Goal-Oriented Requirements Analysis and an Extended Design Pattern using Scala for Artificial Intelligence Programming Contests

Kazunori Sakamoto
Dept. Computer Science
and Engineering
Waseda University
Tokyo, Japan
kazuu@ruri.waseda.jp

Hiroaki Hosono
Dept. Information Science
Tokyo Institute of Technology
Tokyo, Japan
wand1250@gmail.com

Seiji Sato, Hironori Washizaki, Yoshiaki Fukazawa
Dept. Computer Science and Engineering
Waseda University
Tokyo, Japan
{r0d8h8i0h@asagi., washizaki@, fukazawa@} waseda.jp

***Abstract*—An artificial intelligence programming contest with game software is one of the most effective way of learning programming. Contestants can spontaneously learn programming to win in such contests. Although our previous work helps to hold artificial intelligence programming contests, its effectiveness is limited owing to an insufficient requirement analysis and uses of an unrefined design pattern.**

In this paper, we report on ACM JavaChallenge 2012, that is an artificial intelligence programming contest. we elicit requirements on a contest with a goal-oriented requirements analysis and extend the state design pattern using Scala to hold JavaChallenge 2012. We evaluate JavaChallenge 2012 very highly by questionnaire investigation.

***Index Terms*—artificial intelligence programming contest; goal-oriented requirements analysis; design pattern; Scala; framework**

I. Introduction

A programming contest is an effective way of learning programming [1], wherein contestants can spontaneously learn programming to win in these contests. Numerous programming contests such as the ACM International Collegiate Programming Contest (ICPC) have been held [2]. In particular, an artificial intelligence (AI) programming contest related to game software can interest various participants who are interested in programming, games and just competition.

Existing work developed several online judge systems that aid to hold the programming contests such as the ICPC [3]–[5]. Other work analyzes such programming contests [2], [6], [7]. However, no existing work supports for AI programming contests related to game software.

Previously, we held four AI programming contests related to game software written in Java. We extracted reusable source code from our game software as a framework, called Game AI Arena (GAIA) [8]. GAIA were implemented using Java to satisfy requirements which were elicited using the quality model defined by ISO 9126. However, these requirements are not on AI programming contests but only game software. Thus, we overlooked a fault in game rules of our previous contests with GAIA. Moreover, the helps of GAIA for applying the state [9] is insufficient because the expression power of Java is limited.

In this paper, we report on our experience of holding JavaChallenge 2012 (JC2012), that is an AI programming contest related to game software, in conjunction with the ICPC Asia Regional Contest. Table I shows a summary of our programming contests including JC2012. We elicit requirements on such contests including JC2012 with a goal-oriented requirements analysis. We also extend the state design pattern using Scala to improve software quality. We evaluate JC2012 very highly by questionnaire investigation, and thus, confirmed the requirements were efficiently satisfied.

We investigate the following research questions:

- RQ1: What requirements of AI programming contests and game software should be satisfied?
- RQ2: Are these requirements on the contests satisfied efficiently by applying the extended pattern?

II. Goal-Oriented Requirement Analysis

Our previous requirement analysis treats only game software because the quality model defined by ISO 9126 is for software quality. In contrast, a goal-oriented requirements analysis can elicit requirements on both game software and game rules [10]. Thus, we carried out a goal-oriented requirements analysis.

Figure 1 shows the results of the requirement analysis for JC2012. The circles and squares indicate goals and implementations, respectively. We determined our two goals: fun and learning Java because one of the most important characteristics is fun [11]. The fun is also divided into six sub-goals (SGs): interesting, understandable, usable, attractive, reliable and fair.

SG1. Interesting and SG2. Understandable: We made game rules by integrating two well-known games, “The Settlers of Catan (Catan)” and “Galcon” to make game software interesting and understandable. The integration of well-known games make game software novel and understandable because contestants can easily understand well-known games and the integration brings novelty. While “Catan” is a board game,

TABLE I
SUMMARY OF OUR PROGRAMMING CONTESTS

Contest	Inspired game	Game type	Language for game software	Language for AI programs	Period	#Player	#Team
JC2009	Bomberman	Turn-based action	Java	Java	1.5 hours	2	35
JC2010	None	Turn-based action	Java	Java	4 hours	2	45
WR2010	None	Turn-based action	Java	Java, Ruby, Python, Scala	19 days	4	19
WG2011	Pac-Man	Turn-based action	Java	Java, Ruby, Python, C, C++, C#	36 days	4	109
JC2012	Galcon, Catan	Turn-based strategy	Java, Scala	Java	2 hours	6	34

“Galcon” is a digital game whose category is real-time strategy. We decided to use “Catan” as a basis adding the game elements of “Galcon” into it. We also removed game elements to make game rules simple because a oversimple integration increases complexity of game rules.

SG3. Usable and SG4. Attractive: Contestants want to properly use a character-based user interface (CUI) and a graphical user interface (GUI) in terms of usability. Sometimes contestants adapt machine learning to AI programs. While a CUI is suitable for programs for manipulating game software in such case, a GUI is suitable for showing games to audiences. Moreover, user want to use user-manipulation and AI-manipulation modes for different purposes. However, it is difficult to construct various UI modes because existing patterns using Java cannot modularize such modes well. Thus, we decided to use Scala to construct fine-grained modules for the modes. Moreover, we decided to extended an existing pattern for achieving separation of concerns (SoC). SoC aids designers to concentrate on designing attractive user interfaces.

SG5. Fair: The exclusive AI execution prevents from interceptions of other AI programs, and the read-only state prevents changing of the game state illegally. GAIA provides a feature to execute AI programs exclusively and several immutable classes. However, GAIA cannot help adding new immutable classes that represent the game state. Thus, we decided to use Scala to develop game software for JC2012 because functional programming languages including Scala prefers immutability and supports immutability.

SG6. Reliable: Faults do not occur in only game software but also in game rules. To find faults in game rules, we decided to review whole AI programming contests including game rules. We released game documents without game software to the public one week before JC2012. Moreover, we also decided to conduct beta testing for verifying JC2012. We asked programmers worked in sponsor companies to participate in our beta testing one week before JC2012.

Note that JavaChallenge should encourage contestants to learn Java. Thus, AI programs for JavaChallenge should be written in Java. We decided that our game software provides Java API for communicating with AI programs.

III. IMPLEMENTATION OF JAVACHALLENGE 2012

In this section, we describe how to satisfy the analyzed requirements by explaining the design and implementation of game rules and game software, called Asterobots.

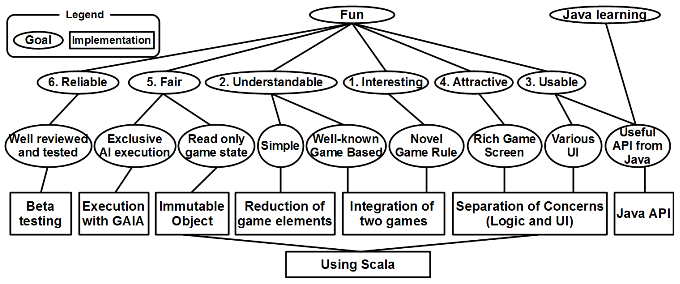

Fig. 1. Results of goal-oriented requirements analysis for JC2012

A. Game Rules for JavaChallenge 2012

We developed the game rules of Asterobots for JC2012 integrating “Catan” and “Galcon” to satisfy **SG1. Interesting** and **SG2. Understandable**. In “Catan”, the player earns points by gathering the resources and developing the land to win. In “Galcon”, the player occupies enemies’ planets, which produce soldiers, with soldiers to win. In Asterobots, the player also occupies enemies’ veins with robots to win. The veins produce materials similar to “Catan” and robots similar to “Galcon”. Although the soldier productivity of each planet is constant in “Galcon”, the material and soldier productivities of each vein can be upgraded by consuming materials similarly to “Catan” in Asterobots. Although “Catan” has four resources, Asterobots has three materials to make game rules simple in terms of **SG2. Understandable**.

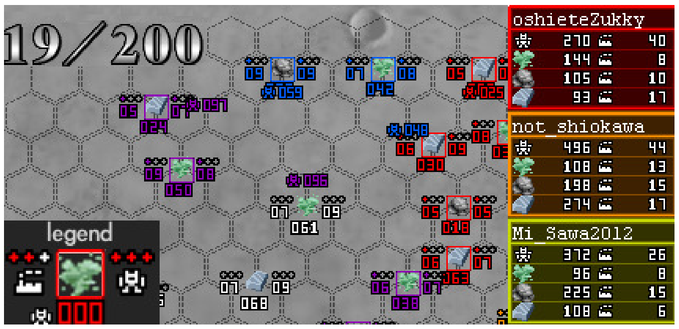

Fig. 2. Screen shot of Asterobots for JavaChallenge 2012

Figure 2 shows the game screen of Asterobots. We published the full rule document[1] on our web site[2].

[1]http://www.ai-comp.net/javachallenge2012/manual.pdf
[2]http://www.cs.titech.ac.jp/icpc2012/regional-contest/java-challenge-e.html

TABLE II
ABSTRACT CLASSES AND TRAITS WHICH REPRESENTS SCENES EXCEPT FOR BASIC CLASSES

Name	`Player`	`Main`	`Vein`	`Console`	`TextBox`	`Graphical`	`InitialRunner`	`MainRunner`
Module type	Class	Class	Class	Trait	Trait	Trait	Trait	Trait
Responcibility	Game rules	Game rules	Game rules	UI	UI	UI	UI	UI

B. Game Software for JavaChallenge 2012

Asterobots is developed using Scala but GAIA is developed using Java. Although Scala can work with Java, specialized methods for Scala are unnecessary for Java. We defined interfaces for all the Scala classes that are used by AI programs to hide the unnecessary methods.

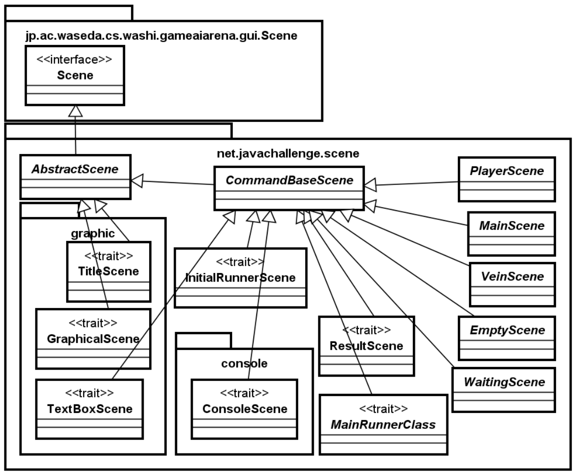

Fig. 3. Class diagram of scene classes

We utilize the state pattern to modularize scenes classes, which represent game scenes such as a title, a main and a end scene. However, scene classes designed by the state pattern strongly combine processing under game rules and rendering UIs on the scenes. We extend the state pattern using Scala, which provides a mixin feature through traits, to address this problem. The mixin feature provides better way of modularizing classes than an inheritance feature in Java. We divide a scene class into two traits of processing under game rules and rendering a UI using the mixin to satisfy **SG3. Usable** and **SG4. Attractive**.

Figure 3 shows the class diagram of scene classes in Asterobots. The `Scene`, `AbstractScene` and `CommandBaseScene` interface or classes provide basic features for representing game scenes. Talbe II shows the abstract classes and traits which represents scenes except for above-mentioned three classes. We abbreviates class and trait names in "Name" and the following by removing the `Scene` word (e.g. `Player` indicates `PlayerScene`). The `Player`, `Vein` and `Main` abstract classes are core classes for constructing game flow, which have only a responsibility of processing under game rules without rendering a UI. These classes have responsibilities which enable users to choose the number of players and player names, to select the initial veins and to play the game manipulating players, respectively.

The `Console` and `TextBox` traits provide methods for a CUI and a GUI, respectively. While the `Console` trait provides methods to read commands and write messages through standard I/O, the `TextBox` trait provides methods to show two text boxes for reading commands and writing messages. The `Graphical` trait provides methods to render the game screen using bitmap images. The `InitialRunner` and `MainRunner` provides methods to read commands from AI programs. We can use these classes construct game scenes with various features about the UI. For example, the mixin of the `Main` abstract class, `Console` and `MainRunner` traits creates an object that reads commands from AI programs and writes messages in the standard output.

Lists 1 and 2 show methods for constructing game scenes for the CUI and for the GUI, respectively. Lists 2 use the `Waiting` abstract class instead of the `Player` abstract class. The `Player` abstract class, which asks users for the player information, is unnecessary for AI programs because AI programs provide the player information through getter methods. Moreover, the `Waiting` abstract class allows users to control the timing for starting the game. In this way, we improved the state design pattern by utilizing the mixin feature in Scala.

List 1. Scala code constructing console game with user manipulation

```scala
def startConsoleGame() = {
  val end = new EmptyScene(null) with ResultScene
                 with ConsoleScene
  val main = new MainScene(end) with ConsoleScene
  val vein = new VeinScene(main) with ConsoleScene
  new PlayerScene(vein) with ConsoleScene
}
```

List 2. Scala code constructing graphical game with AI programs

```scala
def graphicalContestScenes() = {
  val main = new MainScene(null) with GraphicalScene
             with TextBoxScene with MainRunnerScene
  val wait = new WaitingScene(main)
             with GraphicalScene with TextBoxScene
  val vein = new VeinScene(wait) with GraphicalScene
             with TextBoxScene with InitialRunnerScene
  new WaitingScene(vein) with CompositeGraphicalScene
                         with TextBoxScene
}
```

IV. EVALUATION AND DISCUSSION

We evaluated how JC2012 and Asterobots satisfied the analyzed requirements, which are six sub-goals, using a questionnaire. To do so, our questionnaire asks about the interest of the game corresponding to **SG1. Interesting**, the clarity of the game rules corresponding to **SG2. Understandable**, the usefulness of the API corresponding to **SG3. Usable**, the graphics corresponding to **SG4. Attractive** and the overall.

TABLE III
RESULT OF QUESTIONNAIRE INVESTIGATION

	Interest of game	Clarity of game rules	Usefulness of API	Graphics	Overall
Very good	34	17	18	31	28
Good	5	14	20	7	8
Poor	2	9	3	3	5
Very poor	0	1	0	0	0

Table III shows the results of this questionnaire investigation. We obtained answers from 41 contestants.

SG1. Interesting: Ninety-five percent of the answered contestants felt that Asterobots is interesting. This is the best evaluation in this investigation. We integrated two well-known games, which are "Catan" and "Galcon", to make our game attractive and understandable. Several contestants commented that the game rules of JC2012 are well designed and very interesting. Although one contestant said that it may be preferable that be played by hand rather than AI programming, most of the contestants evaluated the game rules highly. Therefore, we successfully developed the interesting game rules.

SG2. Understandable: Seventy-six percent of the answered contestants felt that the game rule is clear. This value is relatively low in comparison with other values. Several contestants commented that the time is too short because contestants implement AI programs within only two hours in JC2012. Although we released game documents before JC2012, the contestants felt that the time is still insufficient.

Moreover, several contestants said that the game rules should be simpler. In particular, a contestant said that the integrated game rule is complex. This comments indicate an integration of two games essentially increase complexity and our reduction of game elements are insufficient. Therefore, we should consider the simplicity of the game rule.

SG3. Usable: Ninety-three percent of the contestants felt that our Java API is useful for implementing AI programs. This is a very high evaluation in this investigation. Although we developed Asterobots using Scala, we also use Java to define interfaces as pure Java code for Java API. No contestant commented on API. This indicates that our Java API had no problem. Therefore, we successfully provided the Java API through interfaces for the Scala program.

SG4. Attractive: Ninety-three percent of the contestants also felt that the game graphics is attractive. This value is also a very high evaluation in this investigation. We modularized the scene classes with the mix-in feature of Scala to make our game attractive. The SoCs between game logic and UIs helps us divide the labor appropriately. In particular, our designers concentrated on how to render the game screen. Therefore, we successfully provided the fine game screen and useful UIs.

SG5. Fair: No AI program intercepts other AI programs illegally. Therefore, we successfully achieved the fair game.

SG6. Reliable: We received seven issue reports in beta testing and fixed them. As a result, no fault occurs in both Asterobots and game rules. Therefore, we successfully provided the reliable game.

Overall: Eighty-eight percent of the contestants concluded that, overall, JC2012 is good. This is a high evaluation in this investigation. Most of the other evaluations are good except for the clarity of the game rule. Approximately half of the contestants commented "JC2012 is fun" or "Thank you for the very fun game". Therefore, JC2012 was successfully held.

As a result, we found that our analyzed requirements are enough because no other requirement was acquired from questionnaire investigation. Therefore, we answered RQ1 by showing the requirements. Moreover, we confirmed the extended design pattern help to satisfy the requirements efficiently because the results of the evaluation except for SG2. Understandable are very high. Therefore, we answered RQ2 by showing this results. However, we should deal with the simplicity by reducing dramatically more game elements or by introducing novelty without an integration of games. We plan to elicit better way of creating simple game rule in future.

V. Conclusion

We reported on JavaChallenge 2012 with our previous programming contests and GAIA. We carried out a goal-oriented requirements analysis and extended the state pattern using Scala to satisfy the requirements efficiently. We evaluated JavaChallenge 2012 and Asterobots using a questionnaire. Most of the answers are quite good with respect to the analyzed requirements except for the understandable. We plan to elicit better way of creating simple game rule in future to improve understandability. In this way, we concluded JavaChallenge 2012 was successfully held.

References

[1] N. V. Shilov and K. Yi, "Engaging students with theory through acm collegiate programming contest," *Commun. ACM*, vol. 45, pp. 98–101, September 2002.

[2] A. Trotman and C. Handley, "Programming contest strategy," *Computers & Education*, vol. 50, no. 3, pp. 821 – 837, 2008.

[3] J. P. Leal and F. Silva, "Mooshak: a web-based multi-site programming contest system," *Softw. Pract. Exper.*, vol. 33, no. 6, pp. 567–581, May 2003.

[4] A. Kurnia, A. Lim, and B. Cheang, "Online judge," *Computers & Education*, vol. 36, no. 4, pp. 299 – 315, 2001.

[5] Y. Luo, X. Wang, and Z. Zhang, "Programming grid: a computer-aided education system for programming courses based on online judge," in *Proceedings of the 1st ACM Summit on Computing Education in China*, ser. SCE '08. ACM, 2008, pp. 10:1–10:4.

[6] N. Gulley, "Patterns of innovation: a web-based matlab programming contest," in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '01. ACM, 2001, pp. 337–338.

[7] M. Forišek, "The difficulty of programming contests increases," in *Proceedings of the 4th International Conference on Informatics in Secondary Schools - Evolution and Perspectives*, ser. ISSEP '10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 72–85.

[8] K. Sakamoto, A. Ohashi, H. Washizaki, and Y. Fukazawa, "A framework for game software which users play through artificial intelligence programming (in japanese)," *IEICE Transactions*, vol. 95, no. 3, pp. 412–424, mar 2012.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[10] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, 2001, pp. 249 –262.

[11] P. Salenieks and J. Naylor, "Professional skills assessment in programming competitions," *SIGCSE Bull.*, vol. 20, no. 4, pp. 9–14, Dec. 1988.