

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.



SCHOOL OF ENGINEERING AND ADVANCED  
TECHNOLOGY

---

# Adding Traceability to an Educational IDE

---

A thesis presented in partial fulfilment of the requirements for the Master degree  
in Computer Science at Massey University, Manawatu,  
New Zealand

*Author:*

Li SUI

*Supervisor:*

A/Pro Jens DIETRICH

A/Pro Eva HEINRICH

August 12, 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
<b>2</b>	<b>Background</b>	<b>14</b>
2.1	Challenges in teaching/learning programming and existing approaches	14
2.1.1	Gamification . . . . .	14
2.1.2	Game classification . . . . .	16
2.1.3	Existing educational platforms . . . . .	17
2.2	Related work . . . . .	21
2.2.1	SoGaCo . . . . .	22
2.2.2	PrimeGame . . . . .	23
2.2.3	PrimeGame strategy classification . . . . .	25
2.3	Conceptual foundations . . . . .	30
2.3.1	Notional machines . . . . .	30
2.3.2	Conceptual model . . . . .	32
2.4	Technical foundations . . . . .	34
2.4.1	Continuations . . . . .	34
2.4.2	The Java debug interface . . . . .	35
2.4.3	Instrumentation libraries . . . . .	36
2.4.4	Instrumentation . . . . .	38
<b>3</b>	<b>A Layered and Reversible Notional Machine</b>	<b>44</b>
3.1	Bi-directionality . . . . .	44
3.2	Two level hierarchy . . . . .	45
<b>4</b>	<b>Design and Implementation</b>	<b>48</b>
4.1	Client . . . . .	48

4.2	Server . . . . .	51
4.2.1	Snapshots . . . . .	51
4.2.2	Implementing tracing using source code instrumentation . . .	53
4.2.3	Implementing tracing using byte code instrumentation . . .	54
4.2.4	Discussion . . . . .	57
4.2.5	Compression methods and encoding schemes . . . . .	58
<b>5</b>	<b>Experiment Analysis</b>	<b>65</b>
5.1	Methodology . . . . .	65
5.1.1	Build performance measurements . . . . .	65
5.1.2	Runtime performance measurements . . . . .	66
5.2	Build performance (source code instrumentation) . . . . .	67
5.3	Build performance (byte code instrumentation) . . . . .	69
5.4	Runtime performance . . . . .	71
<b>6</b>	<b>Conclusion and Future work</b>	<b>78</b>
6.1	Conclusion . . . . .	78
6.2	Future work . . . . .	79
6.2.1	End user validation . . . . .	79
6.2.2	Compression improvement vs Game play strategy . . . . .	80
6.2.3	Other games . . . . .	80
<b>7</b>	<b>Appendix</b>	<b>82</b>
7.1	Java Debug Interface Code Example . . . . .	82
7.2	JavaFlow Code Example . . . . .	87
7.3	ASM byte code . . . . .	88
7.4	BlackMamba . . . . .	103
7.5	Java Parser . . . . .	108

## List of Figures

1	Thesis Structure . . . . .	12
2	Greenfoot Interface . . . . .	19
3	If condition in Blockly . . . . .	20
4	PrimeGame Board . . . . .	24
5	Cautious VS Greedy . . . . .	33
6	Prime number VS Cautious . . . . .	33
7	Java Platform Debugger Architecture [29] . . . . .	35
8	Example Code before Instrumentation . . . . .	39
9	Abstract Syntax Tree . . . . .	40
10	Example Code after Instrumentation . . . . .	41
11	Eclipse Debugger . . . . .	44
12	Program code Comprehension: Assignment . . . . .	45
13	Program code Comprehension: Method Call . . . . .	46
14	Game strategy comprehension: State Changing . . . . .	46
15	Visual plus texture view . . . . .	47
16	Editor page . . . . .	49
17	Testing page . . . . .	49
18	Bot Selection . . . . .	50
19	Choose Who Plays First . . . . .	51
20	Snapshot . . . . .	52
21	Method Flow of Source Code Instrumentation . . . . .	53
22	Nested Map data structure for Source Code Instrumentation . . . . .	54
23	Byte code Instrumentation . . . . .	55
24	Example code: Miss capture on different JDK version . . . . .	57
25	Level of Access: depth 2 . . . . .	59

26	LinkedList structure . . . . .	60
27	Baseline Encoder Structure . . . . .	61
28	Memory usage: Build Bots with source code instrumentation(MB) .	68
29	Memory usage: Build Bots without instrumentation(MB) . . . . .	69
30	Memory usage: Build Bots Byte code instrumentation(MB) . . . . .	70
31	Producing extra contents . . . . .	76

## Listings

1	Randomly strategy . . . . .	26
2	Cautious strategy . . . . .	26
3	Greedy strategy . . . . .	27
4	Largest prime number strategy . . . . .	27
5	Max-gain strategy . . . . .	28
6	For loop . . . . .	30
7	Debug For loop . . . . .	31
8	Bad Writing Habit . . . . .	43
9	A variable that stays the same across all the moves . . . . .	62
10	A variable that stays the same in a single move . . . . .	62
11	Changing variable . . . . .	62
12	Adding a variable using edit distance . . . . .	63
13	Remove a number in an array using the edit distance . . . . .	64
14	Remove a number in an array using the tree edit distance . . . . .	64
15	Monitor field . . . . .	82
16	Connect to VM . . . . .	84
17	Monitored Code . . . . .	86
18	JavaFlow Code Example . . . . .	87
19	(JDK version: 1.8.0_60)Bytecode Demonstration . . . . .	88
20	BlackMamba Source Code . . . . .	103
21	JavaParser Code Example . . . . .	108

## List of Tables

1	List of educational platforms . . . . .	17
2	Local Variable Table . . . . .	55
3	Time for Building bots using source code instrumentation (milliseconds) . . . . .	67
4	Time for Building bots using byte code instrumentation(milliseconds) . . . . .	70
5	Different Encoding and Compression Result in Time(milliseconds) . . . . .	71
6	Different Encoding and Compression Result in Memory(KB). (blue:depth 2, red:depth 3) . . . . .	72



## Abstract

High dropout and failure rate in introductory programming courses indicate the need to improve programming comprehension of novice learners. Some of educational tools have successfully used game environments to motivate students. Our approach is based on a novel type of notional machine which can facilitate programming comprehension in the context of turn-based games. The first aim of this project is to design a layered notional machine that is reversible. This type of notional machine provides bi-directional traceability and supports multiple layers of abstraction. The second aim of this project is to explore the feasibility and in particular to evaluate the performance of using the traceability in a web-based environment. To achieve these aims, we implement this type of notional machine through instrumentation and investigate the capture of the entire execution state of a program. However, capturing the entire execution state produces a large amount of tracing data that raises scalability issues. Therefore, several encoding and compression methods are proposed to minimise the server work-load. A proof-of-concept implementation which based on the SoGaCo educational web IDE is presented. The evaluation of the educational benefits and end user studies are outside the scope of this thesis.

## **Acknowledgments**

Firstly, I would like to express my sincere gratitude to my supervisor A/Prof. Jens Dietrich for the continuous support of my study and related research, for his patience, motivation, and immense knowledge. His guidance has pointed me in the right direction throughout the work.

I also would like to express my appreciation to my co-supervisor A/Prof Eva Heinrich for her patient guidance and advice on computer science education. My sincere thanks also goes to Prof. Manfred Meyer and Mr. Johannes Tandler for their active collaborations on writing related papers.

Finally, I must express my very profound gratitude to my parents with unfailing support and continuous encouragement.