battle result, File path</td>
</tr>
</tbody>
</table>

Contest management server WinG-CS

Execution container

WinG execution module

tournament chart

battle logs

Strategies

Battle logs

Host machine

mount

Strategies

Battle logs

Figure 5. Architectures of support system WinG

Figure 6. Functions of WinG-CS

Figure 7. Flow of acceptance processing

Figure 8. Flow of battle processing

Figure 9. Structure of execution environment in Docker container

(a) Total ranking tab

(b) Submission log tab

(c) Strategy summary tab

Figure 10. GUI of student side
5 IMPROVEMENT OF RATING INDEX

5.1 Competition method and winning point

In this exercise, a match consists of two game sets, with players taking turns to go first and second for fairness. A player is given winning points by winning or losing. Table 3 shows the winning point (WP). If the result of two sets is one win and one loss, the outcome is decided on points based on the sum of captured stones. By WP, we can simply subdivide results which were judged a draw, and classify them clearly. If battle numbers are the same in a round-robin tournament, the individual total sum of WP stands for the strength of the strategy. The simple winning grade SWG, which is calculated by dividing WP by 4 times the battle number, is the actual value in the interval [0,1]. SWG is a percentage of victory considering WP. If the result is composed only of complete win and complete loses, the SWG agrees with the normal victory percentage.

5.2 Necessity of rating index

In the preliminary league, the ranking may be influenced by a tendency of the main strategy group during a period. At the beginning of the period, very weak strategies by incomplete programs are included. A high winning degree by these does not show their exact strength. While a few aggressive students upload many similar strategies, the deviation of strategies causes inaccurate or unfair results with ill-suited battles. Thus, an affinity for the same type of strategy strongly appears. In some cases, it is possible to prepare a dumping strategy like kingmaker, and to raise the ranking of the strategy which wants to win. Therefore, it is necessary to reduce bias due to apparent strength and affinities. For this problem, a WP grade considering the strength of the opponent is necessary.

In games such as chess, the Elo rating is a famous player rating index. Elo rating assumes that the competence of each player is close to the normal distribution, and the change in ability is based on a logistic curve [20]. However, in this exercise, we would like to calculate the appropriate WP for each confirmed strategy. In reality, differences in programming skills are large, and extremely strong and weak strategies are mixed. In particular, the trends are remarkable at the beginning of the league. In addition, the rating is used as a ranking for the current record rather than for future prediction. For these reasons, we consider a rating index based on the standing at that time.

5.3 Improvement of weighted winning grade WWG

In this exercise, we introduce a weighted winning grade (WWG) as a rating index. WWG is a refinement index of SWG. It can make comparisons, even in situations with different number of battles in the middle of the preliminary league. WWG is the actual value on the interval [0, 1], and weights its own WP by the opponent’s WP. It is necessary to calculate recursively, because one’s own WWG changes with the change in the opponent WWG. When the number of battles is small in the early stage of the preliminary league, the definition of WWG prior to 2016 had a defect that caused a large fluctuation of WWG. For that reason, we adopt a new definition as of 2017 [21]. In addition, at a sufficient number of battles, the difference between the new and the old WWG is not noticeable except for in extreme battle results.

The fight number of strategy \( x \) is \( N(x) \) and winning points for strategy \( x \) of \( y \) is \( WP(x, y) \); let \( WWG_k(x) \) be the WWG for strategy \( x \) by the \( k \)th calculation. The initial value is \( WWG_0(x) = 1.0 \). \( WWG_{k+1}(x) \) is calculated by a recurrence formula. We repeat the calculation of the formula until the change in the value fails below an appropriate threshold. The value is the confirmed WWG.

\[
WWG_{k+1}(x) = 0.5 + \frac{\sum_{i \neq x} \left\{ WWG_k(i) \times WP(x, i) \right\}}{\sum_i \{ WWG_k(i) \}}
\]

In this definition, it includes a virtual battle with the same strategy. This definition accounts for draws by the +0.5 in the numerator. Here, the 1st calculation \( WWG_1(x) \) is the simple winning grade \( SWG(x) \), including self-draw.
The denominator is the sum of the WWGs, and it makes the WWG relative. If a strategy wins against all strategies, WWG does not result in 1.0 by self-draw. As the count of complete wins increases, it approaches 1.0 slowly. Furthermore, the WWG in complete loss to all strategies is not 0.0, but half of the reciprocal of the sum of WWGs of all strategies by self-draw. The as the number of battles increases, it slowly approaches 0.0.

The previous definition of WWG doesn’t include the self-battle in the numerator. In this way, if there is all-loss strategy, the weight of a win to the strategy is 0.0. Then, it spreads in the recurrence formula, and all WWG values converge to 0.0. In an early stage in 2016, the trend became noticeable, so we introduced the correction of the normalization. However, in contrast, if all the forces are in equilibrium and the victory or defeat becomes close to half, the WWG may be contrary to intuition. By the new definition of WWG, it can calculate a reasonable value in extreme situations of battle results. The convergence is also improved.

Table 4 shows winning or losing in 6 strategies of A to F, and the convergence condition of WWG. (a) shows the case when superiors always win completely to subordinates. A with all winning converges to 0.866, and F with all defeat converges to 0.183. Other strategies also converge to a reasonable value after about 6 iterations with oscillation. On the other hands, (b) is a slightly more complex winning and losing situation. A and B display 4 wins and 1 loss. However, A wins against B in addition to D, E, and F. B wins over C in addition. At that time, A wins against a stronger opponent than B, so A has a higher WWG than B. E and F each have 1 win and 4 losses. However, E wins only against D, and F wins only against E. E wins against a higher-rank strategy than F. Therefore, E’s WWG is higher than F’s. Both C in (a) and C in (b) have 3 wins and 2 losses, but the WWG of C in (b) is a little higher than C in (a) because of the win against the top strategy.

### Table 3. Winning point

| 2 wins | complete win | 4 |
| 1 win | decision win on point | 3 |
| 1 lose | (more captured) | 1 |
| 1 win | draw by the same point | 2 |
| 1 lose | (same captured) | 1 |
| 1 win | decision lose on point | 1 |
| 1 lose | (less captured) | 1 |
| 2 loses | complete lose | 0 |

### Table 4. Standing and calculation status of WWG

<table>
<thead>
<tr>
<th>(a)</th>
<th>Win</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 5 0 BCDEF</td>
<td>0.917</td>
<td>0.861</td>
<td>0.857</td>
<td>0.869</td>
<td>0.867</td>
<td>0.865</td>
<td>0.866</td>
<td></td>
</tr>
<tr>
<td>B 4 1 CDEF</td>
<td>0.750</td>
<td>0.611</td>
<td>0.615</td>
<td>0.643</td>
<td>0.636</td>
<td>0.633</td>
<td>0.635</td>
<td></td>
</tr>
<tr>
<td>C 3 2 DEF</td>
<td>0.583</td>
<td>0.417</td>
<td>0.451</td>
<td>0.478</td>
<td>0.464</td>
<td>0.462</td>
<td>0.465</td>
<td></td>
</tr>
<tr>
<td>D 2 3 EF</td>
<td>0.417</td>
<td>0.278</td>
<td>0.341</td>
<td>0.353</td>
<td>0.338</td>
<td>0.339</td>
<td>0.341</td>
<td></td>
</tr>
<tr>
<td>E 1 4 F</td>
<td>0.250</td>
<td>0.194</td>
<td>0.264</td>
<td>0.256</td>
<td>0.246</td>
<td>0.249</td>
<td>0.250</td>
<td></td>
</tr>
<tr>
<td>F 0 5</td>
<td>0.083</td>
<td>0.167</td>
<td>0.198</td>
<td>0.183</td>
<td>0.180</td>
<td>0.183</td>
<td>0.183</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b)</th>
<th>Win</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 4 1 BDEF</td>
<td>0.750</td>
<td>0.722</td>
<td>0.729</td>
<td>0.739</td>
<td>0.740</td>
<td>0.736</td>
<td>0.737</td>
<td></td>
</tr>
<tr>
<td>B 4 1 CDEF</td>
<td>0.750</td>
<td>0.667</td>
<td>0.701</td>
<td>0.706</td>
<td>0.698</td>
<td>0.699</td>
<td>0.701</td>
<td></td>
</tr>
<tr>
<td>C 3 2 AEF</td>
<td>0.583</td>
<td>0.583</td>
<td>0.598</td>
<td>0.574</td>
<td>0.584</td>
<td>0.587</td>
<td>0.583</td>
<td></td>
</tr>
<tr>
<td>D 2 3 CF</td>
<td>0.417</td>
<td>0.444</td>
<td>0.549</td>
<td>0.520</td>
<td>0.505</td>
<td>0.516</td>
<td>0.516</td>
<td></td>
</tr>
<tr>
<td>E 1 4 D</td>
<td>0.250</td>
<td>0.306</td>
<td>0.318</td>
<td>0.331</td>
<td>0.326</td>
<td>0.322</td>
<td>0.325</td>
<td></td>
</tr>
<tr>
<td>F 1 4 E</td>
<td>0.250</td>
<td>0.250</td>
<td>0.271</td>
<td>0.258</td>
<td>0.266</td>
<td>0.265</td>
<td>0.263</td>
<td></td>
</tr>
</tbody>
</table>

### 6 OUTLINE AND RESULT OF PRACTICES

#### 6.1 The outline of practices in our programming exercises

Table 5 shows a summary of the exercise practices from 2013 to 2016. The exercise period for this practice is ~5 weeks. In the first lesson, we introduce the rules of the game, and also explain strategy programming, local execution environment, and contest management server. The following week, the preliminary contest period begins. In class, we check the general ranking or submission state between other practices, and explain concepts further. The deadline is the end of the preliminary league. After that time, learners select their best strategy for the final league. In 2014, the server had failed. In 2015, the end time of other practices was postponed, so the start time of this practice was delayed. For this reason, we extended the practice period to avoid deadline conflicts with the other practice. However, the substantial development period is as usual. In 2016, the start of the contest was also delayed in relation to other practices. We
presented the development environment in the two weeks before the start of the contest, which is as short as the preliminary league. Therefore, we also provided a substitutional development period as usual. In an exercise in 2016, we had not adopted the thinning round-robin series, so the submitted strategy battles with all registered strategies were done at that time. It was not a problem until the middle of the contest period, by improvement of battle processing. However, in the end stage of the league, a delay occurred due to the increasing number of submissions.

6.2 The transition situation of submission

In 2016, the total number of submissions is 800. The average number of submissions per person is about 20. The maximum submission number is 67, which greatly exceeded 2015 numbers. Fig. 11 shows the transition of the number of submissions each year. The number in 2016 was similar to that in 2013. The plateau in the middle of contest period is less than in other years, and the number of submissions consistently increased. Fig. 12 shows a frequency distribution of the submissions. In 2013, there was a learner who submitted over 100 strategies. In 2016, the number of learners who submitted five or fewer decreased, and the distribution peak is on the right compared to the normal year. On the other hand, half of the learners submitted 20 or fewer. In the future, it is necessary to support these learners. Currently, we focus on the transition of results by a correlation of STG and WWG. It is necessary to find low-rank students based on submissions, students with less submissions, and low-rank students, and support or encourage them. On the other hand, we must also induce high-ranking students with fewer submissions to work continuously. If it is difficult to develop a new strategy, we attempt to induce learners to change their focus to the quality of their code.

6.3 Application of improved WWG to final leagues

We tentatively apply the new definition of WWG in Chapter 5 to the final leagues after 2013. Fig. 13 shows their correlation. The horizontal axis X is SWG, and the vertical axis Y is WWG. Spearman’s rank-order correlation ρ is also mentioned in these figures. Each year, the correlation ρ exceeds 0.99, and there is a strong correlation between WWG and SWG. Thus, it is considered to calculate ranking intuitively. Strategies with 0.4 to 0.7 SWG are differentiated by the strength of opponents. On the other hand, the number of strategies with lower differences is shrinking. This is because of the influence of self-battles.

6.4 Consideration

We consider changing regulations in the future exercise. First, considering the recent progress of game AI, we need to respond to strategies including random-number elements. In the case of strategies including random-number elements, the execution time becomes long. Therefore, we must consider the execution server load. In the evaluation method of the strategies, there is also room for consideration of whether a single match against the first or second player is good. It is also conceivable to combine point games, like the composed Go problem or Tsume-shogi. In addition, we also need to introduce some supports for students who don’t have good game records. Currently, for the purposes of discovering such students, we have proposed a transition graph (STG) and trace diagram TDQP, and are analyzing them [22][23]. STG is a combination plot of execution results and time courses (Fig. 14). TDQP visualizes the update status with the execution results and the absolute amount of code as two axes (Fig. 15). In the future, we will add some functions using these methods to enable the system to support them. In the current exercise, students are allowed to submit only one strategy file as one class at a time. However, it is also possible to submit a strategy that defines multiple classes. In this case, we need to allow them to submit an archived JAR file. In addition, we may also need to correspond to strategies using external files such as a collection of standard theories. In such case, it may be possible to communicate with outside and acquire standard theories. In this regard, we need to consider this as regulation. In the future, there is a high possibility of submitting strategies incorporating reinforcement learning in this exercise.
Furthermore, as an exercise for knowledge of information processing, it is possible to hold exercises on the theme of machine learning.

Table 5. Summary of practices in each year

<table>
<thead>
<tr>
<th>year</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
<th>2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period (week)</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Learners</td>
<td>37</td>
<td>35</td>
<td>44</td>
<td>40</td>
</tr>
<tr>
<td>WWG</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>◎</td>
</tr>
<tr>
<td>Thinning round-robin series</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>×</td>
</tr>
<tr>
<td>Total submissions</td>
<td>942</td>
<td>377</td>
<td>406</td>
<td>800</td>
</tr>
<tr>
<td>Average personal submissions</td>
<td>25.5</td>
<td>10.8</td>
<td>9.2</td>
<td>20.0</td>
</tr>
<tr>
<td>Maximum personal submissions</td>
<td>142</td>
<td>32</td>
<td>44</td>
<td>67</td>
</tr>
</tbody>
</table>

●: New introduction  ○: Modified method  ◎: Regular use  ×: Unused

Figure 11. Ogive curves of submission number in each year

Figure 12. Frequency distribution of submissions

Figure 13. Application of improved WWG to final leagues in each year
7 Proposal of an execution environment framework

7.1 Execution requirements and characteristics about strategy programs

In this research, we consider external and internal evaluations for game strategy programs. External evaluation is mainly for obtaining calculation accuracy and processing speed as external performance. Regarding game strategy, it refers the score of execution results or the win or loss of the match, which expresses strength. Internal evaluation is the evaluation of code quality by software metrics (e.g., redundancy, efficiency, structuralization, and readability).

In addition, it is also important to grasp the progress of continuous program submission. We classify strategy programs into four groups from the viewpoint of execution performance.

1. Both processing time and execution result are normal.
2. Processing time is long, execution result is good.
3. Execution result is remarkably bad.
4. Processing time is too long.

Of these, (3) becomes noise for the records of other strategies. Therefore, in actual processing, there are no problems, even when deprioritizing the battle processing. (4) becomes a large cost by occupation of computational resources. They are necessary to eliminate as an error of execution time or excessive processing.

In addition, we also enumerate the functional requirements of contest management servers.

- Protecting non-public data in a server
- Responding to many executions and errors
- Ease of replication and change in execution resources
- Comparative verification of program operation and performance
- Estimation of operation and response to exceptional functions

Currently, based on these requirements, we have tried to realize the execution environment for each system. However, in processing by a single server, some problems occurred, such as occupation of the resource in protecting non-public data, and responding to any executions. It is necessary to solve these problems.

7.2 Present condition about distributed processing environments

Recently, distributed processing environments for high-load processing are improving. In the Apache Hadoop project, they have been developing and openly distributing some distributed processing libraries such as MapReduce and Spark, distributed storage application HBase [24]. Amazon and Google provide managed services, such as Hadoop, on their own cloud environment [25][26]. However, the exercises in class have a greatly different load situation depending on time. Therefore, it can be said that efficient operation is difficult in an environment, assuming always
operation. Adding a processing node at high load may be difficult for teachers. For these reasons, it is necessary to have an environment that easily handles high loads.

### 7.3 Execution environment using container environment

Therefore, we propose the environment “Cachalot” using a virtual environment that can execute programs independently, safely, and effectively. This is a sandbox for execution that aggregates execution environments prepared individually by each contest server. This environment is isolated from contest servers, so it is possible to protect non-public data. In addition, it is also possible to stop unusual execution from the outside.

Each contest management server can acquire the execution result by only sending an execution setting file (execution recipe) and the source code and so on to Cachalot (Fig. 16). In Cachalot, it prepares a container type virtual environment internally, and execute the code inside the container. As a result, independence and safety can be secured. It is also possible to safely stop unusual execution by transmitting an exit signal from outside. In addition, faster processing is realized by load distribution using multiple inexpensive computers.

Cachalot consists of 4 modules (Fig. 17). The contest server management module manages the contest server, which performs submission processing. It also manages images of execution environments related to each system. The execution container management module manages the execution time and resource information, such as assigned cores and assigned memory, and also inspects whether the execution condition is specified by the contest server. If a program executes for longer than the time limit, Cachalot kills the container safely. The processing nodes management module manages multiple nodes which process the execution container. The module also manages components such as the CPU cores and memory. The load balancer determines the destination of processing by the current state of the nodes.

### 7.4 Improvement of thinning round-robin series

We describe the improvement in execution of WinG-CS using Cachalot. First, we separate the regular battles, currently one stage into three phases (Fig. 18).

After checking battle in acceptance processing, a strategy does a provisional battle. In the provisional battle, it joins about 10 prepared index strategies in a battle game. The WWG of index strategies is calculated as a criterion in advance. After finishing the provisional battle, the system calculates a temporary WWG of the strategy, and publishes the general ranking table as a flush report.

Next, during the submission suspension from 24:00 to 6:00, the system performs entry battles for all strategies submitted during the day. In the entry battle, a strategy competes about 100 strategies in a battle. At this time, the system equally chooses strategies from the WWG distribution. By this result, the system recalculates the WWG of all strategies, and publishes the general ranking table as daily result early in the morning. This is the initial WWG for strategies submitted the day before.

At that time, Cachalot estimates the execution load by the execution time of provision battle, and reallocates resources. After the next day, the strategies perform defensive battle as an opponent of other strategies in an entry battle. Strategies submitted in the early stages of the contest perform more defensive battles. The ordinary WWG of strategies change through these daily defensive battles.

### 7.5 Application to other systems

We describe the applications of Cachalot to other systems. First, each contest server receives a file from students. The server processes specific tasks for the exercise such as rewriting packages or renaming files as dependency injections. After processing, the server uploads the target file, necessary libraries, and execution recipe with setting into Cachalot. In Cachalot, resources are allocated in the processing node based on the execution recipe. This allocation is performed according to the load situation of the processing node. After the allocation, Cachalot starts the
execution in a container. After execution, Cachalot sends the execution result and information to the contest server, and destroys execution container. Cachalot also collects logs, such as execution time. In the future, we plan to realize load distribution based on the trends of each contest system.

8 CONCLUSIONS

We have practiced an applied Java programming exercise with a board game strategy. The exercise is in a contest style. Strategies are battled on a server, and their results and rankings are published. The students modify their code repeatedly using feedback during a contest period. We revise the contest management server to improve reliability and efficiency of battle processing. In particular, we adopted a virtual environment, and made the battle process efficient. We also revise the rating index WWG to refine the battle results. Furthermore, we reconsider the thinning round-robin series by the result in 2016. Thus, we arrange three stages of battle situations. In addition, we laid the foundations for distributed processing and the thinning round-robin series.

We propose the environment “Cachalot” for more effective distributed processing. Cachalot performs load distribution based on the statuses of the processing nodes. It also manages processing nodes with different performance. This realizes inexpensive, easy, and suitable load distribution for the educational site. In future work, we will try to improve the reliability and efficiency. First, we will implement functions for a revised thinning round-robin series to improve efficiency. Next, we will introduce code metrics as an internal evaluation to make students aware of not only the battle result but also the quality of code. This has learners promote continuous modification for refactoring. In addition, we will try to analyze learners’ actions during the contest.

ACKNOWLEDGEMENT

The authors would like to thank the students who participated in this exercise, and Dr. Takahashi and Mr. Okazaki, who assisted with this exercise. We would also like to thank Editage (www.editage.jp) for English language editing.
REFERENCES


