

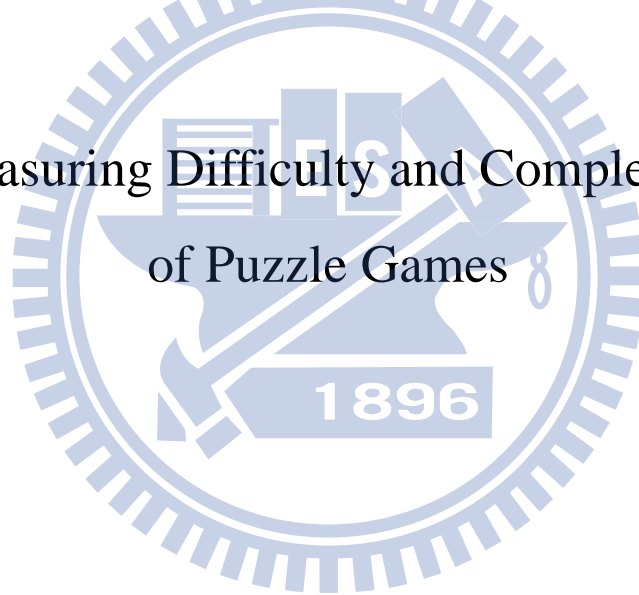
國立交通大學

多媒體工程研究所

碩士論文

益智遊戲難度與複雜性之衡量

Measuring Difficulty and Complexity  
of Puzzle Games



研究生：張景照  
指導老師：孫春在 教授  
中華民國 100 年 6 月



國立交通大學

多媒體工程研究所

碩士論文

益智遊戲難度與複雜性之衡量

Measuring Difficulty and Complexity  
of Puzzle Games

研究生：張景照  
指導老師：孫春在 教授  
中華民國 100 年 6 月

益智遊戲難度與複雜性之衡量

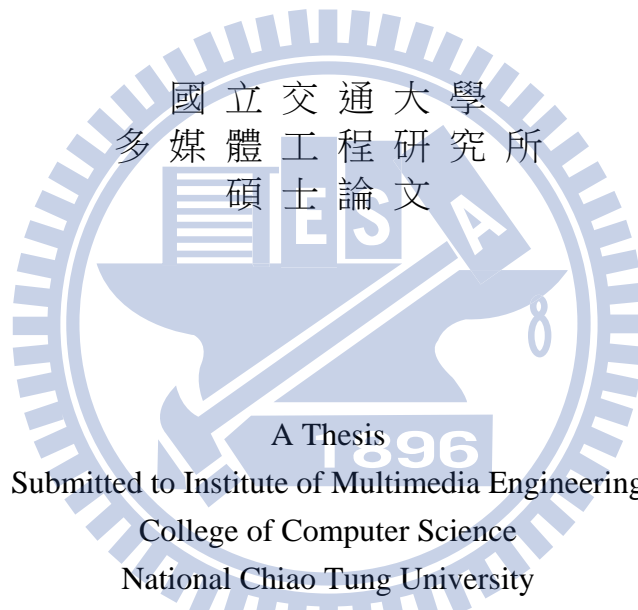
Measuring Difficulty and Complexity  
of Puzzle Games

研究生：張景照

Student : Ching-Chao Chang

指導老師：孫春在

Advisor : Dr. Chuen-Tsai Sun



A Thesis  
Submitted to Institute of Multimedia Engineering  
College of Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Master  
In  
Computer Science

June 2011

Hsinchu, Taiwan, Republic of China

中華民國 100 年 6 月

# 益智遊戲難度與複雜性之衡量

學生：張景照

指導教授：孫春在 博士

國立交通大學  
多媒體工程研究所

## 中文摘要

如何對關卡難度進行排序？這個問題是所有益智遊戲(puzzle game)設計師所需面對的最重要課題。在以前，人們常常先分析解開該遊戲所使用的技巧有哪些，並依技巧掌握的難度來決定最後的排序，不過，這個方法的問題點在於太過依賴遊戲本身的特性而造成計算上的困難與難以理解。——設計者必需對該遊戲非常精通才行。

其實，要找出一個益智遊戲的解，就像是在替迷宮找出口一樣，裡面充滿了許多選擇(Choice)及死路(Dead Ends)，然而，不同於「真實」的迷宮問題，益智遊戲的難度決定於該關卡所需的洞察力(Insight)。從複雜系統的觀點來看，這些選擇及死路會隨著系統複雜度的上升而湧現，進而影響到玩家找到答案的難度。因此，本研究不同於以往的做法，試圖以衡量湧現狀況做為出發點，設計出一個適合所有益智遊戲的複雜度衡量模型。

本研究總共規畫了四個實驗，並使用了消除方塊(Cross Block)及數獨(Sudoku)這二個益智遊戲來進行複雜度的衡量及比較。最後的實驗結果顯示了我們所提出的複雜度計算模型，在數獨的複雜度排序上與「台灣數讀發展協會」上的難度排序結果能夠達到 86%的相似度。

關鍵詞：益智遊戲、難度衡量、複雜度衡量、洞察

# Measuring Difficulty and Complexity of Puzzles Games

Student : Ching-Chao Chang

Advisor : Dr. Chuen-Tsai Sun

Institute of Multimedia Engineering

National Chiao-Tung University

## Abstract

How do we sorting puzzle levels according to their difficulty? This is most important problem for all puzzle game designer. In past time, game designer must analysis the techniques used by the puzzle, and then can use the result to sort their difficulty ranks. However, this method depends on specific game feature that are very difficult to calculate and understand because designer must master the game first.

Basically, find out a solution for a puzzle just like find out a way for a maze, there are many choices and dead ends. But, different from “real” maze problem, the difficulty of a puzzle decided by the insight it required. From the aspect of complex system, emergent phenomenon describes when overall system complex increase then amount of choice and dead ends also changed according to some principle that will affect how difficult a puzzle is.

In this paper we propose a new approach to calculating puzzle complexity, one based on approximating player ability to produce insights that lead to puzzle completion. Our test results indicate an 86% sorting similarity rate.

Key Word: Puzzle Game, Difficulty Measure, Complexity Measure, Insight

## 誌 謝

經過了無數個失眠的夜晚，這篇論文終於誕生了。特別是經歷過日本 311 地震與輻射外洩事件之後，更讓我覺得能夠平安回來並順利的將論文完稿是一件非常幸運的事。

首先，我必須要感謝指導我的孫春在老師，謝謝他同意讓我在碩二這麼關鍵的時期能夠到日本一圓留學的夢想。雖然因為地震的關係提早了半個學期回來，但在東京大學交換學生這半年的時間讓我體驗到非常寶貴的生活經驗。另外，也非常很高興能夠跟在孫老師身邊進行遊戲相關的研究，讓我有機會從學術的觀點進入這塊領域，並給予我許多以前從未思考過的方向。能在孫老師身邊做研究，真的是一件非常享受的事。

另外也謝謝學習科技實驗室的同學們，在交大的這段時光能夠跟你們在一起渡過真的很高興，特別讓我懷念的是每天聚在一起玩遊戲、打 Board Game 的那段時光，真的是非常的快樂。也很感謝東京大學近山老師及其實驗室的同學們，在日本的這段期間受到你們不少的照顧。

此外，也謝謝我的家人，以及其他關心我的朋友們。沒有你們的支持，這篇論文是難以完成的。在此，致上最深的謝意。



## Index

中文摘要.....	i
Abstract.....	ii
誌謝.....	iii
Index.....	iv
Figure Index.....	vi
Table Index.....	vii
Code Index.....	vii
Chapter 1: Introduction.....	1
Section 1.1: Background.....	2
Section 1.1.1: Digital Game: What is difficult?.....	2
Section 1.1.2: Digital Game: Dynamic Difficult Adjust.....	4
Section 1.1.3: Digital Game: Player Modeling and DDA .....	7
Section 1.1.4: Puzzle Game: What is Puzzle? —— Meaning of Play....	8
Section 1.1.5: Puzzle Game: What is Puzzle? —— Component.....	9
Section 1.1.6: Puzzle Game: Difficult Measure and Sorting .....	11
Section 1.1.7: Puzzle Game: Complexity Theory .....	12
A. Computational Complexity .....	12
B. Complex System and Emergence .....	13
Section 1.2: Motivation: Challenge in Puzzle Game Sorting .....	16
Section 1.3: Motivation: Mobile Game, Market and Puzzle Game .....	16
Section 1.4: Goal.....	17
Section 1.5: Contribution .....	17
Chapter 2: Literature Review.....	19
Section 2.1: Tree Search .....	19
Section 2.2: Local Search.....	25
Section 2.3: Simulated Annealing .....	26
Section 2.4: Pseudo-Random and Real-Random .....	30
Section 2.5: Game, Digital Game and Media .....	30
Section 2.6: Flow Theory .....	32
Section 2.7: Three-part rule model .....	32
Chapter 3: Method .....	34
Section 3.1: Experiment One .....	34
Section 3.2: Experiment Two .....	38
Section 3.3: Experiment Three.....	39
Section 3.4: Experiment Four .....	40
Chapter 4: Experiment .....	41
Section 4.1: Experiment One .....	41



Section 4.1.1:	Phase 1: Random Generated Puzzle .....	41
Section 4.1.2:	Implement Phase 1: Random Generated Puzzle .....	41
Section 4.1.3:	Implement Phase 2: Calculate Branch and Dead Ends.....	43
Section 4.1.4:	Result of Phase 3: Branch and Dead End .....	45
Section 4.1.5:	Implement Phase 4: Calculate Complexity.....	46
Section 4.1.6:	Result of Phase 5: Complexity Sample Mean .....	48
Section 4.1.7:	Conclusion .....	51
Section 4.2:	Experiment Two .....	53
Section 4.2.1:	Phase 1: Select Puzzle Levels .....	53
Section 4.2.2:	Result of Phase 3: Average difficulty and Sorting .....	53
Section 4.2.3:	Implement: Calculate Sorting Similarity .....	54
Section 4.2.4:	Result of Phase 5: Sorting Similarity.....	56
Section 4.2.5:	Conclusion .....	56
Section 4.3:	Experiment Three.....	57
Section 4.3.1:	Phase 1: Select Puzzle in Each Rank.....	57
Section 4.3.2:	Result Phase 3: Calculate Branch and Dead Ends.....	58
Section 4.3.3:	Result Phase 4: Compute Complexity .....	58
Section 4.3.4:	Result Phase 5: Compare Rank Result .....	61
Section 4.3.5:	Conclusion .....	62
Section 4.4:	Experiment Four .....	62
Section 4.4.1:	Phase 1: Select Training Sample.....	62
Section 4.4.2:	Implement Phase 2: Parameter Tweak.....	63
Section 4.4.3:	Result of Phase 2: Parameter Tweak.....	64
Section 4.4.4:	Result of Phase 3: Calculate New Complexity .....	64
Section 4.4.5:	Result of Phase 4: Compare Rank Result .....	67
Section 4.4.6:	Conclusion .....	67
Chapter 5:	Conclusion .....	68
Section 5.1:	Complexity Sorting and Difficulty Mapping .....	68
Section 5.2:	Measuring Digital Game Complexity .....	68
Appendix A:	Puzzles in Experiments.....	70
Appendix B:	Collection of Pure Puzzle.....	72
Appendix C:	More Result of Experiment One.....	75
Appendix D:	Calculate Branch and DeadEnds.....	76
Appendix E:	Game Data Format .....	76
Appendix F:	Puzzles in Experiment Two.....	80
Reference.....		82

## Table Index

Table 1 Max values in puzzle database. ....	46
Table 2 Complexity and Difficulty result. ....	53
Table 3 Result of match and sorting similarity. ....	56

## Figure Index

Figure 1 Two type of DDA. ....	6
Figure 2 Concept Model of Puzzle developed by Scott Kim. ....	9
Figure 3 Five genres of puzzle ....	10
Figure 4 Emergence Phenomenon Example ....	15
Figure 5 Example of a puzzle Solution. ....	20
Figure 6 Example of tree search.....	21
Figure 7 Concept of minimum local search ....	26
Figure 8 Flow Chat of Simulated Annealing. ....	27
Figure 9 Game Taxonomy by Media.....	31
Figure 10 Mental State in flow theory. ....	32
Figure 11 <i>Cross Block's</i> Puzzle Game Space. ....	35
Figure 12 Branch and Dead End Calculating Process.....	36
Figure 13 BD-Complexity Calculating Model ....	36
Figure 14 Example of random generated process of puzzle "cross block". ....	43
Figure 15 Calculate for Branch and Dead ends by using answer node.....	44
Figure 16 Average Number of Branches of each step.....	45
Figure 17 Average Number of dead ends of each step.....	45
Figure 18 Average complexity of each step ....	48
Figure 19 Result of Average all solved step complexity. ....	48
Figure 20 More Detail of complexity mean in puzzle game space. ....	49
Figure 21 Average complexity before step 16.....	49
Figure 22 Average complexity before step 11.....	50
Figure 23 Ratio of basic difficulty in puzzle database. ....	51
Figure 24 Complexity Average of each Crossblock difficulty level.....	52
Figure 25 Complexity and Difficulty rank result.....	54
Figure 26 <i>Sudoku</i> Puzzles provides in TSA. ....	57
Figure 27 Average branch for each difficulty levels. ....	58
Figure 28 Average dead ends for each difficulty levels.....	58
Figure 29 Average Degree of Complexity for Each Difficulty Level ....	59
Figure 30 Puzzle Samples Sorted by Complexity ....	60
Figure 31 Rank of complexity Sorting for each puzzle samples.....	60
Figure 32 Process of select sample from puzzle database as sorting list. ....	61

Figure 33 Result of Sorting Similarity .....	61
Figure 34 Training sample select process.....	63
Figure 35 Error and iteration of simulated annealing.....	64
Figure 36 Result of parameter Band D adjusts over 1500 iteration. ....	64
Figure 37 Average complexity for each difficulty level after parameter tweak.....	65
Figure 38 Complexity of each puzzle sample after parameter tweak. ....	65
Figure 39 Rank of complexity sorting after parameter tweak. ....	66
Figure 40 Sorting Similarity after training.....	67

### Code Index

Code 1 Data Structure Node. ....	22
Code 2 Implementation for BFS tree search algorithm. ....	22
Code 3 Implement for expand function.....	23
Code 4 implement for graph search. ....	24
Code 5 Boltzman distribution for simulated annealing. ....	28
Code 6 Temperature decrease function for simulated annealing. ....	28
Code 7 Accept function for simulated annealing.....	29
Code 8 Implement for random generate <i>cross block</i> .....	42
Code 9 Implement normalize function for BD-Complexity Calculate Model .....	47
Code 10 Implement for sorting similarity. ....	55
Code 11 Implement for energy function in Simulated Annealing. ....	63

# Chapter 1: Introduction

How do we measure a puzzle's level of difficulty? Game designers may employ their expertise and feedback from players to rank the puzzles they have designed. They may also take a simpler but nevertheless systematic way to reach the goal, for example, adopting a game-specific feature as criterion and sort the puzzles out. A handy measurement is the shortest step required to solve the puzzle, the less steps required, the easier the puzzle.

In practice, it may be fun enough for most gameplay, because players can reach the flow experience (M. Csikszentmihalyi, 1998) when they conquer one puzzle after another arranged in ascending level of difficulty. When the difference in difficulty is not that clear, the game producer can put puzzles into categories of difficulty, e.g., easy, medium, and hard, to allow the players select their current goal and test their skills. Obviously, there is a risk factor in such approach. When puzzles arranged in a wrong order and the players are not capable of picking the right challenge for their current skills, they may endure a long period of frustration and anxiety, so as to give up the game. Therefore, it is desirable to have a way to 'optimize' the gaming experience by arranging puzzles in a smoothly ascending order of difficulty so that most players can enjoy an uninterrupted challenge/skill upgrading experience.

Our goal of this research is to propose a general-purposed method to develop a function that can arrange puzzles in ascending level of difficulty. However, before trying to approach that goal, some questions need to be answered. The first is: what is difficult?

## Section 1.1: Background

*” Game is a system in which players engage in an artificial conflict, defined by rules that result in a quantifiable outcome.”—Rules of Play (Katie Salen & Eric Zimmerman, 2003)*

### Section 1.1.1: Digital Game: What is difficult?

Based on flow theory, we know there has two key elements: Challenge and Skill. In order to discuss about what is difficult, first, I give an assumption below:

Assumption 1: Challenge is a non-linear increase function relate to how many obstacles are designed in game.

Assumption 2: Skill is a non-linear increase function relate to your performance in the game.

As we known, flow theory point out when **challenge meets skill, player is in the state, called flow**. What is this phrase means? Obviously, flow state equal to proper difficulty for player. Degree of difficulty is highly related to challenge and skill. Therefore, I give third assumption to define what is difficult:

Assumption 3: Difficult is a non-linear function relative to the combine of challenge and skill.

Furthermore, if we want to design the function describes above, C-style function prototype may look like following:

1. float Challenge(int numberOfObstacle);
2. float Skill(int playerPerformance);
3. float Difficult(int challenge, int skill);

But, there comes some problem.

How to design difficult function? In order to do that, we must ask: **what is the relation between challenge and skill?** Although we can say relation between difficult and challenge is positive relation, difficult and skill are negative relation, however, relation between challenge and skill are ambiguity. Try to consider following example:

1. If we know both difficult and skill are high, then we can say challenge  $>$  skill, because player feel game are difficult. Therefore, reduce obstacle in game can decrease difficulty.
2. If we know both difficult and challenge are low, then we can say challenge  $<$  skill, because player feel game are easy. Therefore, add obstacle in game can increase difficulty.
3. But, if both challenge and skill are high or low, how can we say? It just means challenge just meet player's skill, and has a "**proper difficulty**", there seems doesn't has any relation between challenge and skill.

If we want to figure out relation between challenge and skill, and then calculate out difficult will be first task. But, there will be some trouble. Why? Since **difficult** is an **objective** concept depends on how people feeling about, thus, if we want to tell it degree, we must compare with player's **previous experience**.

For example, when we read the introduction or manual of a game, there doesn't emerges any feeling call "difficult".—although we may feel the complexity of game rule—After we get into the game, finally, we can tell the degree of difficult compare to other similar game or problems.

Of course, some apparatus, like Eye Tracking, SCR (Skin Conductance Response), EEG (Electroencephalography), to collect physical reply when playing the game that can used to data mining on people's cognitive level of feeling to help us find out what is difficult. For example, there are some research trying to find out player experience to explain the degree of playability for a game (Lennart E. Nacke et al., 2009; Lennart Nacke & Craig A. Lindley, 2008). Of course, it may be used to measure the degree of difficult, but these methods have too much disadvantage when apply in real game design (you can't always ask player to equip physical apparatus when they play the game), therefore, goes beyond our research, so we don't want to dig into this method for discuss about difficulty.

Before close this section, here is summary: we can't design difficult function without knowing the relation between challenge and skill, if we want to know this relation, we must use some subjective method to measure the player's feeling, such as Eye Tracking, SCR, or EEG, to decide this dynamic relation.

From discussion above, we know difficult is a relative concept that based on

player past experience. And then, next problem we want to focus on is: how do we manipulate such concept in digital game? In next section, I will introduce a technique, Dynamic Difficult Adjust (DDA), which trying to create flow state by adjusts difficult based on player's skill.

### **Section 1.1.2: Digital Game: Dynamic Difficult Adjust**

Static difficulty is a popular method that almost single player game uses it for player to adjust game difficulty. For example, in FTG or STG, we can always set difficulty into Very Easy, Easy, Normal, Hard or Very Hard in system setting. But, there exists several problems when we look this method from the viewpoint of flow theory. First, difficult must manually set before game start by game designer. Second, difficult is fixed while playing. Third, therefore, it can't auto-adjust according to player skill, that may trouble in crate flow state. Four, the most important, the feeling of difficult is relative to game designer but not player. From the reason describe above, there comes the research, call Dynamic Difficult Adjust (DDA).

DDA is based on flow theory. It core concept is to adjust difficult according to player skill that try to adjust the game to “**proper difficulty**” for player. Therefore, how to design challenge and skill function are important for this method. As we defined previous, calculate obstacle in a game maybe an easy task for challenge function, but how do we measure player performance for skill function? Fortunately, every game must have a quantifiable outcome. Like game play score or player remaining health, it is useful to help us to decide player performance.

If player skill > challenge, then add obstacle in game can increase difficulty. If player skill < challenge, then reduce obstacle in game can decrease difficulty. Notice, how well of skill function designed effects the performance of DDA. And skill function is affected by the game design. Successful game design must bring meaningful play to player. Katie Salen and Eric Zimmerman's (2003) book “*Rules of Play, chapter3*”, define meaningful play as descriptive and evaluative:

The descriptive definition addresses the mechanism by which all games create meaning through play. The evaluative definition helps us understand why some games provide more meaningful play than others.

The descriptive definition of meaningful play: Meaningful play in a game emerges from the relationship between player action and system outcome; it is the

process by which a player takes action within the designed system of a game and the system responds to the action. The meaning of an action in a game resides in the relationship between action and outcome.

The evaluative definition of meaningful play: Meaningful play is what occurs when the relationships between actions and outcomes in a game are both discernable and integrated into the larger context of the game.

Discernability means that a player can perceive the immediate outcome of an action. Integration means that the outcome of an action is woven into game system as a whole.

From descriptive definition above, we know player performance can be measure through the relationship between action and outcome, if game generate good outcome, means player has good performance in game.

In addition to, evaluative definition indicate where we can find player's performance. Discernability means outcome of action takes in game can help DDA to adjust difficult immediately. Integration means outcome of action takes in game can help DDA to generate next game level according to player overall performance. I call former as Immediate Difficult Adjust (IDA), which try to create an even game—but, beatable—according to player agency and tension of game. And later as Content Difficult Adjust (CDA), which combine Procedural Content Generation (PCG) technique—means use program to auto-generate game content—to consider overall difficult balance to generate levels. I summarize these two DDA methods in Figure 1.



## Two type of DDA

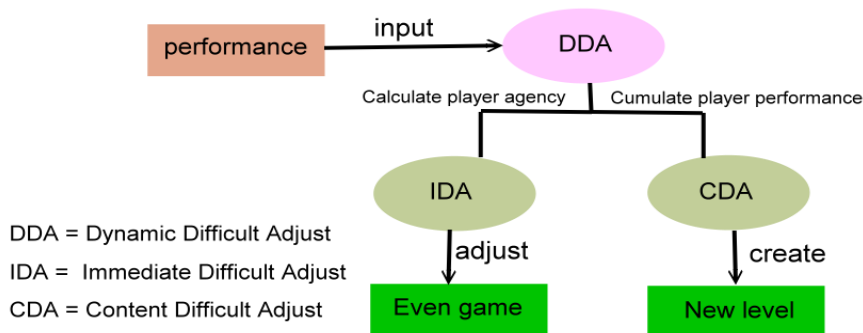


Figure 1 Two type of DDA.

There have some DDA example:

1. Hunicke Robin and Chapman Vernell (2004) have applied DDA to FPS, Half Life, use IDA to adjust difficult.
2. Ben Weber (2010), in his project “*Infinite Mario with dynamic difficulty adjustment*” use CDA to generate new level according to player performance.

I don’t want to go into detail of DDA because it will take several pages to discuss it, but I think it is important to understand how to manipulate the feeling of difficult in this research.

Although we introduce much about DDA, but there still have an ambiguity on challenge and skill function. Before we design these two functions, we must consider following situation in order to decide detail implements method:

1. How players are affected by obstacles? For example, the powerful monster always increases the challenge and healing potions can decrease.
2. How player performance measured by actions they takes in game? In some game systems, players receive more positive performance feedback when they choose certain actions over others.

Next, let’s examine more about how to building challenge and skill function from the view point of player modeling.

### Section 1.1.3: Digital Game: Player Modeling and DDA

The purpose of player modeling is trying to find out the relationship between obstacles, player action, and player feeling about game in order to do content creation task. In Pedersen's research, they collect following three data to validate how player feeling about an action game, *Infinite Mario Bros* (C. Pedersen, J. Togelius, & G. N. Yannakakis, 2010):

1. Controllable Feature: like number of gaps and average width of gaps, spatial diversity of gaps...etc., that can be controlled by game designer or level generate program. This part is related to obstacle.
2. Gameplay characteristics: like number of jump, time you complete the level, item you collected...etc., that can only be collected when a player play the game. This part is related to the action of player takes in game.
3. Questionnaire: After finish a pair of level, player is asked to rank the games in order of emotional preference. Pedersen define following six emotions: Fun, Challenge, Frustration, Predictability, Anxiety, and Boredom. Their questionnaire may looks like this: 1. Level A has more challenge then level B. 2. Both games were equally challenge. 3. Neither of two game felt challenge.

According to collected data, they calculated correlation coefficient. Therefore, we can actually tell the degree of obstacle or actions affects player mental state in "*Infinite Mario Bros*". For example: whether player complete level has "-0.5" negative relation and average of gap width has "0.5" positive relation to challenge. For detail experiment and result, you can find it in their research paper "*Modeling Player Experience for Content Creation*" (C. Pedersen, et al., 2010). Based on the degree of relations, DDA will perform more accurately according to player skill. There has much Player Modeling application, like create personalized race track in racing game (Ratan K. Guha, Erin Jonathan Hastings, & Kenneth O. Stanley, 2009; Togelius. J., De Nardi, & Lucas, 2007), and adapt agent behavior to human player (Kang. Yilin & Tan. Ah-Hwee, 2010).

Here, we finally come into our topic. Is puzzle game can apply such method for sorting difficulty? What is controllable feature? And what is gameplay characteristic? How do we design our questionnaire in order to measure the relationship between obstacles, player action, and player feeling about game? For answer the question, we must ask: 1. what is puzzle? 2. What's difference between other digital games?

### Section 1.1.4: Puzzle Game: What is Puzzle? — Meaning of Play

It is a good start point to quote from Scott Kim's (2003, 2008) presentation slide on Game Development Conference (GDC), "*The Puzzlemaker's Survival Kit*": "*A puzzle is a problem that is fun to solve—as opposed to everyday “problems”—and has a right answer—as opposed to a game (no answer) or a toy (no goal).*" This definition not only explains what puzzle is but also describe the motivation of why people play it. From the definition, we know puzzle is a problem but different from everyday problems we encounters. Although problem means something trouble and undesired, however, we will feel fun to solve it.

Why? Scott Kim (2003) describe: "*puzzle game symbolizes our desire to find order in the universe.*". When we see something in disorder states, people always want to control it—that is why puzzles display itself as complex forms and simple forms after being solved. Furthermore, as James Paul Gee (2005) says: puzzle supply order, control and workable environment, therefore, "goal" and "right answer" are proved in puzzle worlds, not like a toy, puzzles are encouraging us to solve and control those problems. —Lusory attitude, the term mentioned by Bernard Suits (2005), in the book "*The Grasshopper: Games, Life and Utopia*" can also explain the attitude we face the "puzzle problems".

What is Lusory attitude? Lusory comes from the word "ludo", in latin means play, describe the attitude of players required to enter a game (Katie Salen & Eric Zimmerman, 2003). In the puzzle, it is the attitude we confront the complex of puzzle emerges from rules. For example, although we can just rearrange puzzle that simply eliminate it complexity, but people still play it according to game operation rule.

From description above, we already know what is puzzle and attitude people face it. And then, there comes a key problem: what is different between puzzle and other digital game? From Scott Kim's definition, he says game is no answer and puzzle has a right answer. What is it mean? Quote from Chris Crawford (1984), we can give such conclusion: game requires player to build their solution, but puzzle requires player to find out designer's solution. Therefore, we may still feel fun when play other digital-game again and again, but only few times for a puzzle.—a game is fun if there exists uncertainty—because for other digital-game, player doesn't know whether they can complete the level, but for a puzzle, they will remember how to solve it in the few time of play the same level.

So far in this section, I introduce about the meaning of play, it is important concept for designing meaningful game. Next, let's examine about the component in puzzle game.

### Section 1.1.5: Puzzle Game: What is Puzzle? — Component

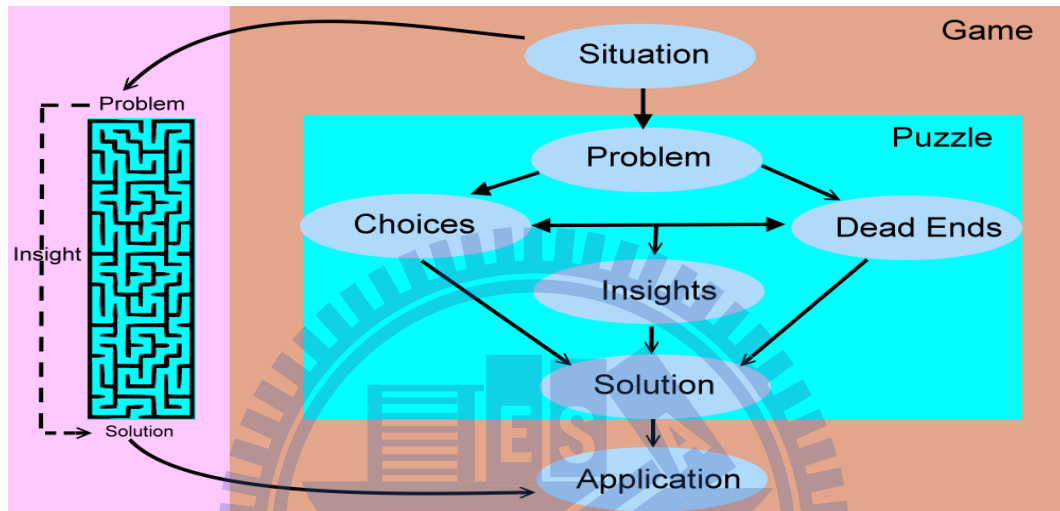


Figure 2 Concept Model of Puzzle developed by Scott Kim.

From Figure 2, Scott Kim (2003) separates game and puzzle as two different concepts, as mentioned in previous section, game and puzzle has different meaning of play. But, by introduce situation component, we can fit puzzles into a game. Situation gives a goal that driven player to solve the puzzle; it explains background by using story or a set of operation rule for player to comprehend it is a “game”. Without situation component, puzzle can't be a game. Situation guides us to handle the problem. The problem, different from everyday problems we encounter, it looks like a maze that has many choices and dead ends reside in it. Choice confuses player to realize which road is correct, and dead ends prevent they from solution. But different from “real maze”, it requires player using insight—the ability to find out which choice is correct and quickly ignore dead ends.—to solve the puzzle. The solution is applied on application in game. The use of Situation and Application component is depending on the genres of puzzle.

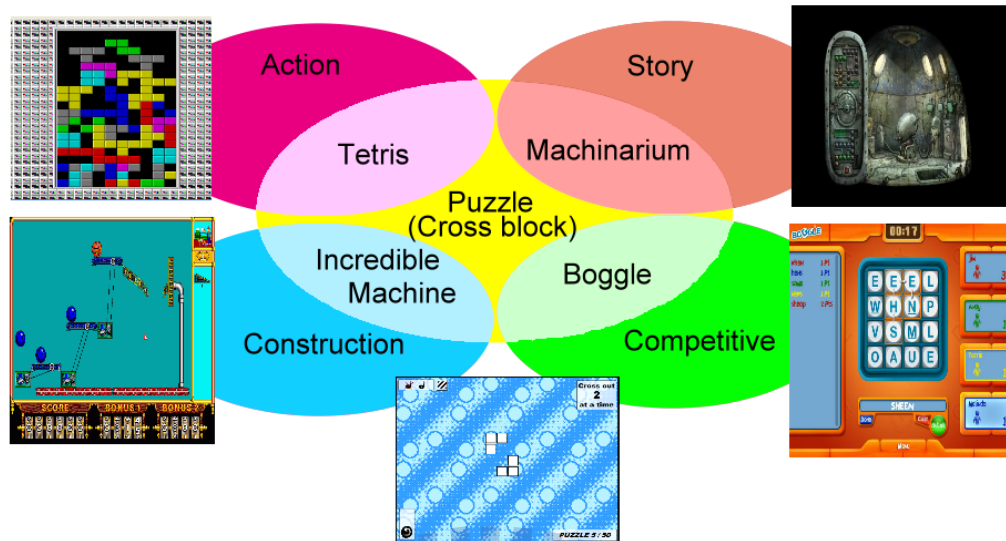


Figure 3 Five genres of puzzle

In Figure 3, we can see Scott Kim (2008) has separated five genres of puzzle: Action, Story, Construction and Competitive and Pure puzzle. Basically, different puzzle will require player different skill, for example:

1. In Action Puzzle, like *Tetris*, it requires player eye-hand coordination skill to handle the game.
2. In Story Puzzle, like *Machinarium* (Amanita Design, 2009) it give a story before each puzzle start in order to create immersion situation for player, but player needs the ability to organize overall story in order to identify which key item are used to solve the problems in game.
3. In Construction Puzzle, like *Incredible Machine*, it requires player Physics knowledge in order to know whether your machine can work properly to solve the problem.
4. In Competitive Puzzle, like *Boggle*, it needs both eye-hand coordination and knowledge of English vocabulary in order to beat out others and win the game.

Remember, the research goal of this thesis is to sort puzzle according to their difficulty. Therefore, this research only focus on pure puzzle, because puzzles that require math logic rather than physical skills or other types of knowledge, like action, therefore it is easier to design difficult measure function compare to other genre of puzzle. You can refer to Appendix to know the rule of “*Cross Block*” and “*Sudoku*”

that used in this research.

### **Section 1.1.6: Puzzle Game: Difficult Measure and Sorting**

In this section, I want to answer the question raised at Section 1.1.3: Is puzzle game can apply DDA for sorting difficult? What is controllable feature? And what is gameplay characteristic? How do we design our questionnaire in order to measure the relationship between obstacles, player action, and player feeling about game?

Of course, it is possible to apply DDA for a puzzle game if we can calculate difficult, but there exists some problem in practical use.

1. For controllable feature, choices and dead ends is obstacle in puzzle but not easy to control it compare to other game. In puzzle, because these two features always emerge from the logic rule of game system, we can't find out a proper number of obstacles easily, it will cost much time to dynamic adjust difficult for puzzle. For example, in *Cross Block*, number of choice and dead ends emerge from the square interaction with other square, therefore, it need much time for auto-generate program to find out proper number of controllable feature for next level.
2. For gameplay characteristic, we can't get accurate data from player. Because meaning of play for a puzzle is to find out a solution, therefore, player tend to know how to solve it if they already solves the problem that same as pervious. Therefore, use time or retry as characteristic for measure performance, will trouble with large variance of collected data for same puzzle that cause analysis difficult.
3. For questionnaire, it is difficult to design question to find out relationship between each obstacle and action, because player feel about the puzzle by their whole emerge pattern but not individual object.

Remember, difficult function is depending on the challenge and skill function, but every player has different skill every time they play puzzle, therefore "optimal arrange" of puzzle will change every time for every player. Let's summary we had discussed so far: difficult is a relative concept that based on player past experience and can't be measure directly; if we want to apply DDA to puzzle game, it will also have some trouble in increase obstacle and measure player performance to dynamic adjust proper difficult for a player.

If such “optimal arrange” is difficult to achieve, why don’t we sort puzzle by some criterion and map it to static difficult (Very Easy, Easy, Normal, Hard and Very Hard)? Because puzzle has the property of emergence that is some kind of complex system, therefore, the goal of this research will focus on how to measure complexity, at the same time, design a method to approximate it to difficult function (static difficult). Compare to DDA, this method is more practical to real puzzle game design process, that we doesn’t need to consider player skill dynamically, and resorting and rescore puzzle’s difficult according to their performance.

My argument here is trying to separate the concept more clearly between Difficulty and Complexity that will more convenience for us to further discuss the topic.

From the aspect of research in task difficulty and task complexity discussion, many papers separate these two terms as different concept: complexity as objective measure and difficulty as subjective. (C. D. Güss, E. Glencross, Ma. T. Tuason, L. Summerlin, & F. D. Richard, 2004; J. Kim, 2005; P. Robinson, 2001)

Similar as our argument above, Jeonghyun Kim (2005) further divided difficulty into two group: first is expected difficulty, which is the percept of difficulty before you start the task; and second is experienced difficulty, which is the feeling after you finish the task.

Next, I want to introduce complexity theory and it relation to puzzle game, which is an essential concept in this research.

### **Section 1.1.7: Puzzle Game: Complexity Theory**

Complexity Theory has two kinds of meaning: one is Computational Complexity, and another is the study of Complex System. In this section, I will briefly introduce these two fields and their relation to puzzle game.

#### **A. Computational Complexity**

Computational Complexity is the study of theoretical computer science and mathematics that focus on how efficiency to handle a problem (M. Sipser, 1997; Sanjeev Arora & Boaz Barak, 2009). For example, there have three famous type of computational efficiency problem NP, NP-Complete and NP-Hard, indicate whether it

can be solved within linear time; furthermore, there also exists the problem about space efficiency: PSPACE, PSPACE-Complete and PSPACE-Hard, indicate whether it can be solved with limited space. Give an overview, there have some research may like: Reduce Time Complexity By an Algorithm for solving a puzzle (R. E. Korf, M. Reid, & S. Edelkamp, 2001), analysis Complexity of Search a Graph (N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, & C. H. Papadimitriou, 1988) and a reduction method for handle games (R. A. Hearn, 2006).

Puzzle is very suitable for further study in this field, because it require player to choose a sequence of action in order to solve it that has many interesting feature for calculate model. Quote from Robert Aubrey Hearn (2006), in his research, Computational Complexity of a puzzle can classify into following category: “If a game is a one-player puzzle with a bounded length, odds are it is NP-Complete.” and “Indeed, unbounded puzzles are often PSPACE-Complete.”

Bounded and unbounded puzzle means whether it has a restrict length to solve it. In unbounded puzzle we can always go back to previous state, therefore it has no restrict length. Both of them need exponential time to compute a solution, but they are different in whether we can use polynomial space to verify a specific action sequence is correct. Because Savitch's (1970) theorem had proofed that NP-SPACE = PSPACE, therefore we can solve any puzzle problem with polynomial space. The main research direction in this filed is how to solve a puzzle more computational and space efficiently. Is computational effort relate to complexity of puzzle and can use for sorting purpose? I think it is not a good idea, because Computational Time and Space problem, your machine will run a long time or crash due to memory lacking when compute a very complex puzzle.

## **B. Complex System and Emergence**

What is complex system? Although this filed has been studied in modern computer science for a long time, but it is one of profound problem that people tends to understand in past several thousand years. Aristotle (384 BC – 322 BC), a noted Greek philosopher, who first organized the concept in his questions about *Metaphysica*: “The whole is more than the sum of its parts.”, that actually indicate the most important property of complex system.

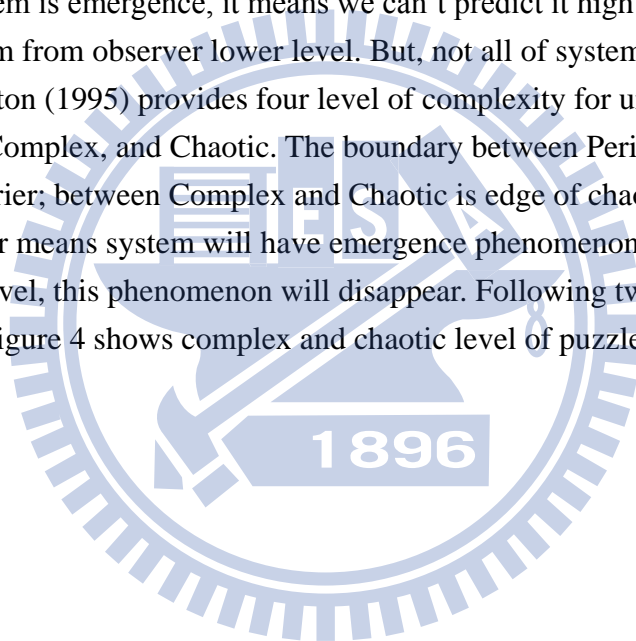
Jeremy Campbell (1982) looks this “whole phenomenon” from the aspect of information, language, and DNA, says that when system beyond a “complex barrier”,



entirely new principle will come into play. The principle, call emergence, may allow a system to self-organizing, replicating, learning, or adaptive itself to environment.

Penny Sweetser (2007) has summarized some common property for a complex system: Elements, Interactions, Formation, Diversity, Environment, and Activities. In other word, if there exist a set of elements, that will inter-interact with a set of rule in an environment for specific purpose, their interaction process has large state space, element will reorganize itself over time changed, and then it is a complex system. The first deep exploration about emergence is from John Holland's (1999) book "Emergence: From Chaos to Order", shows many example about how emergence arise from complexity.

When a system is emergence, it means we can't predict it high level behavior or structure of system from observer lower level. But, not all of system is complex. Christopher Langton (1995) provides four level of complexity for understand system: Fixed, Periodic, Complex, and Chaotic. The boundary between Periodic and Complex is complexity barrier; between Complex and Chaotic is edge of chaos. Beyond complexity barrier means system will have emergence phenomenon, but if it complex reaches chaotic level, this phenomenon will disappear. Following two "Cross Block" puzzle levels in Figure 4 shows complex and chaotic level of puzzle:



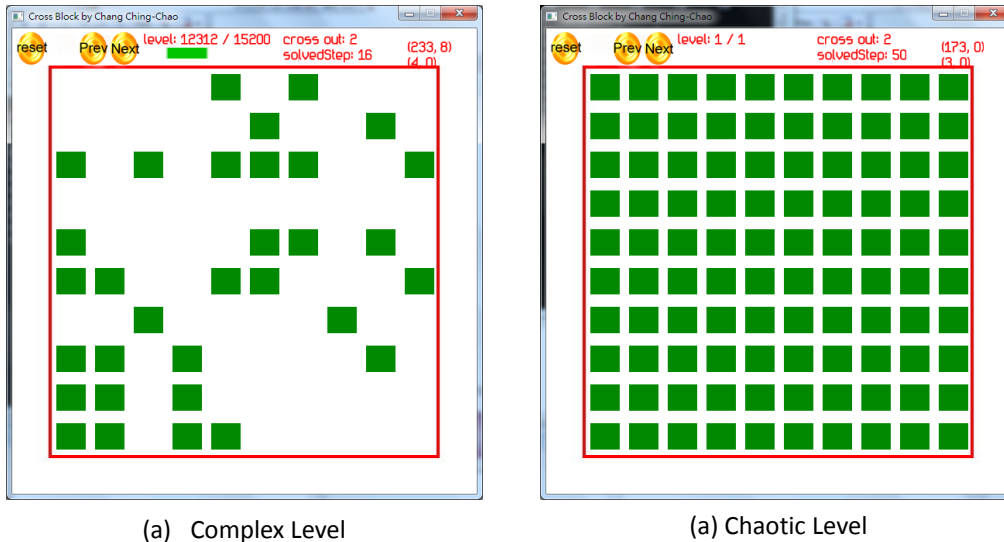


Figure 4 Emergence Phenomenon Example

(a) Complex level of puzzle that has high complexity. (b) Chaotic level of puzzle that with no complexity (no dead ends) that every square can interaction with each other to form a basic element that can be canceled by player.

It shows emergence phenomenon in the “*Cross Block*” puzzle, that both choice and dead ends will increase when it beyond complexity barrier and dead ends will decrease when reach chaotic level.

The study has widespread research in many fields, such as Information Complexity on Communication System (C. E. Shannon, 1948), Artificial Life (Adamatzky. Andrew, 2010; Christopher G. Langton, 1995), Biological System (Gerald M. Edelman & Joseph A. Gally, 2001), Economic System and Human Society (Holling, 2001)...etc. We can’t survey all of those fields here for understand what is complexity, since it will diverse our discussion to focus on puzzle game. With a general idea, quote from Penny Sweetser (2007), we simply define complexity as following meaning : “*Complexity is a measure of the difficulty involved in understanding a system.*”

What means to understanding the puzzle? If someone can solve a puzzle level, we say he/she understand it. How do we measure complexity of a puzzle? From previous discuss, we know insight is important skill to solve a puzzle, and there has two components will affect it: choice and dead ends. But, because they are emergence phenomenon in the puzzle, therefore we can’t directly control it. How do we calculate it? From computational complex theory we discuss, it will fail when we want to expand search space in a puzzle. In chapter 4.1, I will introduce our approximate method.

## **Section 1.2: Motivation: Challenge in Puzzle Game Sorting**

Basically, we can classify difficult model into two classes: dynamic difficult and static difficult. There has several challenge of measure dynamic difficult in puzzle game: First, because puzzle game have emergence property, therefore it is difficult to control obstacle. Second, we must design a method to distinguish those puzzles which players already know their answer in order to measure player's skill correctly. Third, it is difficult to find out relation between each object, because the difficulty of puzzle is "whole" not individual obstacle. Therefore, in this research, we will only focus on how to measure complexity and map it into static difficult.

## **Section 1.3: Motivation: Mobile Game, Market and Puzzle Game**

Recently, mobile game market has dramatically growth, especially when Apple releases their cutting-edge product: iPhone and iPod, there has more and more company starting their game project on mobile platform. According to Apple's official news, the number of App Store—an online software download service for Apple's product (iPhone, iPod, iPod Touch), which launch on July 10, 2008 (Apple).—downloads already exceeds 10 billion, furthermore, it is worth noting that TOP 10 of popular iPhone paid Application, 9 is games (Robin Wauters, 2011). Hence, the market in the mobile game has large amount potential benefit. There has a research shows that Mobile app market will be worth \$25 billion U.S. dollar By 2015, compare to 2010 is \$ 6.8 billion (Sarah Perez, 2011).

My research is focus on puzzle game, which is very suitable for mobile platform, because it has short play session and player can stop it at any time without punishment compare to other hard core game. In fact, "*The games that are popular on the mobile platform are mostly casual games*"(Elina M.I. Koivisto, 2006)

Puzzle is a kind of casual game, which is popular in the mobile game. Just as introduce on background, there have five different genres. But, we only focus on pure puzzle, which doesn't have any other additional element, since it convenience for our research on calculate and sorting complexity.

Barry Clarke (1994) in his book, "Puzzles for Pleasure" collect large number of puzzle and classify into two category according to their difficulty: Popular Puzzle and

Advanced Puzzle. Popular Puzzle requires modest insight and engagement that suit for every people; Advanced Puzzle for those puzzle-solving manias, who think Popular Puzzle too easy. In past time, designer tends to use their own sense to rank the puzzle they design. If we can tell which puzzle is Popular, which is Advanced and tell the degree of its complexity that relative to others, it will be very helpful for puzzle game design process.

## **Section 1.4: Goal**

The purpose of this research is to design a general method that can sort pure puzzle according to their complexity. In order to grasp more accurate purpose of this research, here comes the summary about Difficulty and Complexity that described in background.

Difficulty is a subjective and relative concept that based on player past experience. Both challenge and player skill will affect it, but because measure player skill in puzzle is difficult, therefore, we use complexity instead of difficulty for sorting purpose.

Complexity is an objective concept that is the measure of difficulty involved in understanding a system. In puzzle game, how difficult for a player to understand a puzzle depends on their insight to a puzzle. As introduced in previous section, insight will be affected by choice and dead ends, therefore, this research only uses these two criterions for complexity measure, furthermore, design a method that can approximate complexity to human sorting (difficulty).

## **Section 1.5: Contribution**

Puzzle is a popular game type in mobile platform, which have short play session time that is very suitable for time killing. As describe in motivation section, there has more and more company starting their game project on mobile platform. But, there exists a trouble for design a puzzle game: How do we decide arrange of puzzle? In the past research, they are focus on how to solve a puzzle more efficiently in term of computational and space complexity. However, it is not necessary for real game application. Although all puzzle need to validation a solution, but if it cost too much time, then it is not practical. The focus of this research is on complexity sorting, which takes practical into account, is more essential for puzzle game designer.

Because almost all puzzle game can auto-generate by program, we can simply generate large number of levels, therefore how to pick out a set of appropriate complexity levels is very important topic.

Our result will be a calculate model, which can calculate complexity for any pure puzzle. If you have a solving program and a solving sequence, then our model can tell the score and rank base on all puzzles in the puzzle database.—It is very convenience for puzzle game designer to analysis what is difficult in puzzle.



## Chapter 2: Literature Review

This chapter mainly focuses on some basic technique, theories and method that are related to our experiment method. You can skip this part if you already familiar with it.

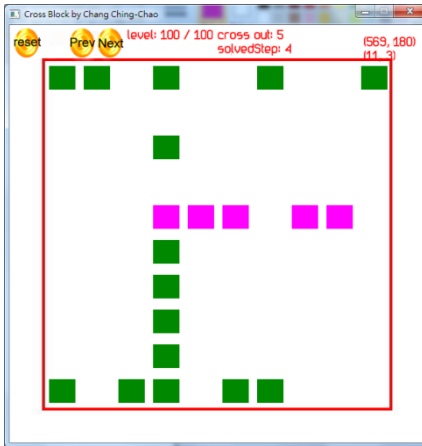
### Section 2.1: Tree Search

Formulate game as tree search problem is very popular technique in the field of Game AI. Programmer always apply this method to create “intelligent” in game, for example, in two-player game like go, chess or Othello, we create intelligent opponent to compete with human player; in RTS, like AOE, StarCraft, we let game agent find out an optimal path from “A” point to “B” point to reduce effort of player control; in puzzle game, like Sokoban, Sudoku, try to find out and validate a solution sequence to give the hint for player.

Although I describe some application of tree search above, but I still doesn't explain what it is. What is tree search in term of programming? It is a problem solving technique by discrete and expanding possible state of problem in order to find out a solution. “Problem” and “Solution” are two essential concepts in this method. Russel (2002) in his book list four element to define what is Problem:

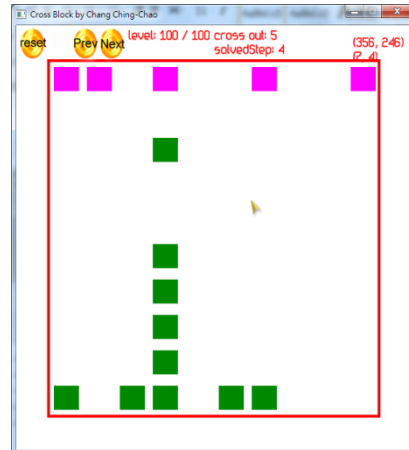
1. Initial State: like concept model of puzzle I mentioned in chapter 1, it is the entry of maze.
2. Successor Function: a set of action-state pair, record which action can lead to which state. In tree search problem, it is branch or choice.
3. Goal Test: test whether our goal is achieve. It can be explicit, if current state is on certain state we already list in goal list; or it can be implicit, if certain condition of current state is achieve.
4. Path Cost: the cost from initial state to current state, it can be simply define as time, distance or number of action executed, depend on your application.

The Solution is a set of action sequence that can lead problem from initial state to goal state. Figure 5 is an example of Solution in Cross Block Puzzle:



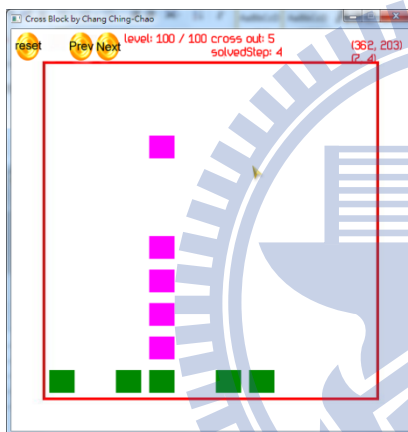
**Initial State, Path Cost = 0**

Action 2: (4, 5) (9, 5)



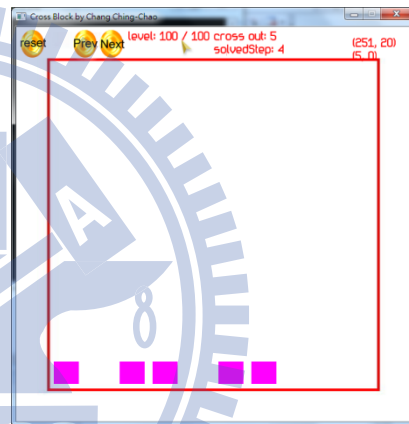
**Path Cost = 1**

Action 2: (1, 1) (1, 10)



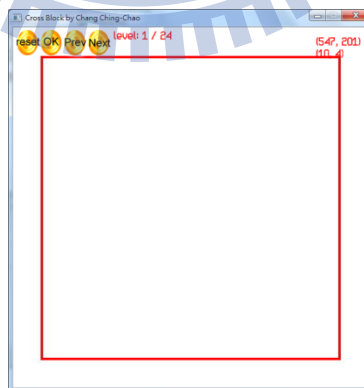
**Path Cost = 2**

Action 3: (4, 3) (4, 9)



**Path Cost = 3**

Action 4: (1, 10) (7, 10)



**Goal State, Path Cost = 4**

Figure 5 Example of a puzzle Solution.

The core idea of tree search is to explore over all state space of problem in order to found out a solution. Like it name, when you explored the state, you will find it similar to the branch of tree, see Figure 6.

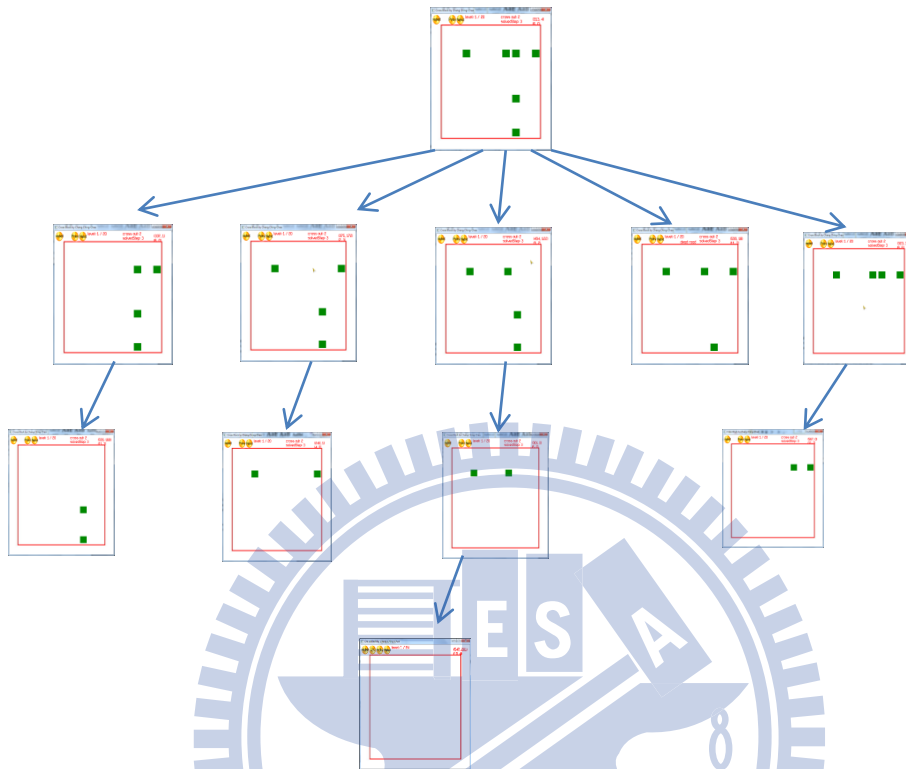


Figure 6 Example of tree search

How do we design such algorithm? Because “state” only store representational dimension of problem, therefore we need other data structure to record other information, such as path cost  $f$ , depth of tree search, current state come from which state(parent), which state current state can go (child), almost all literature call this kind of data structure as “node”. You can simply use adaptive pattern or wrapper pattern, from design pattern(Erich Gamma, Richard Helm , Ralph Johnson, & John M. Vlissides, 1994), to including such information for state, the C++ code like Code 1:



```
[C++ code]

class Node{
public:
    Node(State* _state){state = _state;}
    State* state;
    float g;//real path cost
    float h;//heuristic, guess cost, used in informed search
    float f;// path cost, f = g + h
    int depth; // depth in search tree
    Node* parent; //come from which state
    map<Action*, State*> child; // which state that current state can go
};

[C++ code]
```

Code 1 Data Structure Node.

Have node as basic data structure, our tree search implementation looks like Code 2 and Code 3: algorithm from Russell's (2002) book, chapter 3.

```
[C++ code]

vector<Action*> treeSearch(State initialState){
    vector<Node*> fringe;
    fringe.push_back(new Node(initialState->clone()));

    while(true){
        if(fringe.empty()){
            vector<Action*> noSolution;
            return noSolution;//can't find solution, return null list
        }
        //FIFO, BFS
        Node* node = fringe.front();
        fringe.erase(fringe.begin());

        if(isGoal(node)){
            //trace back from goal node to initial node, we can make the solution
            return makeSolution(node);
        }
        vector<Node*> leaf = expand(node);
        insertAll(fringe, leaf);//insert all node in leaf to fringe
    }
}

[C++ code]
```

Code 2 Implementation for BFS tree search algorithm.

```

[ C++ code ]

vector<Node*> expand(Node* node){
    vector<Node*> leaf;
    node->child = getSuccessors(node);
    map<Action*, State*>::iterator it;
    for(it = node->child.begin(); it != node->child.end(); it++){
        Action* a = it->first;
        State* newState = it->second;
        Node* newNode = new Node(newState);
        newNode->parent = node;
        newNode->g = node->g + stepCost(node, action, newNode);
        newNode->h = heuristic(newNode); //set heuristic as 0
        newNode->f = newNode->g + newNode->h;
        newNode->depth = node->depth + 1;
        leaf.push_back(newNode);
    }
    return leaf;
}

[ C++ code ]

```

Code 3 Implement for expand function.

In this algorithm, only thing you must do is to design your State class and successor function because it various depends on application detail. But still, it have some problem if there have some action that can go back to same state that previous had expanded, then program will fail to explore over all state space because same stae will be expanded again and again. In order to solve such problem, we must introduce a list that can record that already be expanded. The algorithm is called graph search in Russell's book. Code 4 is my implementation:

```

[C++ code]
vector<Action*> graphSearch(State initialState){
    vector<Node*> fringe, vector<State*> closed;
    fringe.push_back(new Node(initialState->clone()));
    while(true){
        if(fringe.empty()){
            vector<Action*> noSolution;
            return noSolution;//can't find solution, return null list
        }
        Node* node = fringe.front();
        fringe.erase(fringe.begin());
        if(isGoal(node)){
            //trace back from goal node to initial node, we can make the solution
            return makeSolution(node);
        }
        if(!isCircle(node->state, closed)){//check if node is already in closed
            closed.push_back(node);
            vector<Node*> leaf = expand(node);
            insertAll(fringe, leaf);//insert all node in leaf to fringe
        }else{
            delete node; node = NULL;
        }
    }
    release(closed); //release memory
}
[C++ code]

```

Code 4 implement for graph search.

There have three main variations for search algorithm: Depth-First Search (DFS), Breadth-First Search (BFS) and Uniform-Cost Search (UCS), all of them are different at which node is expanded first. BFS expands the node from beginning of fringe; DFS expands from back; and UCS expands from lowest cost.

Also, they have different benefit in solving the problem. BFS and UCS can find optimal solution, but because all nodes must keep in memory, therefore space will be a big problem; else, although DFS doesn't have memory problem, but it can't find optimal solution and not suitable for those problems which total depth too high or unlimited.

Therefore, there comes the method to improve the problem describe above, like Iterative Deepening Search. By gradually deeper search depth, our search tree can improve space problem cause in BFS. Another improving technique like A\*, is using heuristic as cost measure, can reduce large amount of node doesn't need to be

expanded that can increase searching performance.

But the algorithm in this section only suit for one player game, for those two-player game problems like go or chess, we need apply min-max or alpha-beta. We don't introduce two-player game tree search, because it is not relate to this research, but it core idea is same in this section.

## **Section 2.2: Local Search**

Although tree search is a powerful problem solving method, but there still exists some weak point, for example, if we want to solve a problem that with very large state space, then it will always cost too much time to find a solution or crashed because run out of memory. It is not very efficiency for those problems, which only wants to find an acceptable goal state but not their solution path, such as 8-queen problem, therefore, here comes another algorithm in computer science, call Local Search.

What is Local Search? It is an optimization technique by only consider current state and gradually move to their better neighbor state and finally find an acceptable goal state. The term optimization in this method doesn't mean it will always find a global optimal, but because we can always find an approximated optimize state, call local minimum/maximum or local optimal.

Before starting search, we must design an objective function to measure the goodness of current state. How do we define what is "better state" will affect we try to find is local minimum or local maximum. If we feed training sample into objective function to tell program what is good and what is wrong, then it is a kind of machine learning. Like genetic algorithm, neural network...etc., all of them are local search.

Figure 7 shows the concept of local minimum search describe above that adapted from (Russell. S. & P. Norvig., 2002) chapter 4:

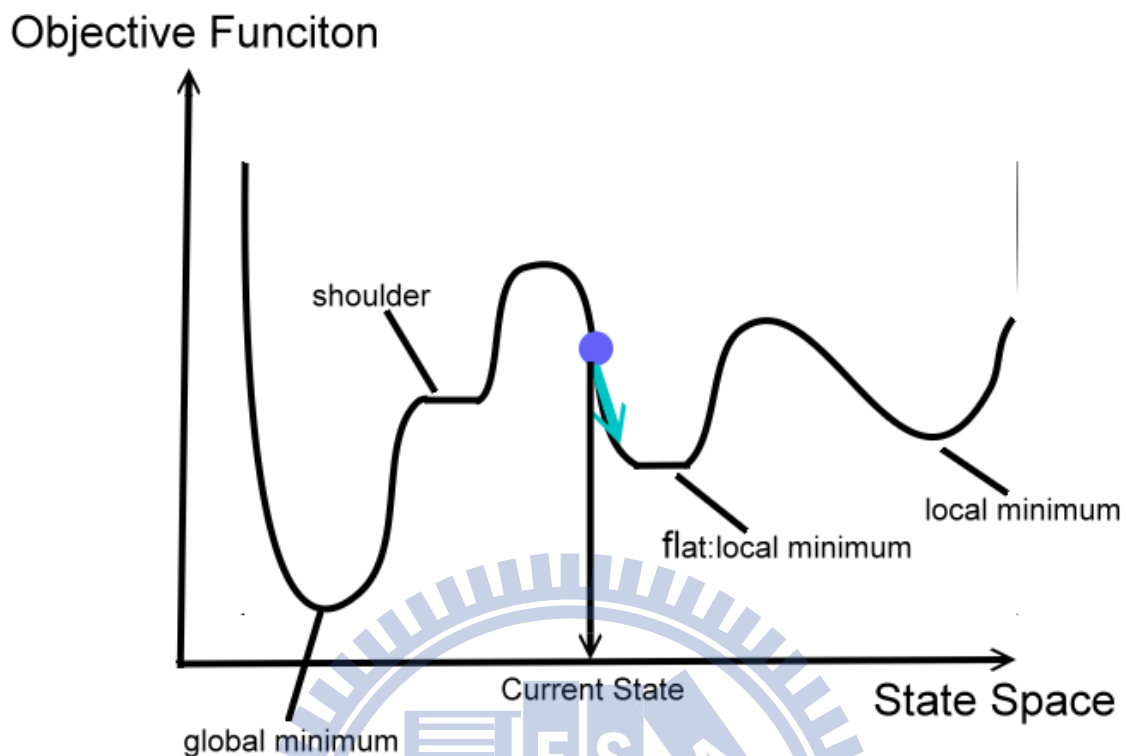


Figure 7 Concept of minimum local search

Of course, it is the best result if we can find global minimum, however, local search algorithm always stuck on following three places: local minimum, flat and shoulder. There doesn't any solution to remove this problem, but instead, we have a principle to get a better result: "If at first you don't succeed, try, try again." (Russell. S. & P. Norvig., 2002), by randomly initialize state, you will get a chance to approach best result over state space.

## Section 2.3: Simulated Annealing

Because classical local search algorithm tends to stuck on local optimal, therefore if we can jump out local then it seem easier to find a better solution. Simulated Annealing that are is such kinds of algorithm. By introduce some probability to do random walk over state space, and then it can help us to jump out local minimum / maximum state. The concept "annealing" come from physical says that it *"is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to coalesce into a low-energy crystalline state."* (Russell. S. & P. Norvig., 2002)

Figure 8 is the flow chart of Simulated Annealing:

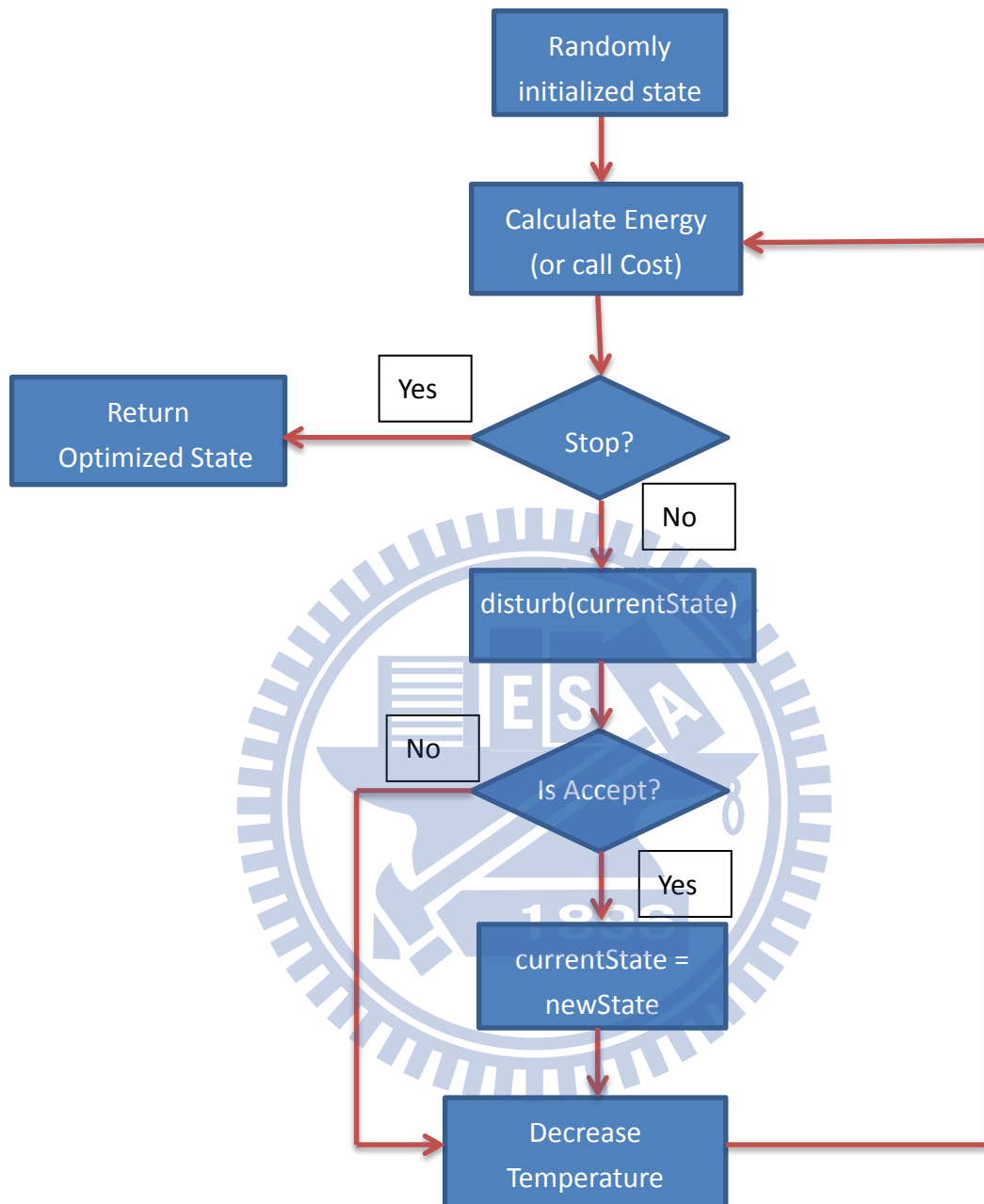


Figure 8 Flow Chat of Simulated Annealing.

First step is randomly generating our state, it is because we want to increase opportunity to find global optimal, and then we calculate energy of current state, because this algorithm is local minimum search, therefore we can also call this energy function as cost function. You must design this function depends on your application. Next, try to test if current state is good enough, if so, and then it is an optimized state and return it; else, try to adjust current state and test if we can accept this new state in current temperature.

Temperature is a core concept in this method, it will affect whether we can accept new state. It will accept new state by following rule, call Metropolis criterion(Kai-Ju Chen & Kou-Yuan Huang, 2007):

1. First, Set  $\Delta E = \text{new state energy} - \text{old state energy}$ .
2. If  $\Delta E \geq 0$ , then accept it immediately.
3. Else, using current temperature  $T_k$  to compute probability  $p_t$  in Boltzman distribution and randomly generate a random probability  $r$ .
4. If  $r \leq p_t$ , then accept it, else reject.

Boltzman function is a function to simulate the probability of transforming physical state in certain temperature. It is defined by Code 5:

```
[C++ code]
float SimulatedAnnealing::Boltzman(float deltaE){
    float e = 2.71828182845904523536f;
    return min(1.0f, pow(e, -deltaE / T k));
}
[C++ code]
```

Code 5 Boltzman distribution for simulated annealing.

When temperature  $T_k$  is high, then we will tend to change our state in spite of it is a bad state compare to old state. Until temperature continue decrease down to certain number, and then state will keep to a stable and find a local optimal. You can try same temperature many times. Code 6 is my implement for temperature decrease function:

```
[C++ code]
// temperature decrease function
float SimulatedAnnealing::schedule(float t){
    float alpha = 0.95;//decrease speed
    float T max = 600;//initial temperature
    return T max * pow(alpha,t-1);
}
[C++ code]
```

Code 6 Temperature decrease function for simulated annealing.

And is Code 7 is my implement for accept function:

```
[C++ code]

SimulatedAnnealing::isAccept(State* oldState, State* newState){
    float oldEnergy = energy(oldState);
    float newEnergy = energy(newState);
    float deltaE = newEnergy - oldEnergy;

    // Metropolis criterion
    if(deltaE <= 0){//if new state good then old state then accept it immediately
        return true;
    }else{
        float r = random.uniform_float(0, 1);
        float pt = Boltzman(deltaE);
        if(r <= pt){
            return true;
        }
    }
    return false;
}

[C++ code]
```

Code 7 Accept function for simulated annealing.

The program used in this research is adapted from (Kai-Ju Chen & Kou-Yuan Huang, 2007; Kou-Yuan Huang & Ying-Liang Chou, 2008), by design our state as mathematical form, and gradually adjust its parameter, then we can get a set of optimal parameter.



## Section 2.4: Pseudo-Random and Real-Random

What is Pseudo-Random? In computer, we can't really generate Real-Random number because it is run by deterministic process. If you feed same random seed (used to calculate) for random program to generate random number, you will find your program generates same random sequence as previous run and this number sequence will repeat again and again as a period length.

Because we can predict number generated by program if its algorithm is known, therefore, we call computer-generated random number as Pseudo-Random. In computer science, there has much research introduce many algorithms about how to approximate real-random.

Mersenne Twister (MT) is a most popular Pseudo-Random method nowadays that developed by Makoto Matsumoto and Takuji Nishimura (1998). Its name is come the fact that period length in algorithm will be a Mersenne prime ( $M_p = 2^p - 1$ ). In this research, we adapt a MT variation call MT19937 which has long period ( $2^{19937} - 1$ ) and can generate 32-bit integer, for our random process.

## Section 2.5: Game, Digital Game and Media

What is game? Beginning works may be trace back to Johan Huizinga in 1954. He was analysis what is game and its meaning from the aspect of philosophy. According to his works, *Homo Ludens* (Johan Huizinga, 1954), says that game will be a game if it satisfy following three feature:

1. Voluntary: Participator must with his/her own will to join the game.
2. Unreality: The content of game must achieve some fantasy content.
3. Separation and Regional limitation: game exist a boundary between reality and fantastic, call magic circle in *Rules of Play*(Katie Salen & Eric Zimmerman, 2003).

What is digital game? It is the game that integrate with many different media, like word, picture, music...etc. We can separate all digital-game and non-digital game as Figure 9:

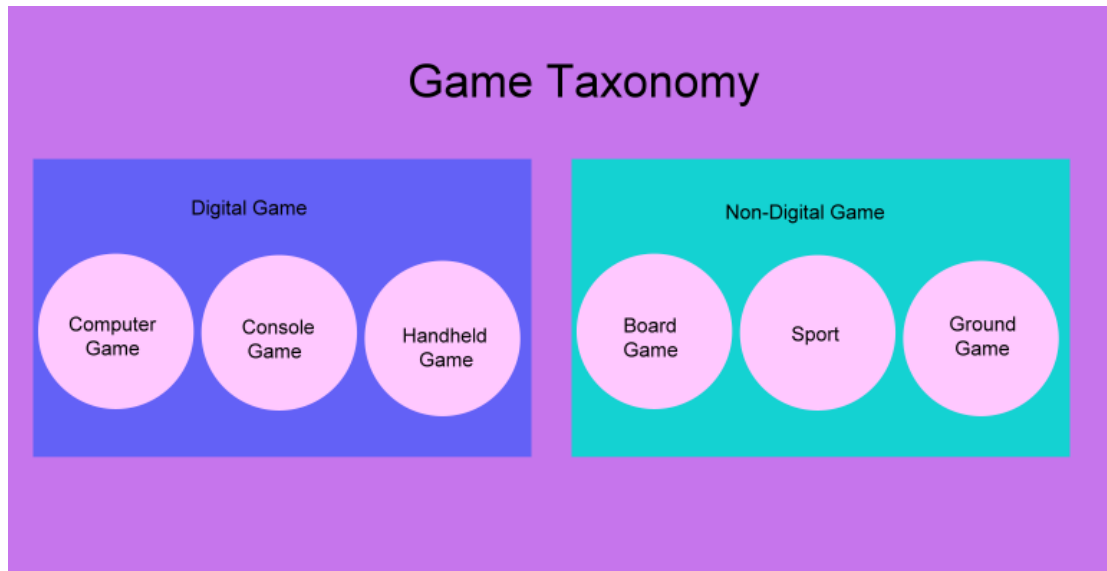


Figure 9 Game Taxonomy by Media.

### Digital Game:

- Computer Game: use computer as game media.
- Console Game: use TV as game media like X-BOX, PS3.
- Handheld Game: use small device as game media like iPhone, PSP.

### Non-Digital Game:

- Board Game: mainly use physical tool pencil, paper, or card ...etc., as play media like Monopoly, Carcassonne, usually as indoor activity.
- Sport: use player's own physical body to compete power for each other.
- Ground Game: Opposite to board game, it is an outdoor activity. Game like Hide and seek, hopscotch, and geocaching may be classified into this category.

But Taxonomy for each game are not fixed, for example, Wii-sport is successful in combining Console Game and Sport as new play style.

## Section 2.6: Flow Theory

In psychology, flow means optimal experience when challenge meets skill. The term are propose by Czikszentmihalyi (1998). Figure 10 is mental state refer in flow theory:

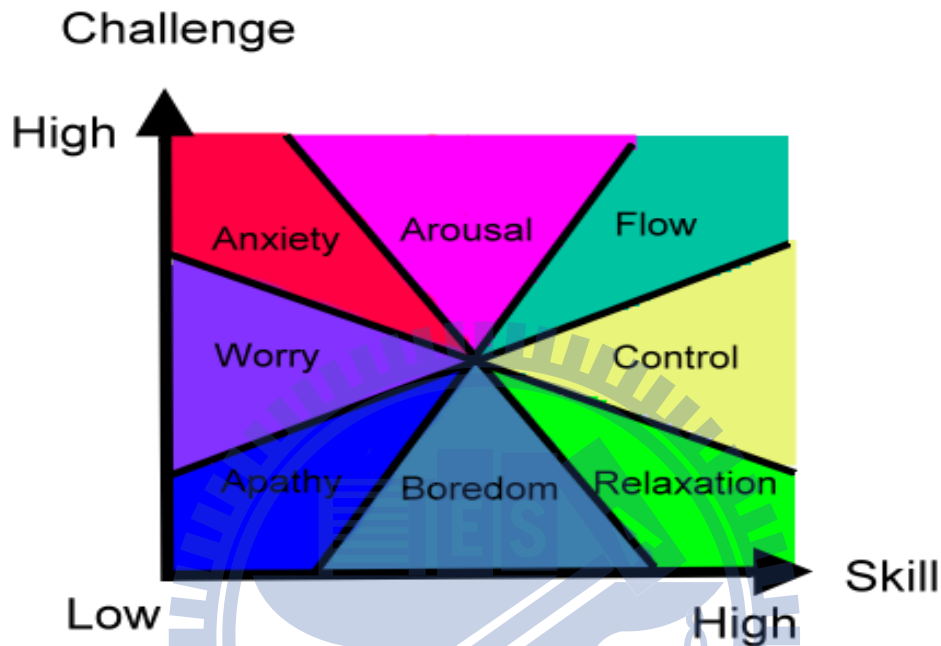


Figure 10 Mental State in flow theory.

It worthy to note that flow condition only occur in high challenge and high skill, where low challenge and low skill are considered as Apathy, means player doesn't care about whether they can get good performance in the game.

## Section 2.7: Three-part rule model

The model are proposed by Katie Salen and Eric Zimmerman (2003). They divide game rule into three parts:

1. Operational rules: structure of a game, how can we operate the games. We must first know the legal input for a game, and then can start gameplay. Operational rules have some property as following:
  - a. It must be an unambiguous and explicit, for example, write down on the manual.
  - b. It must share among all players that everybody can access to it without any information loss.
  - c. It must be fixed and repeatable, so it can helps us to identify and confirm

every game instance we play are actually same.

- d. It must make binding among player that if they break the rule they may pay some penalty that will reduce their fun experience, therefore player may more likely to play the game according to the rules. Although there have some situation that will make player to do some cheat, but the problem doesn't relate to this research, so we don't discuss cheat problem here. You can refer to Mia Consalvo (2007) works about cheat in games to get more detail idea.
2. Constitutive rules: It is logic part of games. How to explain game outputs and select a set of legal inputs is essential part of gameness. When a better explanation can be made, then better you will play the game. Player is required to learn how to "insight" this rules in order to win. If we want to design or analysis gameness for a game, Constitutive rule is most important part we must care, because it will emerge large amount of play strategy. For example, in "Cross Block", Constitutive rule is number of "Cross Out" and wining condition.
3. Implicit rules: like the social norm, it doesn't explicitly write down on the game manual, but everybody will obey the rule voluntarily. For example, when play the chess or Go, it will break implicit rules when one player hide game board from his/her opponent. This rule will always change depends on environment when you play the game. There may have some implicit rule become operation rule in different environment.

By the description above, we conclude that Constitutive rules are the source which brings the feeling of difficulty to players.

# Chapter 3: Method

This research separate into following four experiments:

- **Experiment One: Experiment on Puzzle Game Space for Complexity Measure using “Cross Block”.**
- **Experiment Two: Validate results between Complexity and Difficulty Using “Cross Block”.**
- **Experiment Three: Validate results between Complexity and Difficulty using “Sudoku”.**
- **Experiment Four: Approximate Complexity to Difficulty, using “Sudoku”**

The first two is preliminary experiment, which want to validate some property of puzzle game. And another two is our main experiment, which validate the correct rate of puzzle difficult sorting.

## Section 3.1: Experiment One

As describe in chapter 1, **Difficulty** and **Complexity** in this research is two different concepts. Complexity is objective according to puzzle itself and Difficulty is subjective according to player past experience.

Generally speaking, the more solved step a puzzle required the more difficulty and complexity a puzzle may be. But, is this assumption true? According to the concept model of puzzle game developed by Scott Kim, introduced in chapter 1, we known there have two attribute of puzzle will affect insight: choice, also call branch in this research, and dead ends. I use these two criterions to measure complexity of puzzle.

In order to validate the result, first, I want to introduce how to calculate complexity. Because this research uses “Cross Block” that game board size equal to  $10 * 10$  as experiment puzzle (refer to chapter 1.1.6), therefore, here coming two Property need to validate:

- **Complexity Property 1:** When solved step increase, puzzle’s complexity should increase.
- **Complexity Property 2:** When cross\_out is about half of game board, then it complexity should be highest. In this experiment is cross\_out\_5 ( $10 / 2 = 5$ ).

If result corresponds to both Property, then we can say complexity calculate model in this research is successful. By generate large number of puzzle levels and do statistic to observe whether overall Puzzle Game Space is corresponding property. What is Puzzle Game Space? It indicates every possible state in a puzzle game.

Here have 5 phases in this experiment:

- **Phase 1:** Random generate large enough samples for each solved step. Like Figure 11, only sample puzzle levels in game board Size =  $10 * 10$ .

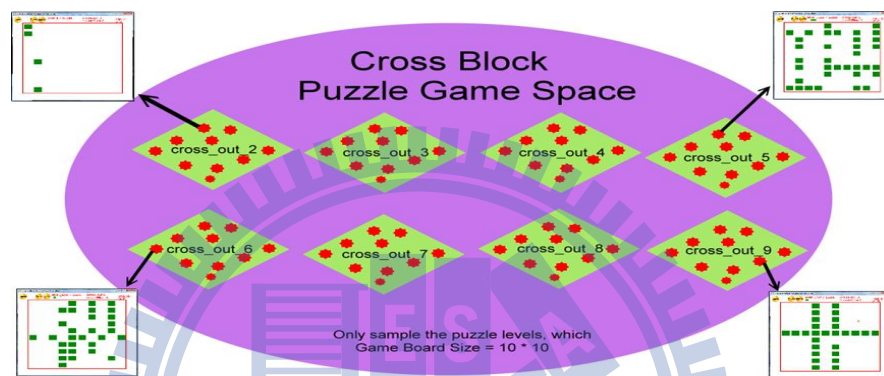


Figure 11 *Cross Block's* Puzzle Game Space.

- **Phase 2:** Calculate branch and dead ends for each puzzle levels. Generally speaking, expand all node will get more accurate result. But I don't do that. Why? Because there has many puzzles is NP-Complete problem. It takes too much time, and either impossible to calculate for some complex puzzle. If we only expand answer node (from random generate process, we know it), we can reduce problem to linear time. Like Figure 12.

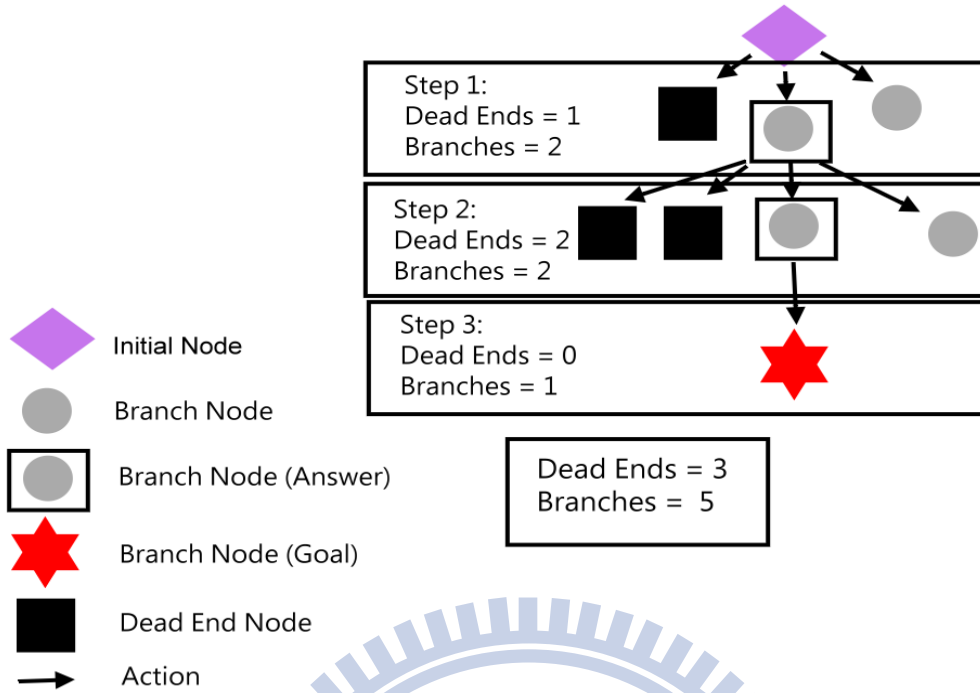
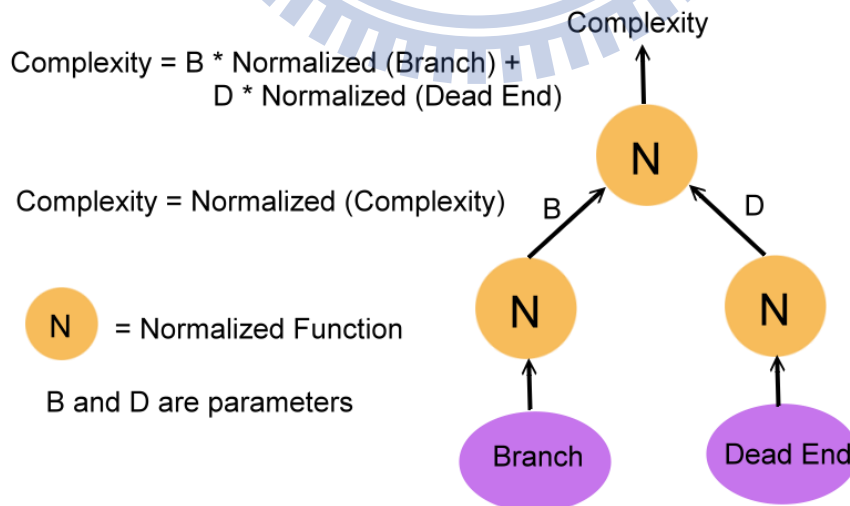


Figure 12 Branch and Dead End Calculating Process

- **Phase 3:** Calculate branch and dead end's sample mean for each solved step.
- **Phase 4:** Calculate complexity for each puzzle levels, you can see the method I propose in Figure 13. Normalize function in this model can help us explain result of complexity and doing parameter adjust.



• Figure 13 BD-Complexity Calculating Model

I will show the result in Section 4.1: for validate Complexity Property 1 & 2.

But, there still need furthermore validate process about the difference between Complexity and Difficulty. Therefore, in next experiment, we use “*Sudoku*” puzzle to validate the result and shows the ability of our method can handle different puzzle.





## Section 3.2: Experiment Two

The purpose of this experiment is to validate correct rate of complexity sorting by compare result in experiment one to human difficulty evaluation of puzzle. Here has 5 phases in this experiment.

- **Phase 1:** Select the puzzle levels from Puzzle Database with proper complexity distribution for human evaluation.

In order to validate the result of complexity, first, we must choose a set of puzzle that have proper complexity distribute over puzzle game space. In experiment one, we will generate a set of puzzle and calculate their complexity value with both parameter B and  $D = 1$ . Because complexity value will be normalized, therefore, we can simply separate into five basic difficulty groups as following:

- **0 ~ 0.125 (Very Easy)**
- **0.125~0.25 (Easy)**
- **0.25 ~ 0.5 (Normal)**
- **0.5 ~ 0.75 (Hard)**
- **0.75 ~ 1.0(Very Hard)**

Although Boundary between each basic difficulty doesn't be validated, however it is convenience enough for us to choose the puzzle. By random select the puzzle levels from those four groups, we can get a set of puzzle with proper difficulty distribution.

- **Phase 2:** Evaluate difficulty by real human player.

Collect data from player, with following process:

1. When player completing (even give up) one puzzle level, let them give a difficulty score between 0 ~ 100.
2. When player finishing all puzzle levels (even there exist some give up levels), let them rescore all puzzle difficulty again.

First score data wants to see whether player will affect their evaluation about difficulty when complete more and more puzzle.

Second score data wants to compare ranking result that the experiment one generated. By average all collect data, we compute arrangement familiar ratio that can tell how successful the experiment one is.

- **Phase 3:** Average difficulty that evaluated by real human and sorting the result.
- **Phase 4:** Calculate sorting similarity between Difficulty and Complexity with **small puzzle base**.
- **Phase 5:** Calculate sorting similarity between Difficulty and Complexity with **large puzzle base**.

The different between Phase 4 and Phase 5 is number of puzzle in puzzle database. Small puzzle base means we only use experiment puzzle set that are picked in phase 1 to compute complexity; large puzzle base means we will consider all puzzle over puzzle space that are generated in experiment one to compute complexity for each puzzle. Generally speaking, large puzzle base has more accurate complexity value and sorting.

### **Section 3.3: Experiment Three**

We use the Sudoku puzzle that provided by Taiwan Sudoku Association (TSA) (W. Kuang-Chen (巫光楨), 2008) to validate our result. In the website, they statistic solve rate, time used to solved for each “*Sudoku*” puzzles and separate it to 5 ranks that can corresponding to basic difficulty: very easy, easy, normal, hard, and very hard.

- **Phase 1:** select proper amount of puzzle in each rank.
- **Phase 2:** find a solution sequence for each puzzle, we use solving program in TSA (W. Kuang-Chen (巫光楨), 2008).
- **Phase 3:** Using solution sequence to calculate branch and dead ends for each puzzle.
- **Phase 4:** compute complexity for each puzzle.
- **Phase 5:** compare rank result by website and complexity rank by calculate sorting similarity.

## Section 3.4: Experiment Four

In this experiment, I will use simulated annealing to tweak parameter B and D in Figure 4.3. By feeding training data collected in experiment three, we can improve our complexity sorting result

- **Phase 1:** Random select training sample from TSA's "Sudoku" puzzle.
- **Phase 2:** Use simulated annealing to tweak parameter B and D.
- **Phase 3:** Calculate new complexity.
- **Phase 4:** Compare rank result.



# Chapter 4: Experiment

## Section 4.1: Experiment One

In this section, I want to validate following two complexity property by generate large enough puzzle sample over Puzzle Game Space:

- **Complexity Property 1:** When solved step increase, puzzle's complexity should increase.
- **Complexity Property 2:** When cross\_out is about half of game board, then it complexity should be highest. In this experiment is cross\_out\_5 ( $10 / 2 = 5$ ).

Chapter 3.1 had already introduced experiment phase for experiments, in this section, I will go into detail about how to implement it in this research and shows the result.

### Section 4.1.1: Phase 1: Random Generated Puzzle

Using "Cross Block" as experiment game, and set game board size as  $10 \times 10$ , program can random-generated up most to 100 squares puzzle. Here is the number of puzzle that used in my experiments.

- Puzzle in each cross\_out's solved step = 100
- Puzzle in each cross\_out(solved step 2~20) :  $19 * 100 = 1900$
- Total Puzzle Amount(cross\_out\_2 ~ cross\_out\_9):  $1900 * 8 = 15200$
- We don't generate cross\_out\_1, cross\_out\_10 and Solved Step 1 because it dead ends is 0 and meaningless to be a puzzle.
- After remove repeat puzzles, we get Total Puzzle Amount: **15155**.

### Section 4.1.2: Implement Phase 1: Random Generated Puzzle

By consider all puzzles as some kind of state, this research we use GameState class to record game board and it property, like branch, dead ends...etc. Function prototype for generate puzzle looks like following:

```
void buildDatabase(int nGame, int maxStep, int crossOut).
```

There have 3 parameters:

- **nGame** : How much puzzle do you want to generate for each step?
- **maxStep**: How long of puzzle you desire? If game board size can't contain more steps then program assigned, it will simply generate it max step. For example, if we assign maxStep as 20, but our puzzle instance can only generate no more than 15 steps puzzle, in such situation, the program will generate 15 steps.
- **crossOut**: how much block you can cancel with each step? For game board size = 10, it range is 1 ~ 10.

Intuitively, cross out 1, 10 and step=1 are meaningless, therefore we generate it from step 2. Refer to Code 8:

```
[C++ code]
vector<GameState*> states;
int sizex = 10; int sizey = 10;
void buildDatabase(int nGame, int maxStep, int crossOut){
    for(int step = 2; step <= maxStep; step++){
        for(int i = 0; i < nGame; i++){
            GameState* state = new GameState(sizex, sizey);
            state->randomGenerate(crossOut, step);
            states.push_back(state);
        }
    }
}
Call:
for(int i = 2; i < 10; i++){
    buildDatabase(100, 20, i);
}
[C++ code]
```

Code 8 Implement for random generate *cross block*.

Vector “states” is our puzzle database for store generated puzzle. And, int sizex and sizey indicate our game board size.

But, what is the mechanism of the function randomGenerate(crossOut, step)? Figure 14 shows the process of randomGenerate function: Cross out = 5, solved step = 4.

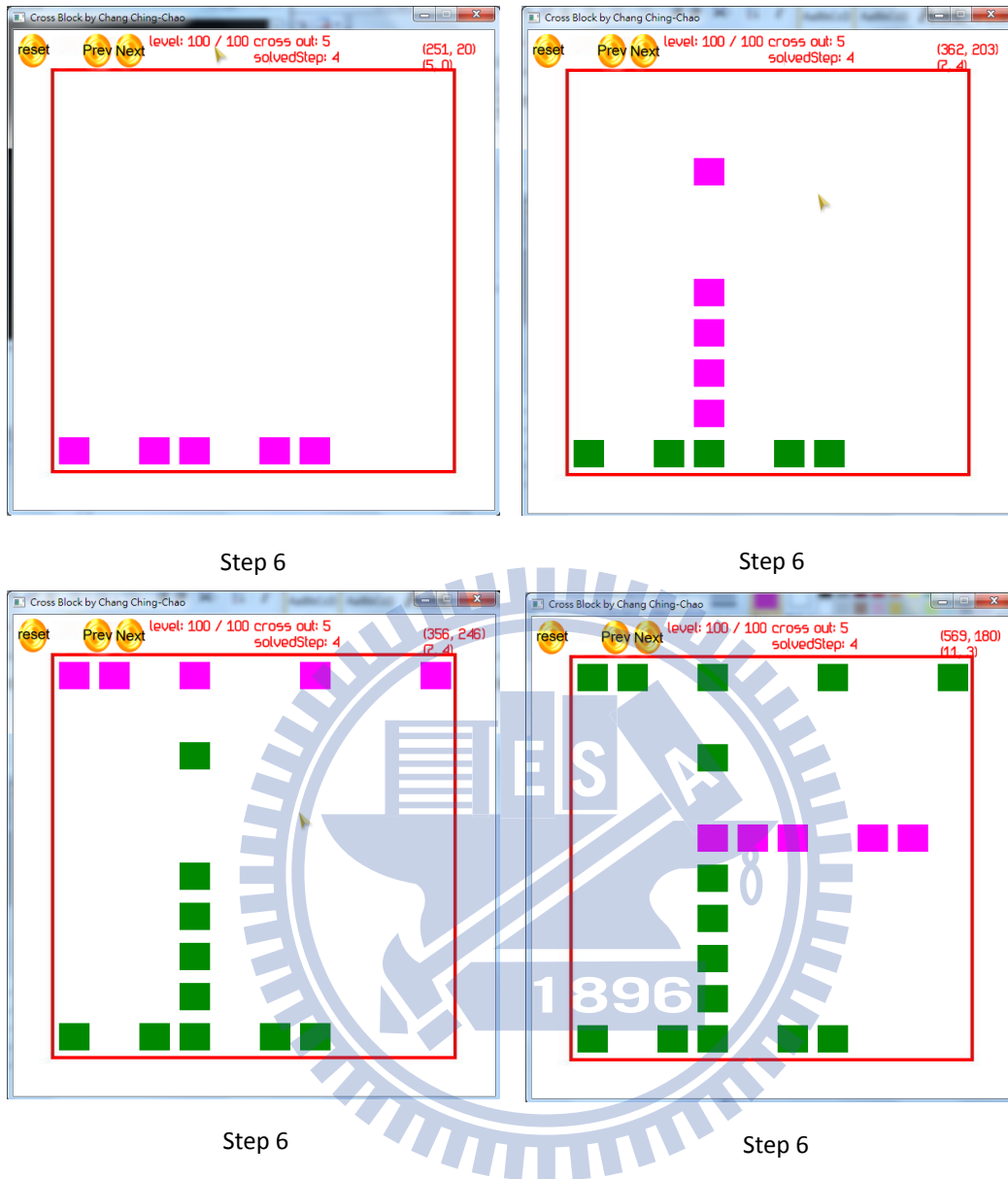


Figure 14 Example of random generated process of puzzle “cross block”.

### Section 4.1.3: Implement Phase 2: Calculate Branch and Dead Ends

Generally speaking, expand all node will get more accurate result. But I don't do that in this research, why? Because there has many puzzles is NP-Complete problem. It takes too much time, and either impossible to calculate for some complex puzzle. If we only expand answer node (from random generate process, we know it), we can reduce problem to linear time. Figure 15 shows our calculate process.

For detail implement code, please refer to appendix.

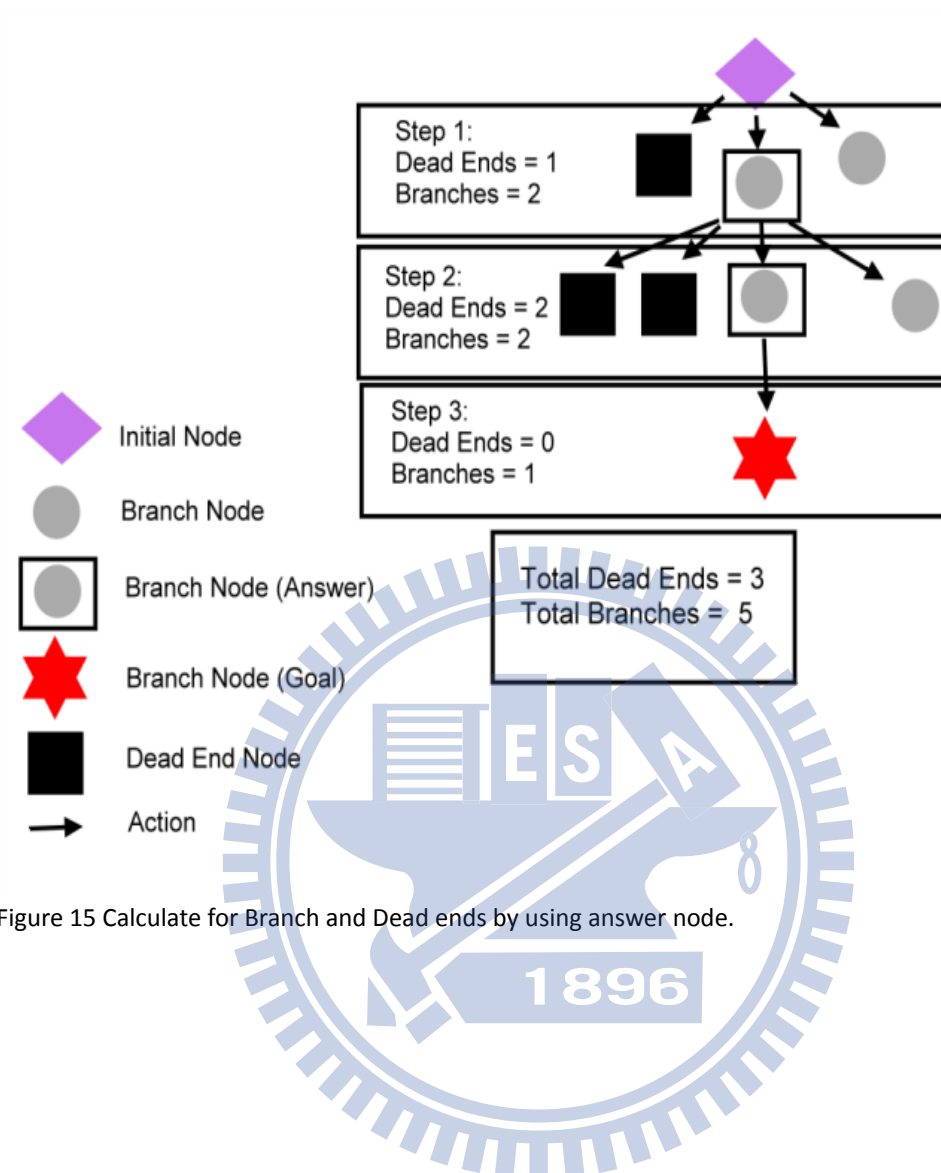


Figure 15 Calculate for Branch and Dead ends by using answer node.

### Section 4.1.4: Result of Phase 3: Branch and Dead End

From the results Figure 16 and Figure 17, we can see both Branch and Dead Ends are increase when solved step increase.

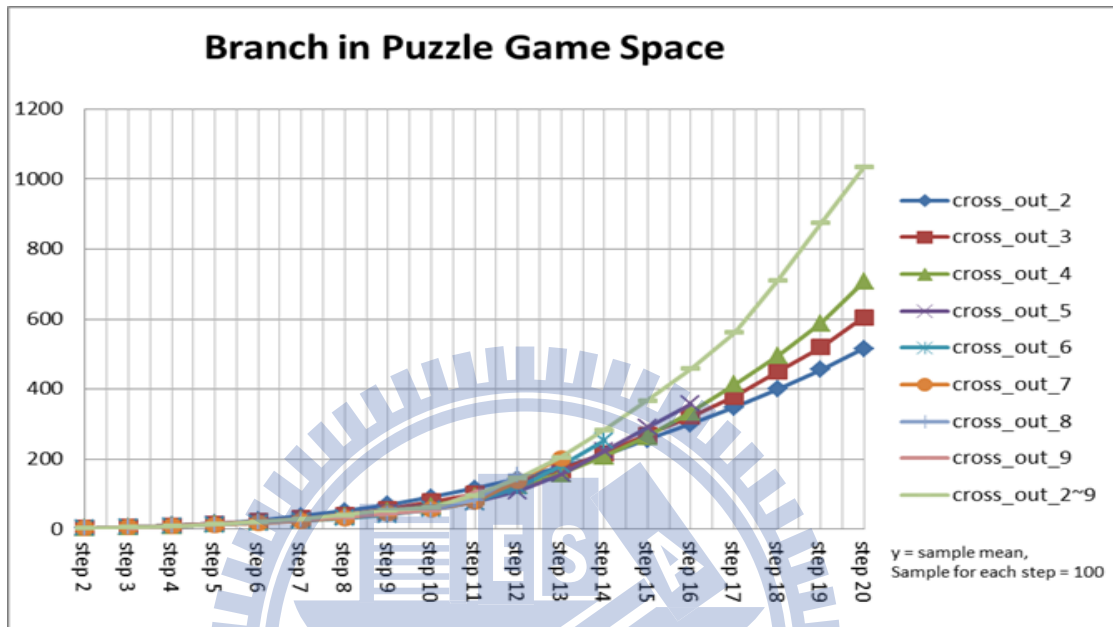


Figure 16 Average Number of Branches of each step

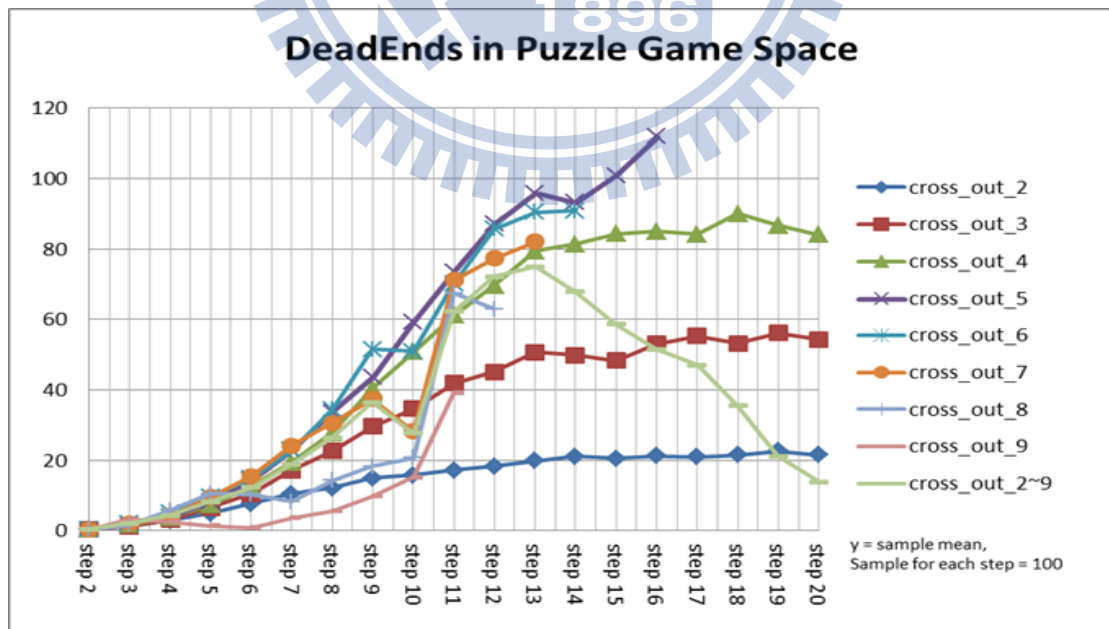


Figure 17 Average Number of dead ends of each step



### Section 4.1.5: Implement Phase 4: Calculate Complexity

Calculate model used in this research are refer in chapter 3, notice that both Branch and Dead Ends are normalized because we need a method to control their weighting.

How to implement normalize function? Because weighting between Branch and Dead Ends are different for complexity measure, therefore normalize function can help us doing parameter adjust for these two criterions. For complexity, it helps us to explain result. We can simply divide by max value in Puzzle Database generated in phase 1, like following:

- $\text{Normalize}(\text{Branch}) = \text{Branch} / \text{MAX}(\text{All Branch in Puzzle Database});$
- $\text{Normalize}(\text{DeadEnds}) = \text{DeadEnds} / \text{MAX}(\text{All Dead Ends in Puzzle Database});$
- $\text{Normalize}(\text{Complexity}) = (\text{Complexity} + \text{MAX}(\text{All Complexity in Puzzle Database in case parameter B and D is negative}) / \text{MAX}(\text{All Complexity in Puzzle Database});$

In this experiment, we set parameter B and D as 1 for simple, and then all max value is summarized in Table 1:

Table 1 Max values in puzzle database.

	Before normalize	After normalize
MAX Branch	1640	1
MAX Dead Ends	198	1
MAX Complexity	1.567916	1

Code 9 is the implement for complexity calculate model.

```
[C++ code]
normalize(int filter, std::vector<GameState*> states){
    switch(filter){
        case COMPLEXITY:
            //in case parameter is negative
            float maxC = MAX(ABSCOMPLEXITY, states);
            for(int i = 0; i < states.size(); i++){
                states[i]->complexity += maxC;
            }
            //normalize
            maxC = MAX(COMPLEXITY, states);
            for(int i = 0; i < states.size(); i++){
                states[i]->complexity = states[i]->complexity / maxC;
            }
            break;
    }
}

normalize(int filter, GameState* state, std::vector<GameState*> states){
    switch(filter){
        case BRANCH:
            return (float)state->nBranch / MAX(BRANCH, states);
            break;
        case DEADEND:
            return (float)state->nDeadEnd / MAX(DEADEND, states);
            break;
    }
    return 0;
}

[C++ code]
```

Code 9 Implement normalize function for BD-Complexity Calculate Model

### Section 4.1.6: Result of Phase 5: Complexity Sample Mean

From Figure 18, we can see it validates our Complexity Property 1: when solved step increase, puzzle's complexity should increase.

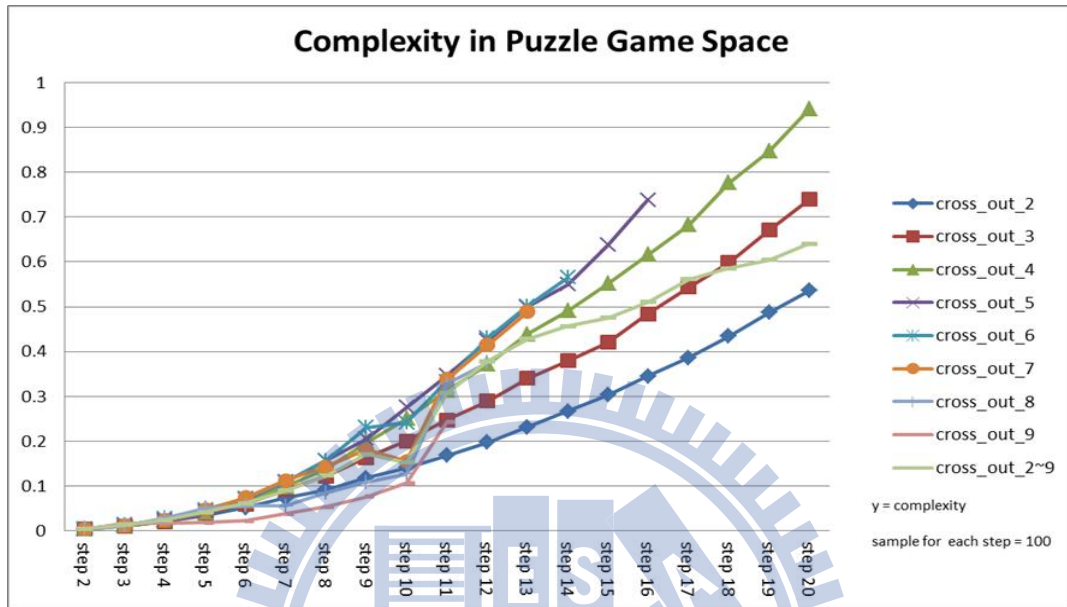


Figure 18 Average complexity of each step

Next, in order to show more clear evidence about Complexity Property 2, Figure 19 is the average complexity over all steps.

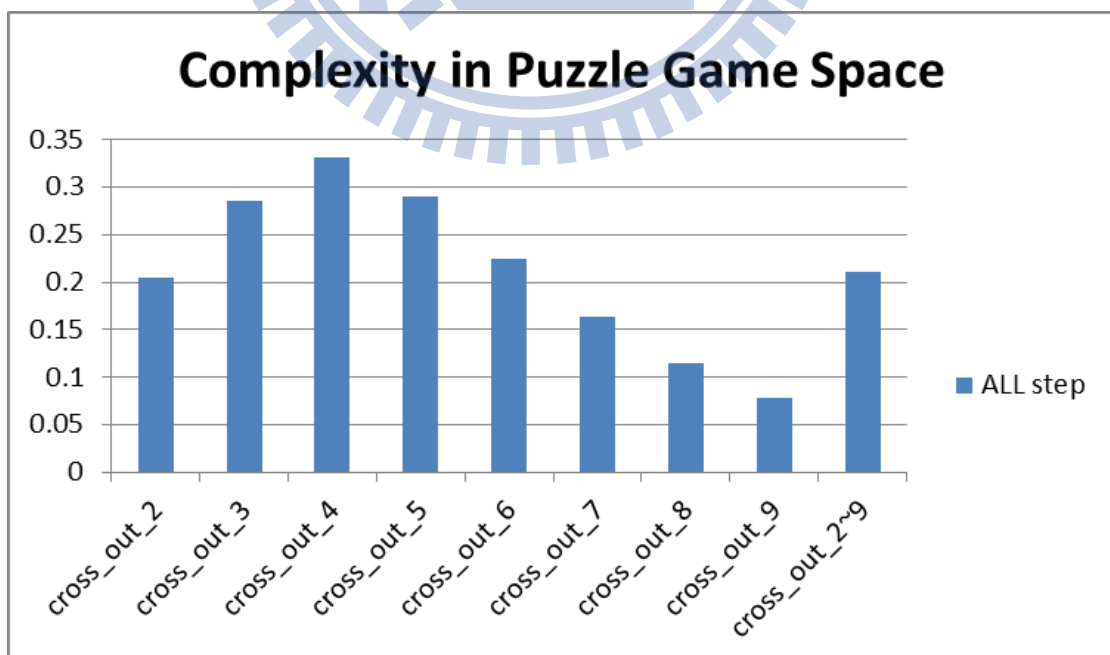


Figure 19 Result of Average all solved step complexity.

From result, we can see cross\_out\_4 is most complexity one, it is because Property 1 is true, therefore the result above is affected.

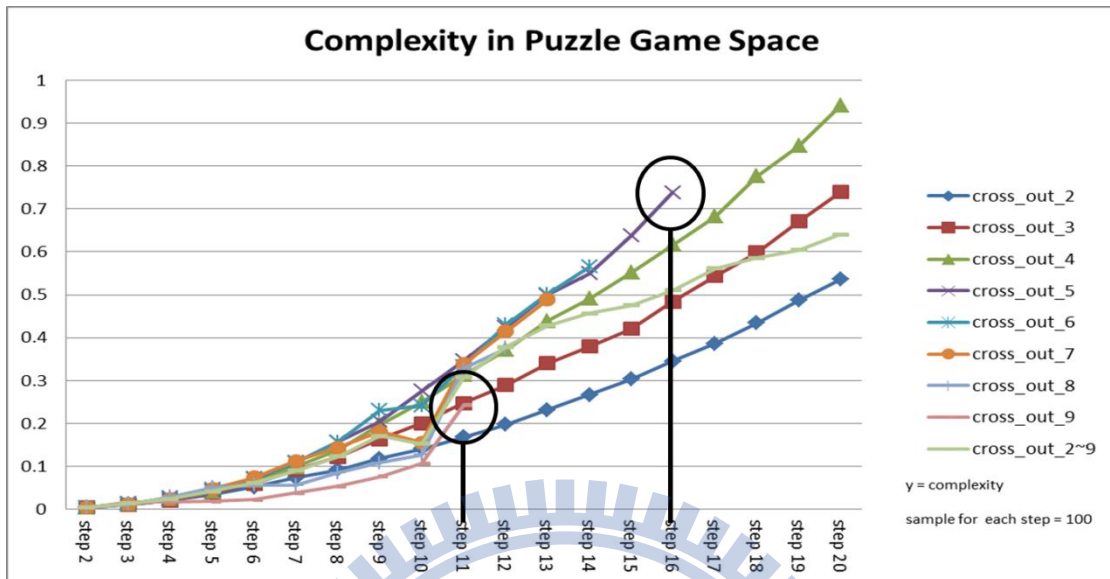


Figure 20 More Detail of complexity mean in puzzle game space.

From the Figure 20, we can see not all cross\_out game can reach solved step 20, therefore we decrease solved step to step 16 and step 11, and then we can see our Complexity Property 2 is proved: When cross\_out is about half of game board, then it complexity should be highest. In this experiment is cross\_out\_5 ( $10 / 2 = 5$ ). Refer to Figure 21 and Figure 22.

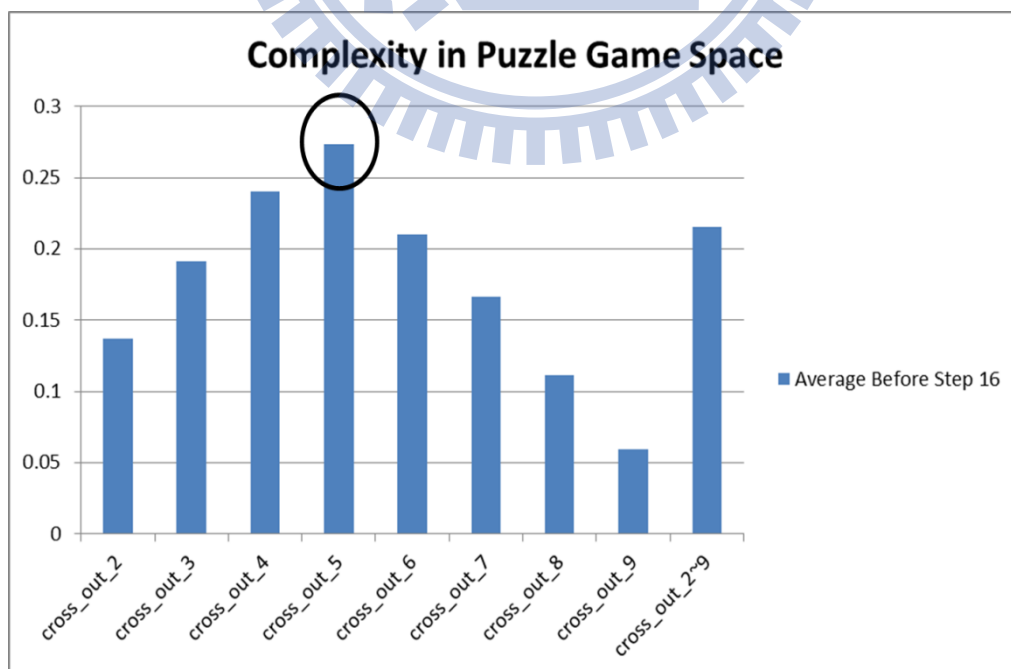


Figure 21 Average complexity before step 16.

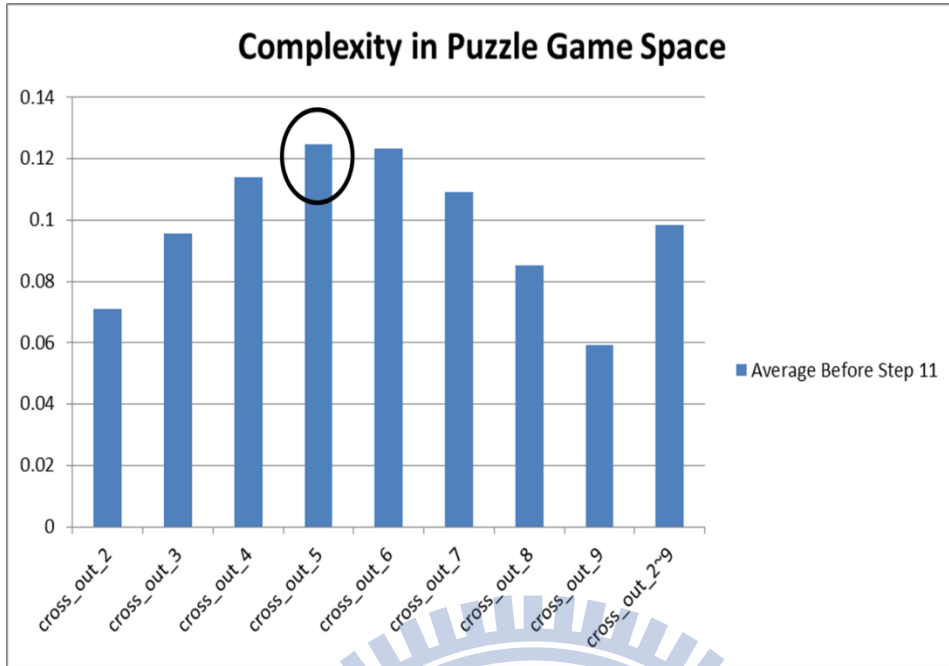
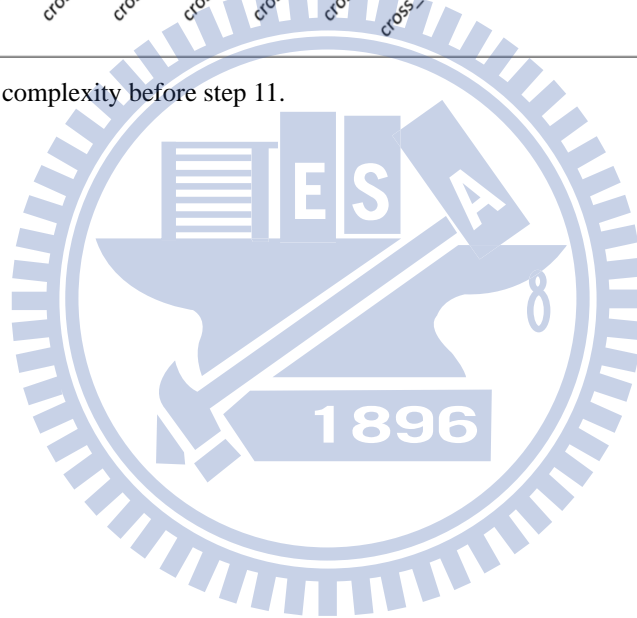


Figure 22 Average complexity before step 11.



## Section 4.1.7: Conclusion

Figure 23 shows the result of puzzle's complexity generated in phase 1 that classify into five groups of basic difficulty. Although boundaries between groups were not validated, the results are shown here as a convenient way to illustrate the distribution of puzzles in terms of difficulty level.

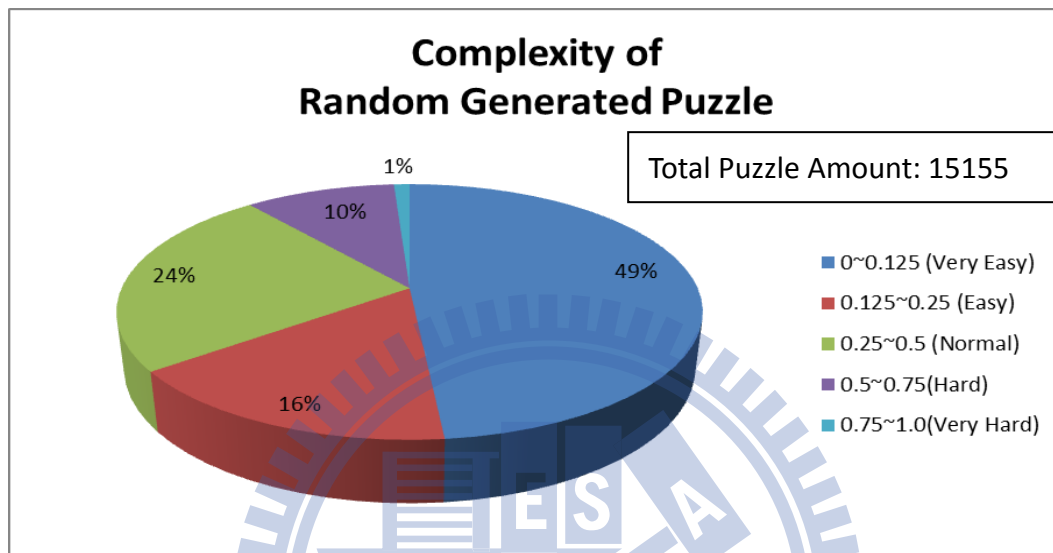


Figure 23 Ratio of basic difficulty in puzzle database.

The results indicate that approximately one-half of the puzzle levels generated by the program could be classified as very easy. According to complexity theory, when a system goes beyond a “complexity barrier”, a behavior pattern will be emergent. In puzzle games, this pattern is represented by the numbers of branches and dead ends, which increase exponentially. In Crossblock, the boundary value between periodic and complexity system is approximately 0.125, which occupy about half of puzzle in puzzle database, when value beyond it and goes higher, then branch and dead ends will increase dramatically more and more. Figure 24 shows the average complexity of each difficulty level that supports our observation. Why? Try to consider following facts: 1. Complexity interval between very easy and easy is  $0.18 - 0.053 = 0.127$ ; 2. Between normal and hard is  $0.37 - 0.18 = 0.19$ ; 3. between normal and hard is  $0.59 - 0.37 = 0.22$ ; 4. between hard and very hard is  $0.8 - 0.59 = 0.21$ .

As shown in first three, their complexity interval is gradually increased that means it must beyond a “complexity barrier”, and when complexity level is “very hard”, we know system almost reach “chaotic level” which must have highest complexity value and will gradually decrease it complexity, that why interval between hard and very hard stop to increase.

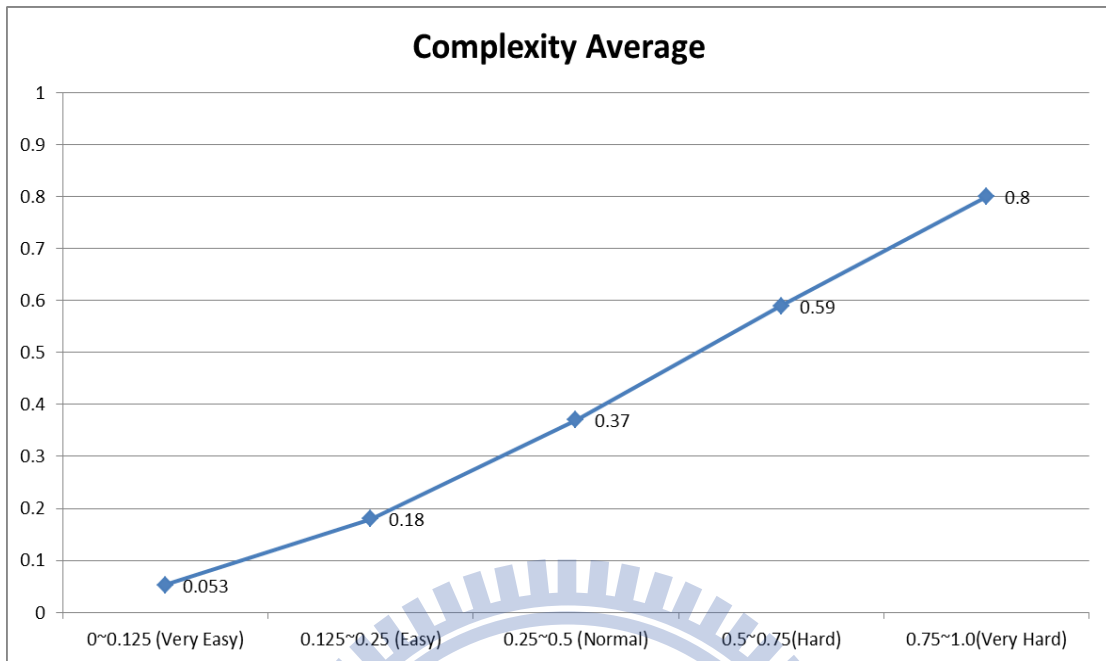


Figure 24 Complexity Average of each Crossblock difficulty level

This experiment is about Puzzle Game Space not about correct rate of puzzle levels sorting. Although I had proved both Complexity Property in this section for “*Cross Block*”, but we still need further result to show that the method proposed in this research is practical for real puzzle game sorting problem. In next experiment I will validate the correct rate between Complexity and Difficulty.

## Section 4.2: Experiment Two

### Section 4.2.1: Phase 1: Select Puzzle Levels

In this experiment, I want to test if human can really tell the difficulty if all puzzle levels have close complexity. Therefore I select 10 puzzles that all complexity in easy group and fixed those puzzles when release to player. In this research, we have 17 human evaluation data.

### Section 4.2.2: Result of Phase 3: Average difficulty and Sorting

Table 2 is the result of puzzle's complexity and difficulty in this Test Experiment.

Table 2 Complexity and Difficulty result.

	10 Puzzle Small Base Complexity	15200 Puzzle Large Base Complexity	Human Difficulty Average
P 0	0.791718	0.115397	0.48
P 1	0.990111	0.205887	0.971
P 2	0.71508	0.180588	0.658
P 3	0.778739	0.18051	0.968
P 4	0.521014	0.09901	0.695
P 5	0.843016	0.224348	0.64
P 6	0.687268	0.156099	0.867
P 7	0.704574	0.168785	0.673
P 8	1	0.219696	0.613
P 9	0.820148	0.183803	0.89

We can see the difference between Small Base and Large Base more clearly, that max value in database will affect our normalize function, all puzzle's complexity in Experiment Test that compute by large base are in very easy and easy group. Because what we want to know is their sorting correct rate, therefore sorting those puzzles according to the value in table above, we can get the rank for each puzzle. Like Figure 25:



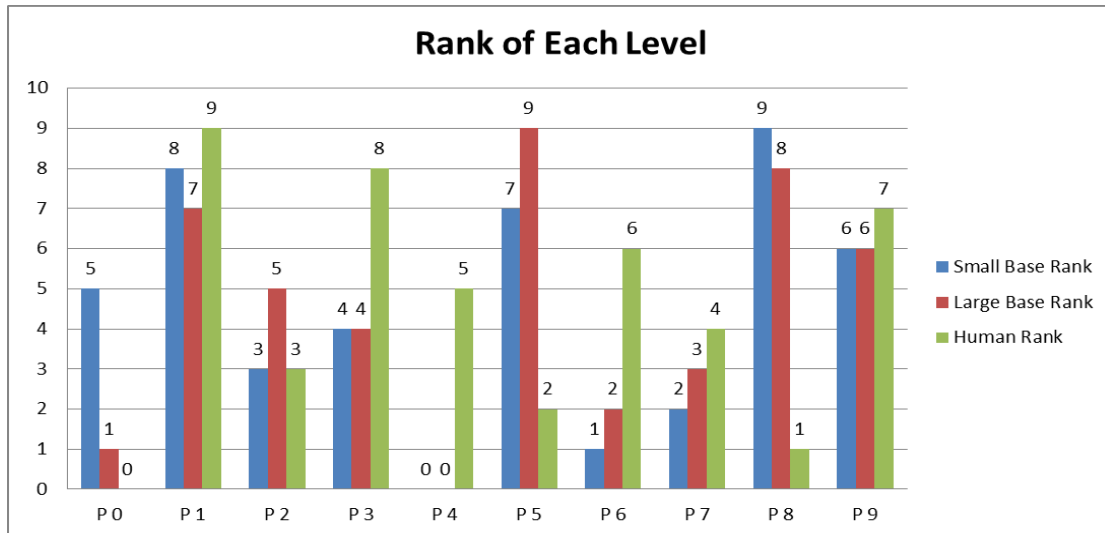


Figure 25 Complexity and Difficulty rank result

Actually, we get different sorting rank for small base, large base and human, it is not convenient for us to compare the result by figure. Therefore, we must design a method that can tell the sorting similarity rate between each rank list.

### Section 4.2.3: Implement: Calculate Sorting Similarity

Here is my implement method for sorting similarity:

- Set if we have two sorted puzzle lists: listA and listB, all puzzles in lists are same but sorted by different method.
- set listA is sorted by complexity
- set listB is sorted by difficulty (human or static difficult)
- If puzzle's rank in two lists is same, then similarity add 1
- If puzzle's rank in two lists is different, then similarity add  $(1 - \text{different of two list}) / \text{list size}$
- Finally, before return the value, divided it by rank list size in order to normalize result.

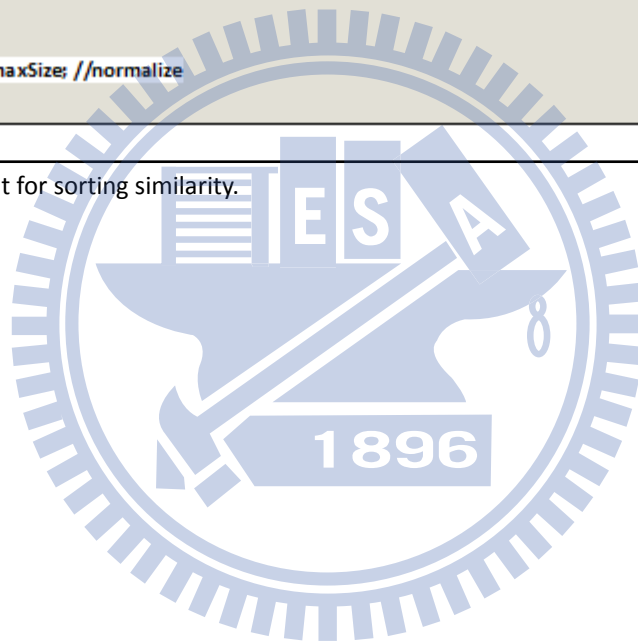
Code 10 is actual implement code for sorting similarity:

[C++ code]

```
float similarityOfRank(vector<GameState*> listA, vector<GameState*> listB){
    float result = 0;
    float maxSize = max(listA.size(), listB.size());
    for(float i = 0; i < listA.size(); i++){
        GameState* stateA = listA[i];
        for(float j = 0; j < listB.size(); j++){
            GameState* stateB = listB[j];
            if(stateA->isSame(stateB)){
                if(i == j){
                    result = result + 1;
                }else{
                    float diff = abs(i - j) / maxSize;
                    result = result + (1 - diff);
                }
            }
        }
    }
    return result / maxSize; //normalize
}
```

[C++ code]

Code 10 Implement for sorting similarity.



#### Section 4.2.4: Result of Phase 5: Sorting Similarity

We compare two rates for two ranked lists in Table 3. First is the percentage of match, it means number of same rank in both. Second is Similarity, it means how similar sorted of two lists.

Table 3 Result of match and sorting similarity.

	Small Base VS. Large Base	Small Base VS. Human	Large Base VS. Human	Human VS. Human
Match	30%	8%	10%	24%
Similarity	86%	61%	64%	68%

We can see small base and large base actually have different rank because max value in database will affect normalize function. Furthermore, compare to small base, large base has higher sorting similarity rate between human. Finally, we compare each people's sorting similarity, their sorting similarity only reaches 68%, it seem surprising that every people have different feeling about difficulty when puzzle have near complexity levels.

#### Section 4.2.5: Conclusion

In this experiment, we see when puzzle have near complexity, then people tends to have different rank because of different skill they have. Therefore, I think the ability that can classify a puzzle into basic difficulty is more important than tell their actual degree.

## Section 4.3: Experiment Three

In this experiment, I want to validate the correct rate of complexity sorting by using Sudoku that had been classified by other method. You can find the puzzle sample we used in TSA(W. Kuang-Chen (巫光楨), 2008).

### Section 4.3.1: Phase 1: Select Puzzle in Each Rank

Like Figure 26, every puzzle in TSA is marked with a difficulty level. Number of “★” of a puzzle indicates difficult rank calculated by TSA, they classify all *Sudoku* into 5 ranks.

題號	sudoku	挑戰次數	成功次數	成功率	平均用時	最新記錄	最快記錄	挑戰
298	<pre> 5 6 9 0 1 4 4 9 1 6 5 7 9 3 5 4 2 5 2 8 7 2 8 4                     </pre> <p>★</p>	54	46	85.18%	3'40"	1. 2070 2'41" 2. anz 5'07" 3. jopye 1'52" 4. gao540 2'08" 5. 12456 2'59" 6. conan1000 1'30" 7. YUYU 4'09" 8. abop 5'26" 9. jiluan 2'22" 10. 龍圖 1'43"	1. meay 1'15" 2. 龍 1'27" 3. conan1000 1'30" 4. aaata 1'42" 5. 龍圖 1'43" 6. jopye 1'52" 7. aaata 1'57" 8. gao540 2'01" 9. jopye 2'05" 10. 龍 2'14"	C
1907	<pre> 1 2 6 3 2 5 4 8 7 5 5 6 1 7 4 6 8 4 2 7 5 1 4 8 3 5 1 2 5 6 8                     </pre> <p>★★</p>	74	63	85.13%	3'03"	1. 5296 2'09" 2. conan1000 1'27" 3. aaata 1'36" 4. aaata 1'46" 5. conan1000 1'43" 6. 龍圖 1'32" 7. mas 3'00" 8. 128789 2'51" 9. 128789 3'44" 10. 35165 3'24"	1. meay 0'56" 2. 龍便玩玩 1'18" 3. TS 1'22" 4. conan1000 1'29" 5. shavonon 1'31" 6. 龍圖 1'32" 7. aaata 1'36" 8. mas 1'38" 9. 龍 1'41" 10. 古★古★古★ 1'42"	C
3788	<pre> 2 1 4 6 7 3 8 7 5 7 9 7 2 5 1 2 9 1 6                     </pre>	47	40	85.10%	1'43"	1. mas 1'23" 2. az 1'48" 3. 龍 1'08" 4. 龍 1'09" 5. 龍 1'59" 6. 龍 1'20" 7. 龍 1'07" 8. 龍 1'17"	1. meay 0'45" 2. 龍便玩玩 1'00" 3. conan1000 1'01" 4. 龍 1'03" 5. 龍 1'06" 6. 龍 1'07" 7. 古★古★古★ 1'08"	C

Figure 26 Sudoku Puzzles provides in TSA.

Every puzzle is marked with a difficulty level. Number of ★ indicates how difficult it is, upmost to five star. Meaning in each column: puzzle id, puzzle, number of challenge, number of success solved, solved rate, average time, newest record, fastest record, start challenge the puzzle.

The method used by TSA to measure difficult of a “Sudoku” is to evaluate number of solve technique that a puzzle solving program require. The more difficulty technique a puzzle required, and then the puzzle is more difficult. But, because we don’t know whether the difficult level that marked by TSA is really correct or not, therefore when choice the puzzle from it, we must take care of this issue. Fortunately, TSA also provide solved rate in the column five for each puzzle, therefore we can choice the puzzle based on this value that will reflect their difficulty more correctly. In

this experiment, we select 100 Sudoku puzzle for each difficult level. (5 \* 100 = 500 puzzles)

### Section 4.3.2: Result Phase 3: Calculate Branch and Dead Ends

Before calculate complexity for each puzzle, we must decide parameter B and D. By observe result in Figure 27 and Figure 28, we know branch is positive relation and dead ends is somehow negative relation (normal and hard are not) when difficulty increase, therefore, we set B as 1 and D as -1.

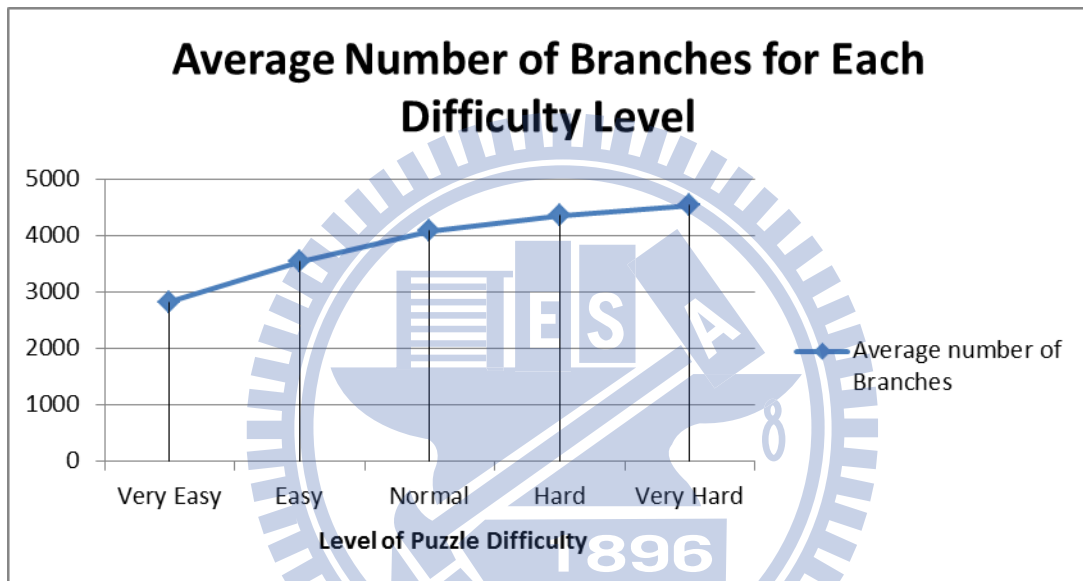


Figure 27 Average branch for each difficulty levels.

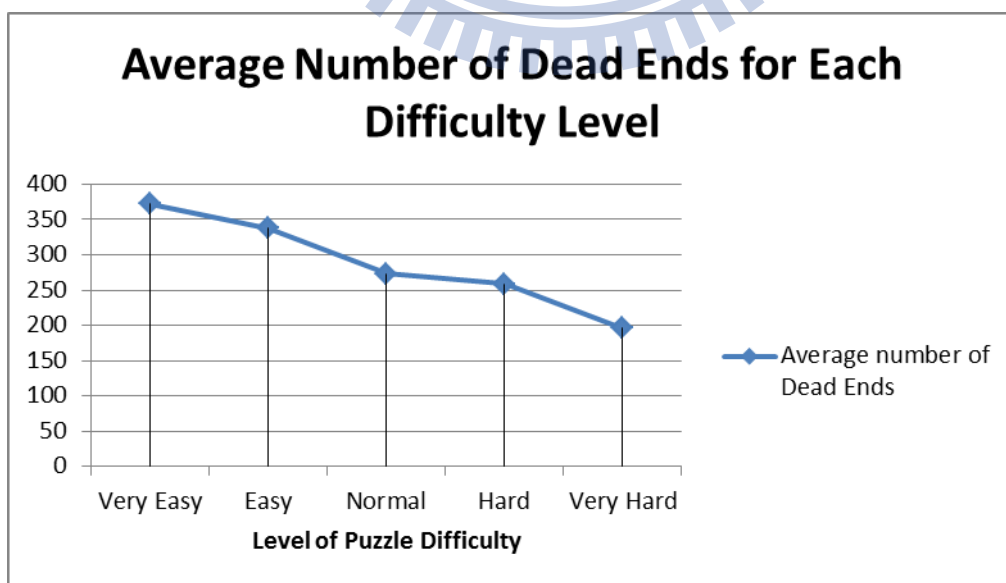


Figure 28 Average dead ends for each difficulty levels.

By using complexity calculate model describe in chapter 3, we get the result in Figure 29:

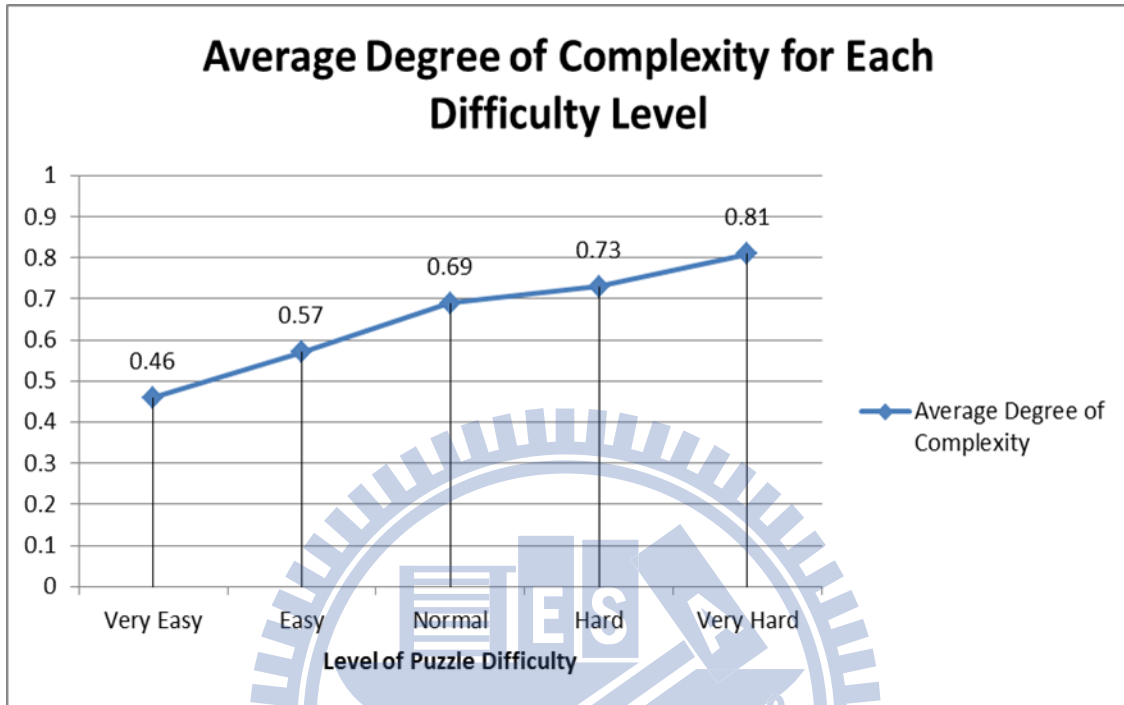


Figure 29 Average Degree of Complexity for Each Difficulty Level

We can see complexity is increase according to difficulty level. Therefore, our method is successful to approximate difficulty of puzzle at minimum requirement. How about overall success for each puzzle? Let examine more detail about complexity we calculate in Figure 30:

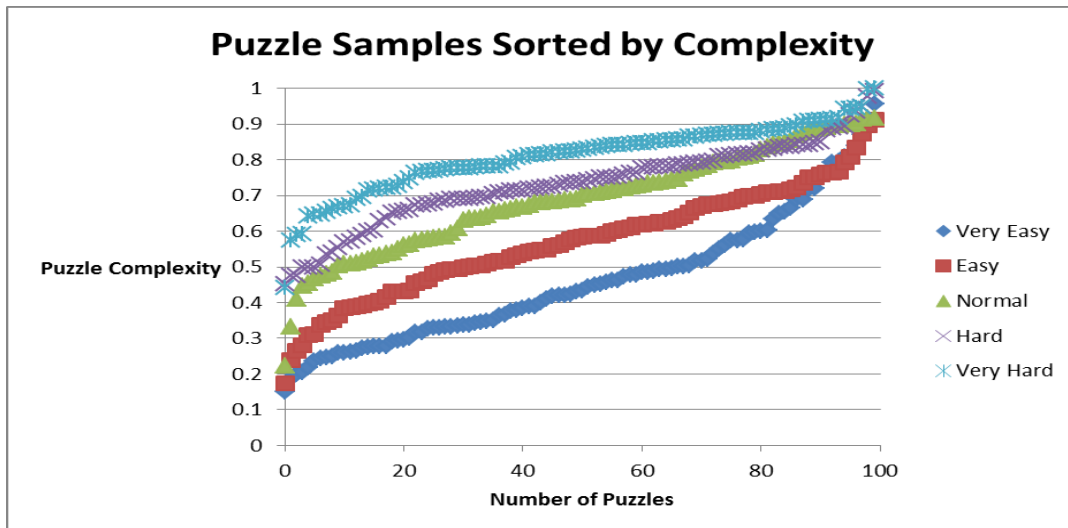


Figure 30 Puzzle Samples Sorted by Complexity

It just put every puzzle into a rank from left to right in Figure 31, and we can see this method is weak on those puzzles have both high or low branch and dead ends which means our complexity calculation will become too high or too low. Another problem may be the puzzle in normal and hard, we can't classify the puzzle in these two groups clearly—I think both of problems is caused by the property of our method. Because we simply combine branch and dead ends as a polynomial, therefore the method used to calculate branch and dead ends will affect result very large. In this experiment, we only introduce a heuristic that simply skip “unique method” step, which every novice player will know this technique, when we doing calculation. In order to get more concrete result, we may need to figure out more concrete heuristic when calculate branch and dead ends.

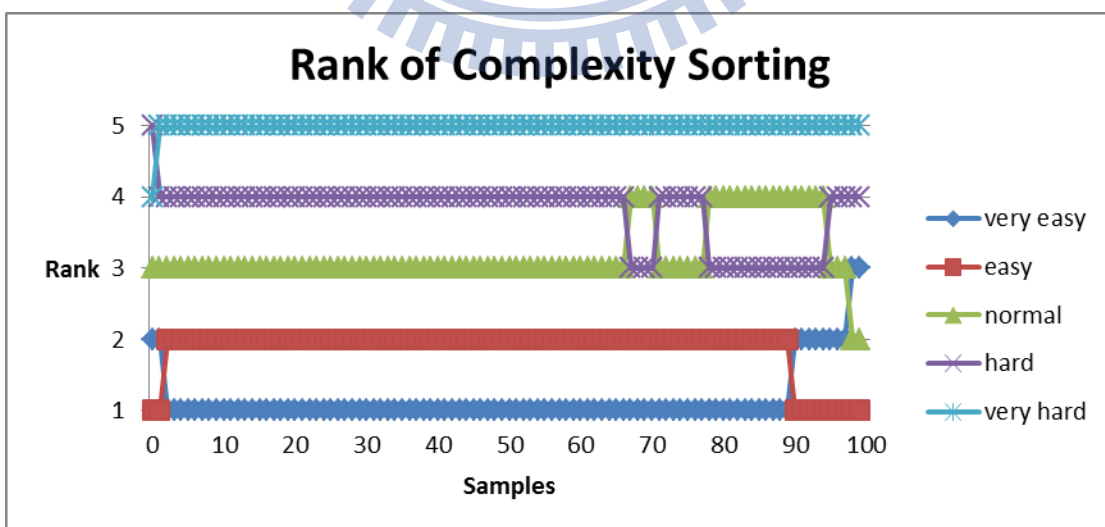


Figure 31 Rank of complexity Sorting for each puzzle samples.

### Section 4.3.4: Result Phase 5: Compare Rank Result

Because there five marked level difficulty in our puzzle database, therefore we can randomly select one sample from each difficulty level (total 5puzzles) as listA, sort it by our complexity as listB, and then we can compute sorting similarity between these two lists. Select process like Figure 32:

listB = sort listA by complexity

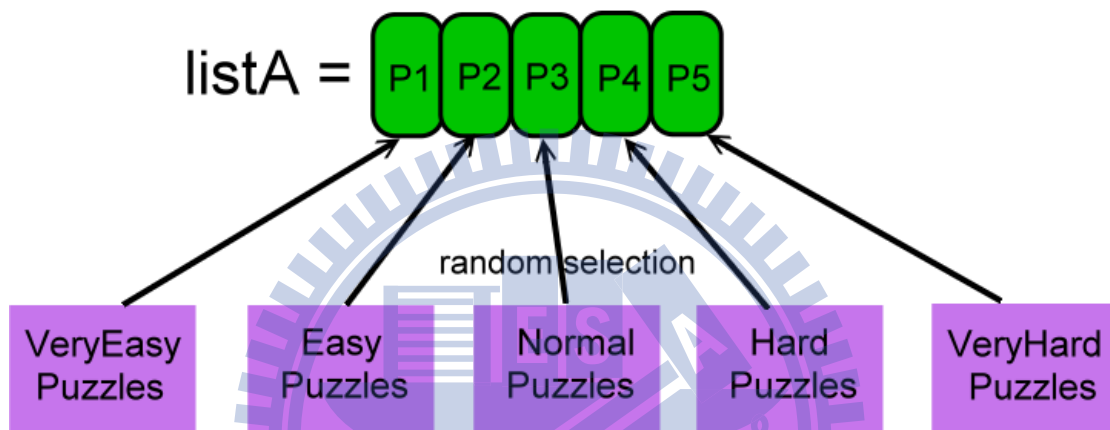


Figure 32 Process of select sample from puzzle database as sorting list.

By repeat large enough iteration of this comparing process, then we can validate the correct of our method. Figure 33 is the similarity result that iterates over 50000 times:

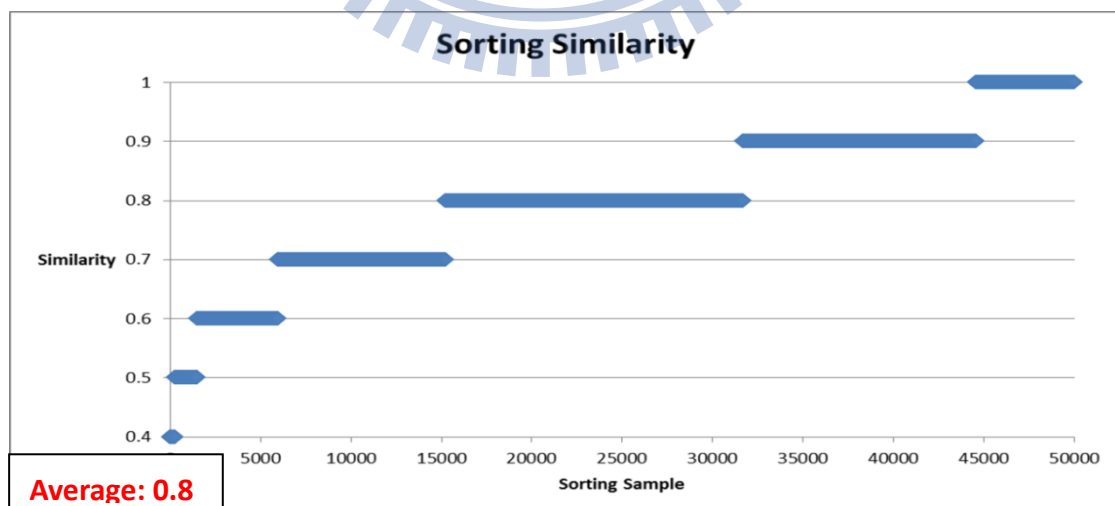


Figure 33 Result of Sorting Similarity



It shows that our sorting looks quite good on most of case, but there still have space for improve. I will try to adjust parameter B and D by machine learning to find out best result of complexity sorting.

### **Section 4.3.5: Conclusion**

This experiment shows the ability of the method we propose can calculate different type of puzzle games. But because different puzzle have different emergent phenomena on their branch and dead ends, therefore sorting correctness will dependent on play feature of different game. By separate all pure puzzle game as following three types: Movement type puzzle like “*Sokoban*”, Elimination type puzzle like “*Cross Block*” and Fill Out type puzzle like “*Sudoku*”. I think most suitable puzzle game for apply the method we propose is Elimination and Movement type. Because possibility of action that player can operate is too large, that generate more exception than other types of puzzle.

In appendix, I collect more puzzle games according to this classification. Although complexity measure for Fill Out type puzzles in this “*Sudoku*” experiment doesn’t perform as good as previous “*Cross Block*” experiment, but I think it is good enough for real application.

## **Section 4.4: Experiment Four**

In this experiment, we use simulated annealing to adjust our parameter B and D in order to get more correct complexity evaluation for experiment three.

### **Section 4.4.1: Phase 1: Select Training Sample**

Because simulated annealing is a machine learning technique, therefore, we need training sample before beginning our tuning program. Figure 34 is our training samples select process: we randomly make 1000 training sample from puzzle database

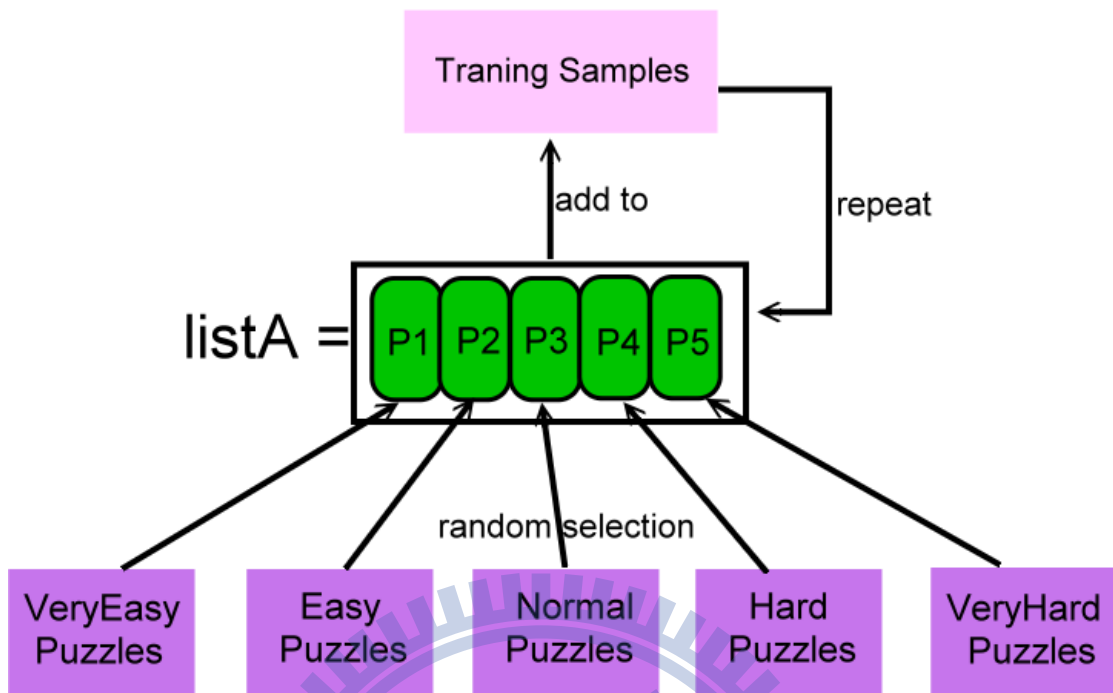


Figure 34 Training sample select process.

### Section 4.4.2: Implement Phase 2: Parameter Tweak

Because our purpose is to improve sorting similarity, therefore we can implement our energy method for simulated annealing as Code 11:

```
[C++ code]
TrainingData* trainingData;
float SimulatedAnnealing::energy( GameParameter *p){
    float similarity = 0;
    for(int i = 0; i < trainingData->datas.size(); i++){
        std::vector<GameState*> sample = trainingData->datas[i];
        ss.calculateComplexity(p, sample);
        vector<GameState*> listB = sortByComplexity(ASCENT, sample);
        similarity += ss.similarityOfRank(sample, listB);
    }
    return 1 - (similarity / trainingData->datas.size());
}
[C++ code]
```

Code 11 Implement for energy function in Simulated Annealing.

Because the concept of simulated annealing is to reduce energy (or error, cost) when repeat training iteration, therefore we minus 1 before returning the result.

### Section 4.4.3: Result of Phase 2: Parameter Tweak

Figure 35 is the result of training process, our adjustment is successfully converge error (1 – similarity) to 0.13.

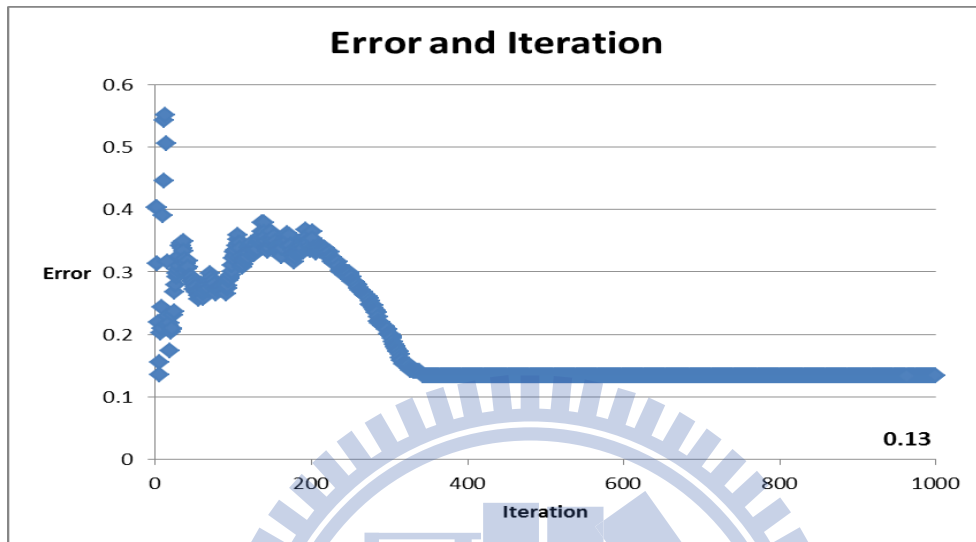


Figure 35 Error and iteration of simulated annealing.

Figure 36 shows the parameter that are adjusted over iteration in this training iteration:

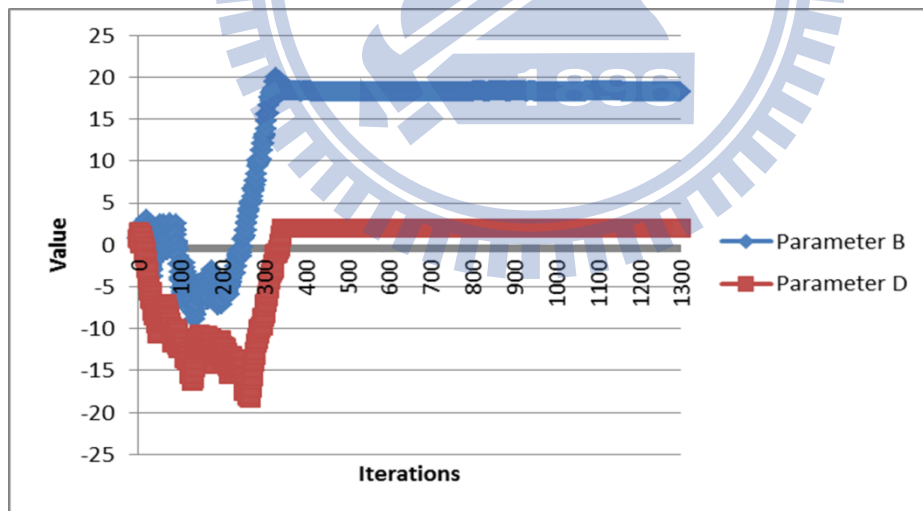


Figure 36 Result of parameter Band D adjusts over 1500 iteration.

Finally, we get  $B = 18.1952$  and  $D = 2.02334$  is one of state that has lowest error. The result may be changed when we start another training iteration.

### Section 4.4.4: Result of Phase 3: Calculate New Complexity

Figure 37 is the result of average complexity for each difficulty levels, we can

see their value is more close between each level compare to the result in experiment three:

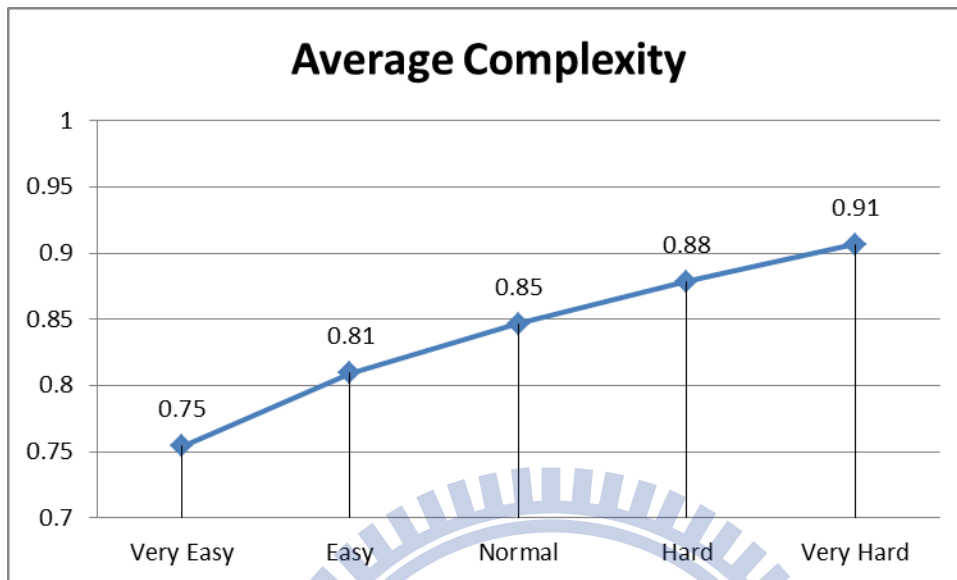


Figure 37 Average complexity for each difficulty level after parameter tweak.

But it is actually improved its result, especially for those low complexity puzzle in each level. Figure 38 and Figure 39 shows detailed sorting result:

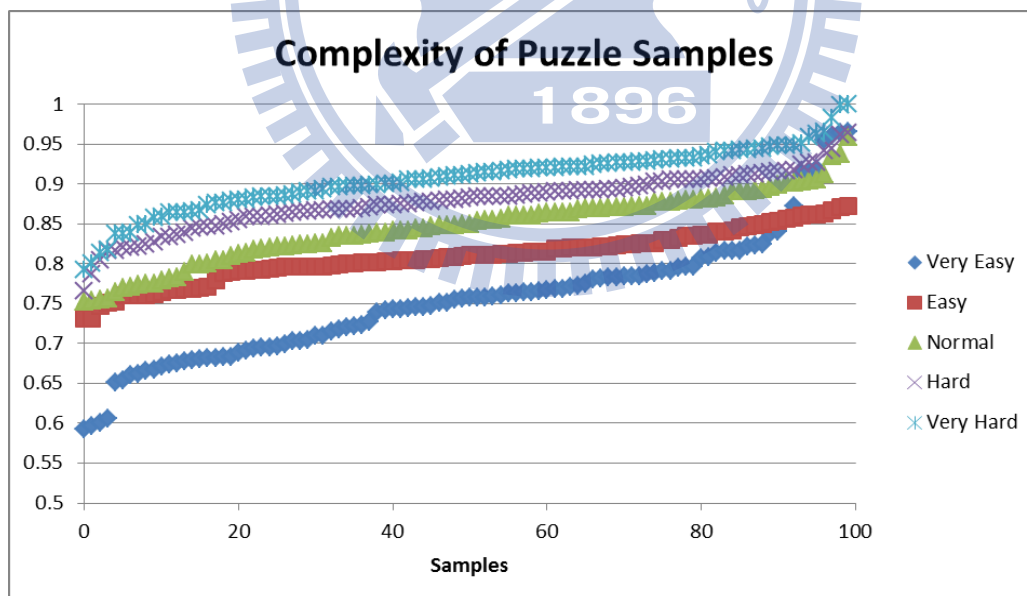


Figure 38 Complexity of each puzzle sample after parameter tweak.

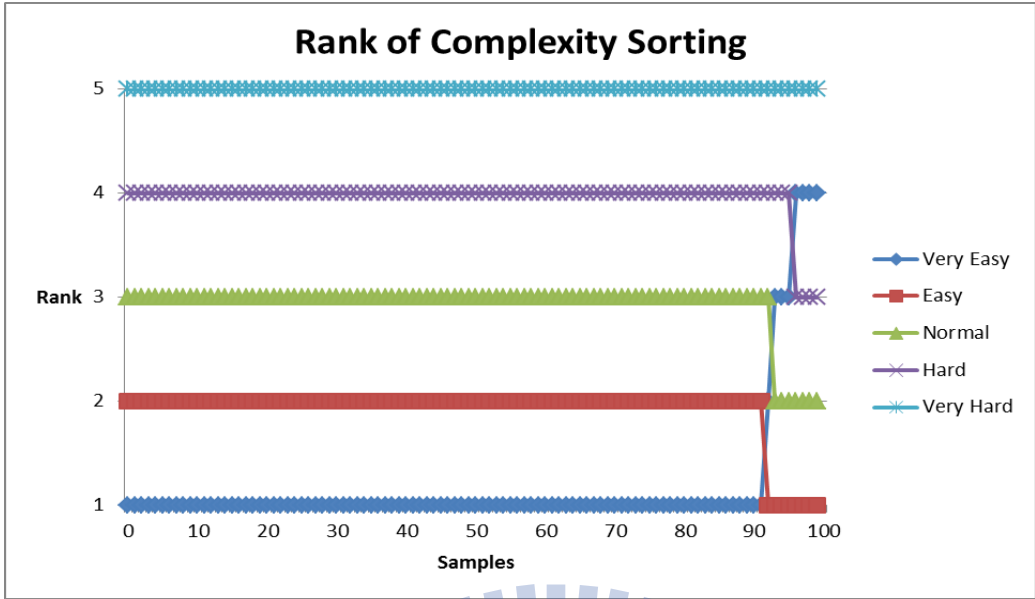
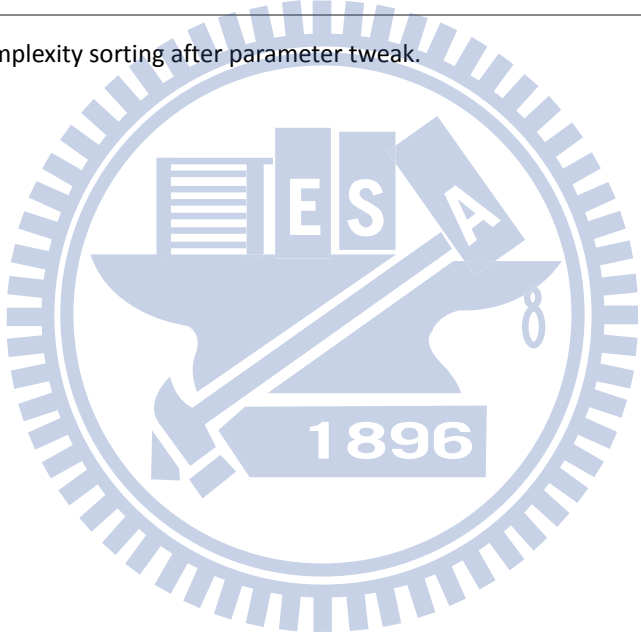


Figure 39 Rank of complexity sorting after parameter tweak.



### Section 4.4.5: Result of Phase 4: Compare Rank Result

Figure 40 shows average sorting similarity is improved from 0.8 to 0.86.

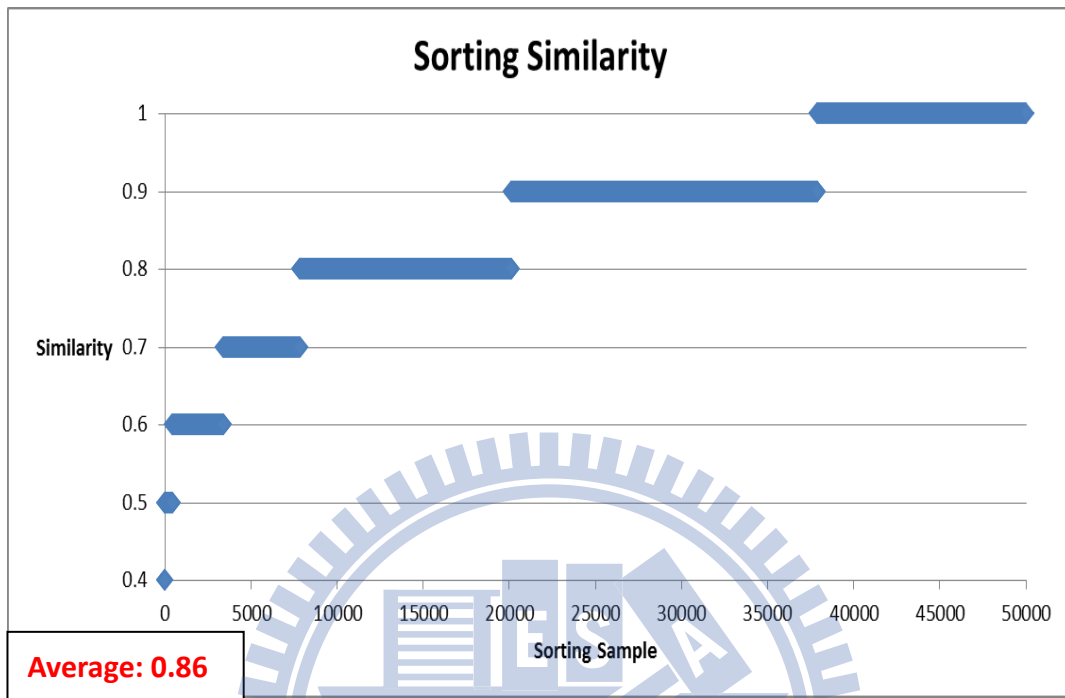


Figure 40 Sorting Similarity after training.

### Section 4.4.6: Conclusion

From the result, we can see although our method is quite simple, but it is a general method that can be used to measure difficult for different puzzle. Although there still have some error, but I think if we can figure out complexity measure heuristic for each different puzzle game, then it sorting correct rate will be improved.

# Chapter 5: Conclusion

## Section 5.1: Complexity Sorting and Difficulty Mapping

Determining game difficulty is a challenging issue requiring detailed understanding of game parameters. For puzzle games, Scott Kim has identified branches and dead ends as universal puzzle components; in this project we tried to use the two features to measure puzzle complexity. According to our experiment results, the proposed method holds potential as an efficient method for mapping complexity to static difficulty. We used simulated annealing to identify optimal parameters, but our final sorting similarity data still suffered from a 14% error rate. Since different puzzles have different emergent phenomena on their branches and dead ends, correct sorting depends on play features that differ across different games. To achieve more accurate results using our proposed method, it is therefore necessary to use game-specific features when calculating numbers of branches and dead ends in order to improve the fit between our process and behavior patterns (e.g., the ability to quickly filter out bad choices and dead ends).

For example, in Sudoku, there exist some solving techniques to help us solve the problem, like Last Digit, Hidden Single in Box...etc.,. In order apply those technique into our complexity calculate process, it is necessary to find out their emergent phenomena on branch and dead ends that can help us to identify which node we need to expand or count. We believe, more difficult technique a puzzle has, means higher complexity value it will.

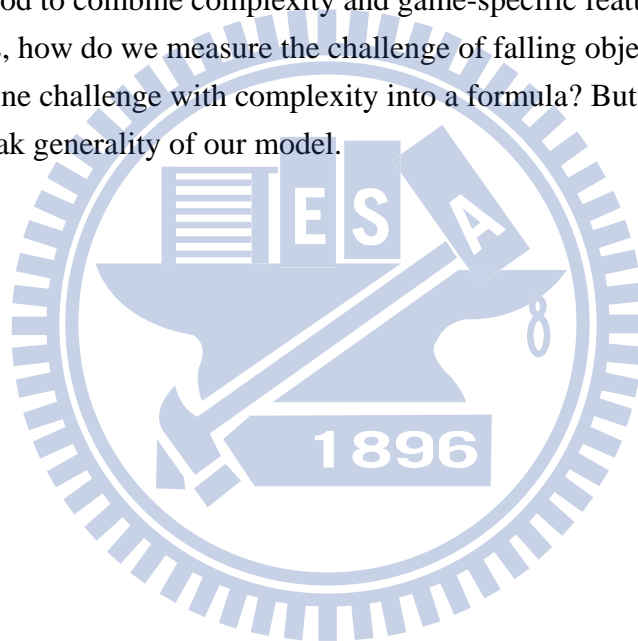
However, the use of game-specific features contradicts our goal of creating a method that can be used for all puzzle games. Therefore our plans include designing a more sophisticated complexity calculation model that considers a wider range of search tree behavior features—for example, backtracking rates (indicating incorrect choices) or number of cycled nodes.

## Section 5.2: Measuring Digital Game Complexity

Does our proposed method can apply to other games? Generally speaking, our proposed model can always apply to any kind of task—if we formulate target problem as search tree form, and then branch and dead ends can be calculated to measure complexity of the task. But, there may cause some problems when we want

to map complexity to difficulty, because there have much games require player many different kind of skill that will diverse subjective feeling about difficulty. For example, Tetris may require player eye-hand coordination, but not all people can follow the speed of falling object; and boggle will require player English ability, player who familiar with English will have obvious advantage.

Therefore, our complexity measuring result will limited to certain high skill player group and meaningless to others. Because for those players that without certain skill or knowledge can't even start play the games. Furthermore, for those medium skill players, game specific skill and knowledge will always be the source of difficulty. Because different player will have different skill, thus, diverse feelings about difficulty trouble us from map complexity to static difficulty. Therefore, we must try to find out a method to combine complexity and game-specific feature first. For example, in Tetris, how do we measure the challenge of falling object's speed? And, how do we combine challenge with complexity into a formula? But, as we discuss before, it will break generality of our model.

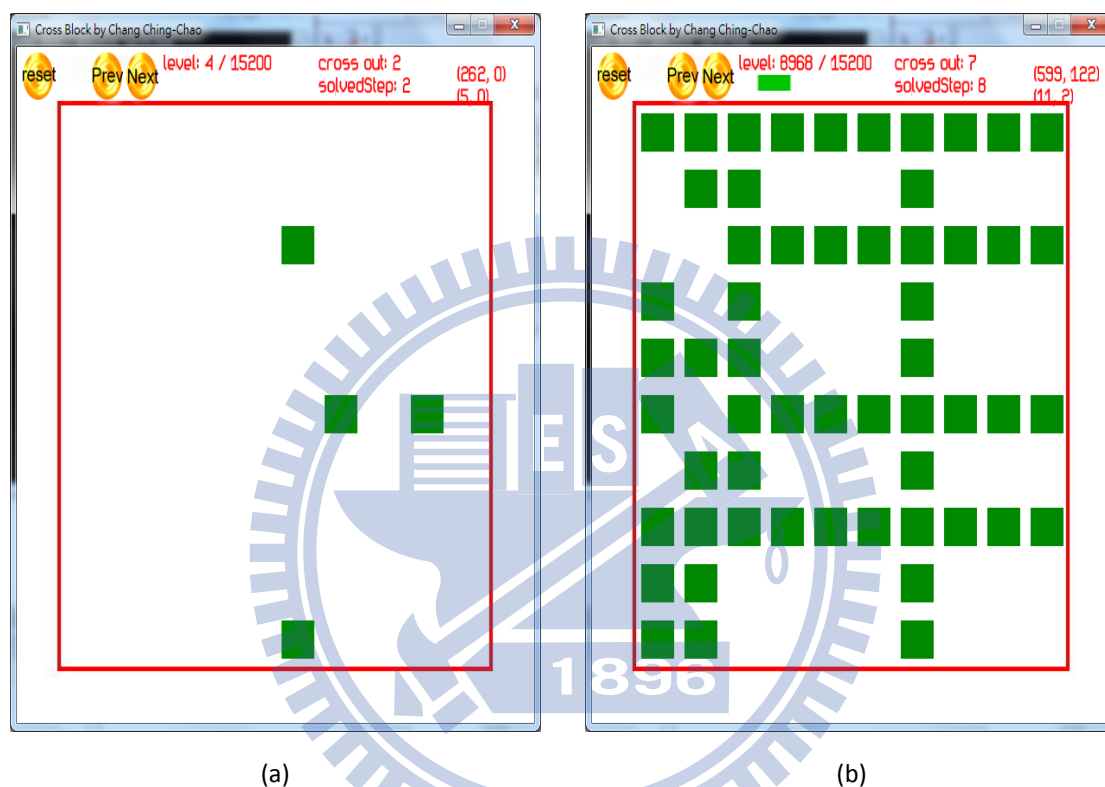




## Appendix A: Puzzles in Experiments

### B.1: Cross Block

*Cross Block* is kind of pure puzzle that invented by DJ Trousdale(DJ Trousdale, 2009), where it goal is to clear all square on game board by drawing vertical or horizon line.

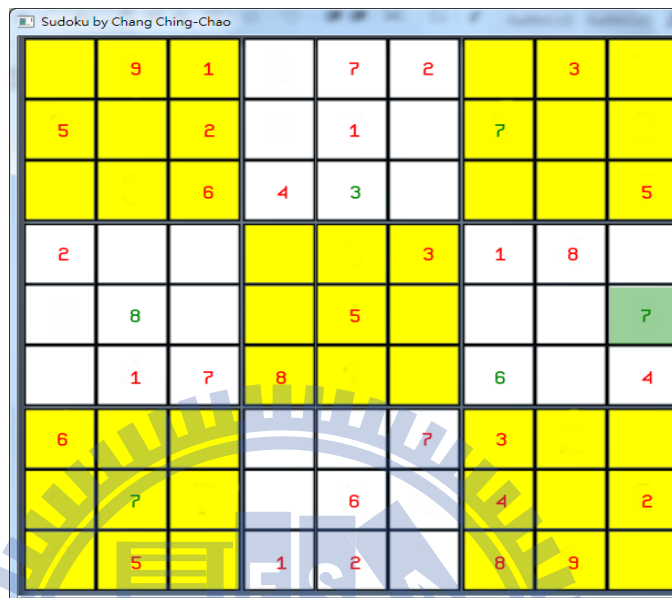


Example of Cross Block, each line must equal to specific cross out number. (a) Cross out 2 squares at one time, it requires 2 steps to solve. (b) Cross out 7 squares at one time, it requires 8 steps to solve.

Generally, we can simply increase difficulty for this puzzle, by putting more squares into game board. Like example in Figure 1.5, when solved step increase, then it difficulty also increase. Although there exists some exception, but we don't discuss about the detail here. I will show overall puzzle game space results in chapter 4.1 for *Cross Block*. Next, let's return to our problem: How to measure the difficult for a puzzle?

B.2: Sudoku

Another puzzle I will use in my experiment is “*Sudoku*” that is a very famous puzzle.



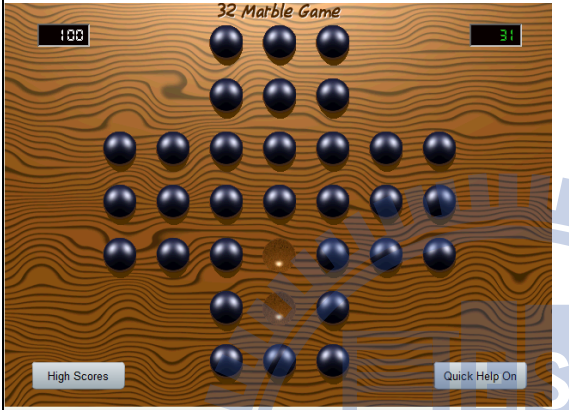
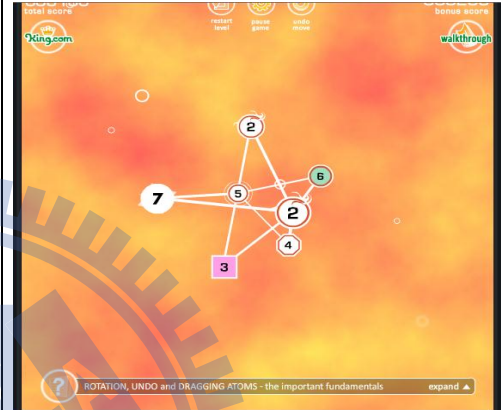

Example of Sudoku Puzzle

The goal of “Sudoku” is to fill all square with a number 1 ~ 9, but constrain with following rule: 1. the number in each row and column can’t repeat. 2. The number in each 3\*3 box region can’t repeat. For example in above figure, here have 9 box regions that marked with yellow and white color.

# Appendix B: Collection of Pure Puzzle

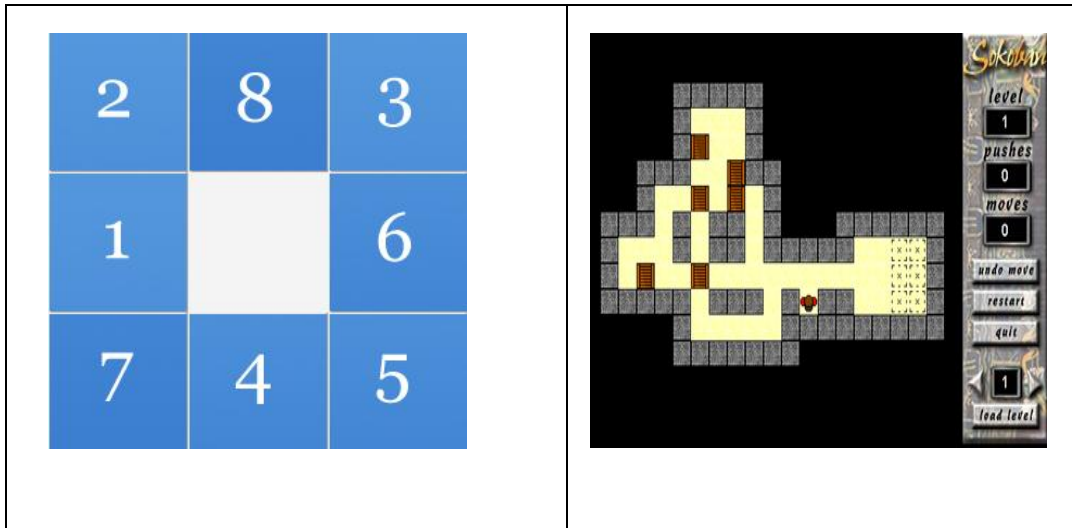
In this appendix, I simply collect some puzzle from internet according to following classification: movement type, fill out type, elimination type.

## B.1: Elimination Type

<p style="text-align: center;"><b>Marble Solitaire</b></p> 	<p style="text-align: center;"><b>Minim</b></p> 
<p style="text-align: center;"><b>NingPo Mahjong</b></p> 	


B.2: Movement Type

<p style="text-align: center;"><b>Exorbis 2</b></p> 	<p style="text-align: center;"><b>Flashmaz</b></p> 
<p style="text-align: center;"><b>Mummy Maze</b></p> 	<p style="text-align: center;"><b>Open Doors 2</b></p> 
<p style="text-align: center;"><b>Telescope</b></p> 	<p style="text-align: center;"><b>Rush Hour</b></p> 
<p style="text-align: center;"><b>Sliding Puzzle</b></p>	<p style="text-align: center;"><b>Sokoban</b></p>

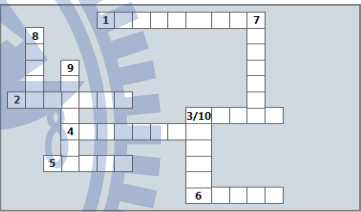


B.3: Fill Out Type

### 3D Logic 2



### Cross word



Check puzzle

**Questions**

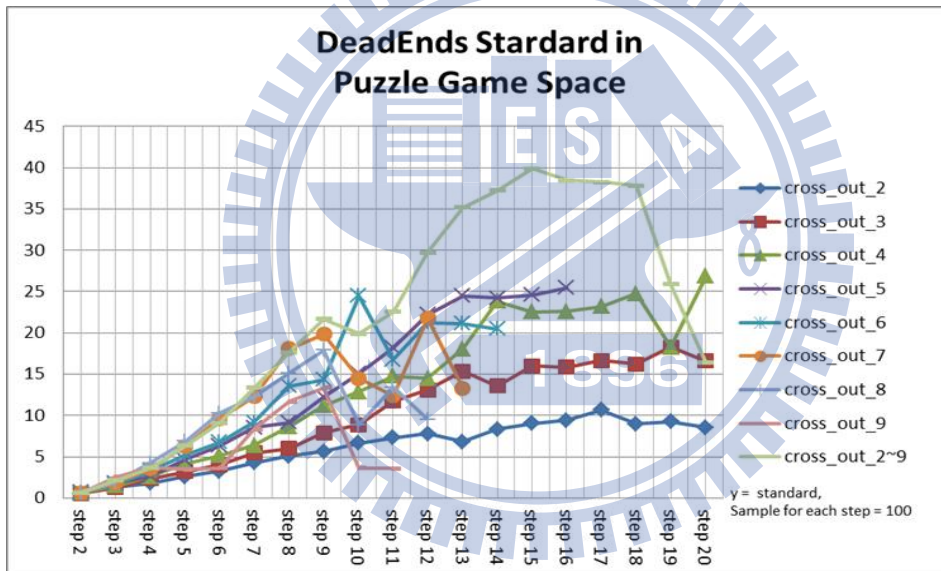
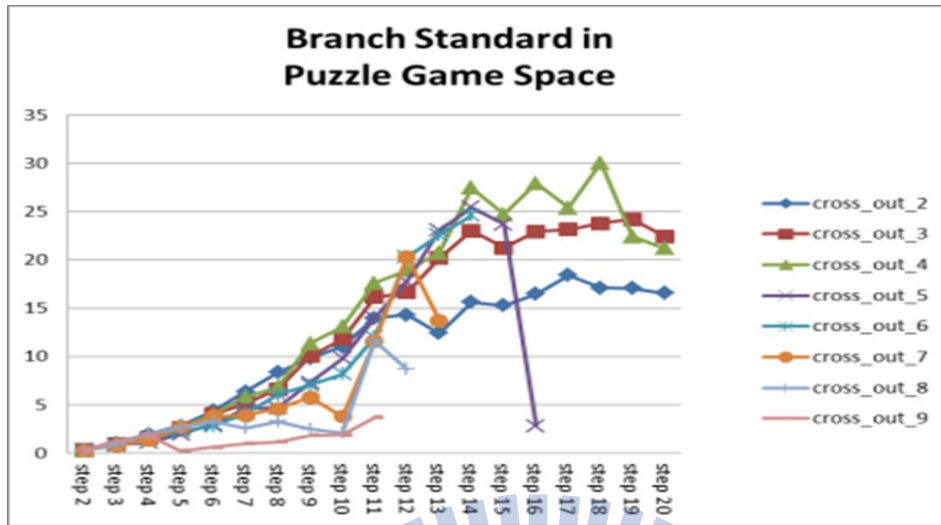
**Across**

- 1. People of Kaerata speak this language
- 2. It is spoken in Karnataka
- 3. It is spoken in Tamlinadu
- 4. The people of Assam speak this language
- 5. It is our national lanhuage
- 6. The language spoken in UP

**Down**

- 7. It is spoken in Maharashtra

# Appendix C: More Result of Experiment One



## Appendix D: Calculate Branch and DeadEnds

```
[C++ code]
void calculateBranchAndDeadEnds( GameState* state){
    GameState* cloneState = state->clone();
    for(int i = 0; i < state->solvedStep; i++){
        Node* node = new Node(cloneState->clone());
        vector<Node*> leaf = expand(node);
        state->nBranch += leaf.size();
        state->nDeadEnd += node->state->nDeadEnd;
        for(vector<Node*>::iterator it2 = leaf.begin(); it2 != leaf.end(); it2++){
            Node* n = *it2;
            delete n;
            n = NULL;
        }
        delete node;
        node = NULL;
        Grid a1 = state->solvedSequence[i][0];
        Grid a2 = state->solvedSequence[i][state->crossOut - 1];
        GameController::getInstance()->operation(cloneState, a1, a2);//cross out
    }
    delete cloneState;
    cloneState = NULL;
}
[C++ code]
```

Implement Branch and Dead Ends calculate function for *Cross Block*.

## Appendix E: Game Data Format

Following data are format example I store that used in my experiment.

E.1: Cross Block

[id]

4782

[cross out]

2

[sizex]

10

[sizey]

10

[game state]

(3,5)(8,5)  
(6,2)(10,2)  
(3,8)(3,10)  
(5,6)(5,10)  
(8,7)(8,10)  
(9,1)(10,1)  
(9,6)(10,6)  
(3,6)(6,6)  
[solved step]  
8  
[nDeadEnd]  
3  
[nBranch]  
72  
[complexity]  
0.0749239

E.2: Sudoku

[id]  
5311b  
[sizex]  
9  
[sizey]  
9  
[game state]  
030902005000400026002000030900004000005080400000500009050000300180007  
000400308060  
[nBranch]  
3894  
[nDeadEnd]  
173  
[complexity]  
0  
[challenge]  
118  
[success challenge]  
30





[time]

18:34

[solve sequence]

1:宮摒餘解--( 8,3 )=3,

2:宮摒餘解--( 3,4 )=8,

3:宮摒餘解--( 5,6 )=9,

4:區塊數對唯餘解--( 7,6 )=1,

5:區塊數對唯餘解--( 7,4 )=2,

6:數對唯餘解--( 8,4 )=6,

7:宮摒餘解--( 9,2 )=2,

8:數對摒除解--( 5,2 )=6,

9:數對摒除解--( 4,3 )=8,

10:單元宮摒餘解--( 7,9 )=8,

11:數對摒除解--( 4,8 )=5,

12:數對摒除解--( 4,7 )=6,

13:數對摒除解--( 8,9 )=4,

14:數對唯餘解--( 8,8 )=9,

15:宮摒餘解--( 8,7 )=2,

16:唯一解--( 8,5 )=5,

17:數對唯餘解--( 9,5 )=9,

18:數對唯餘解--( 7,5 )=4,

19:數對唯餘解--( 9,3 )=7,

20:數對唯餘解--( 7,1 )=6,

21:數對唯餘解--( 9,9 )=1,

22:唯一解--( 7,3 )=9,

23:唯一解--( 7,8 )=7,

24:數對唯餘解--( 5,8 )=1,

25:數對唯餘解--( 5,4 )=7,

26:數對唯餘解--( 4,4 )=1,

27:數對唯餘解--( 4,2 )=7,

28:唯一解--( 9,7 )=5,

29:宮摒餘解--( 1,3 )=6,

30:宮摒餘解--( 6,7 )=7,

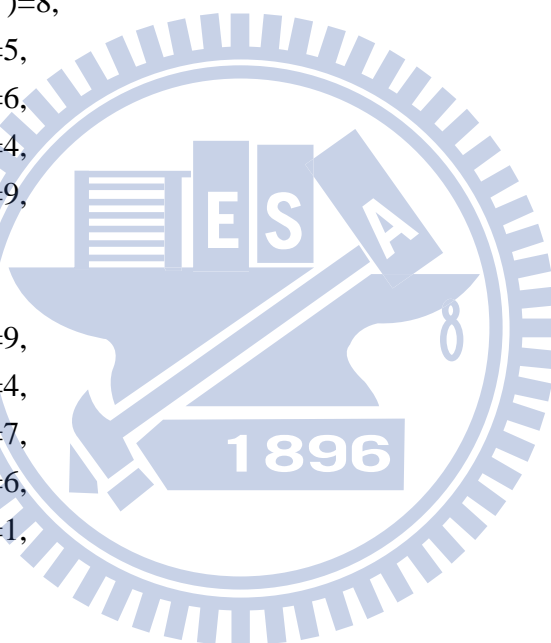
31:宮摒餘解--( 3,9 )=7,

32:宮摒餘解--( 6,8 )=8,

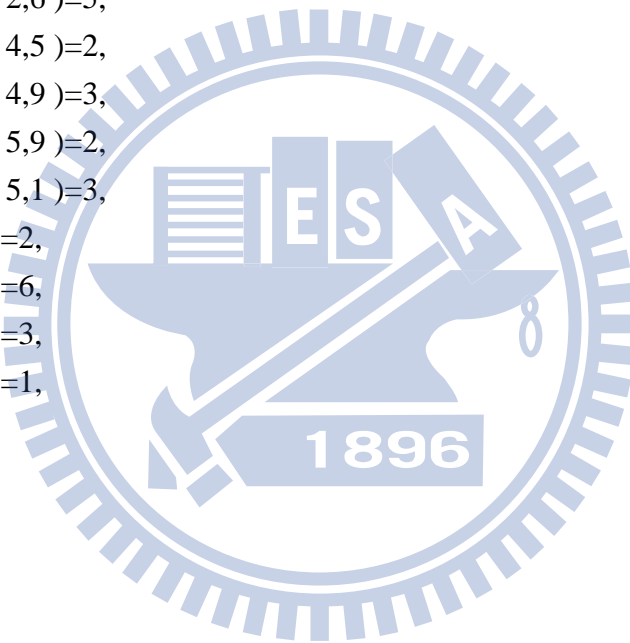
33:唯一解--( 1,8 )=4,

34:宮摒餘解--( 3,2 )=4,

35:宮摒餘解--( 6,3 )=4,

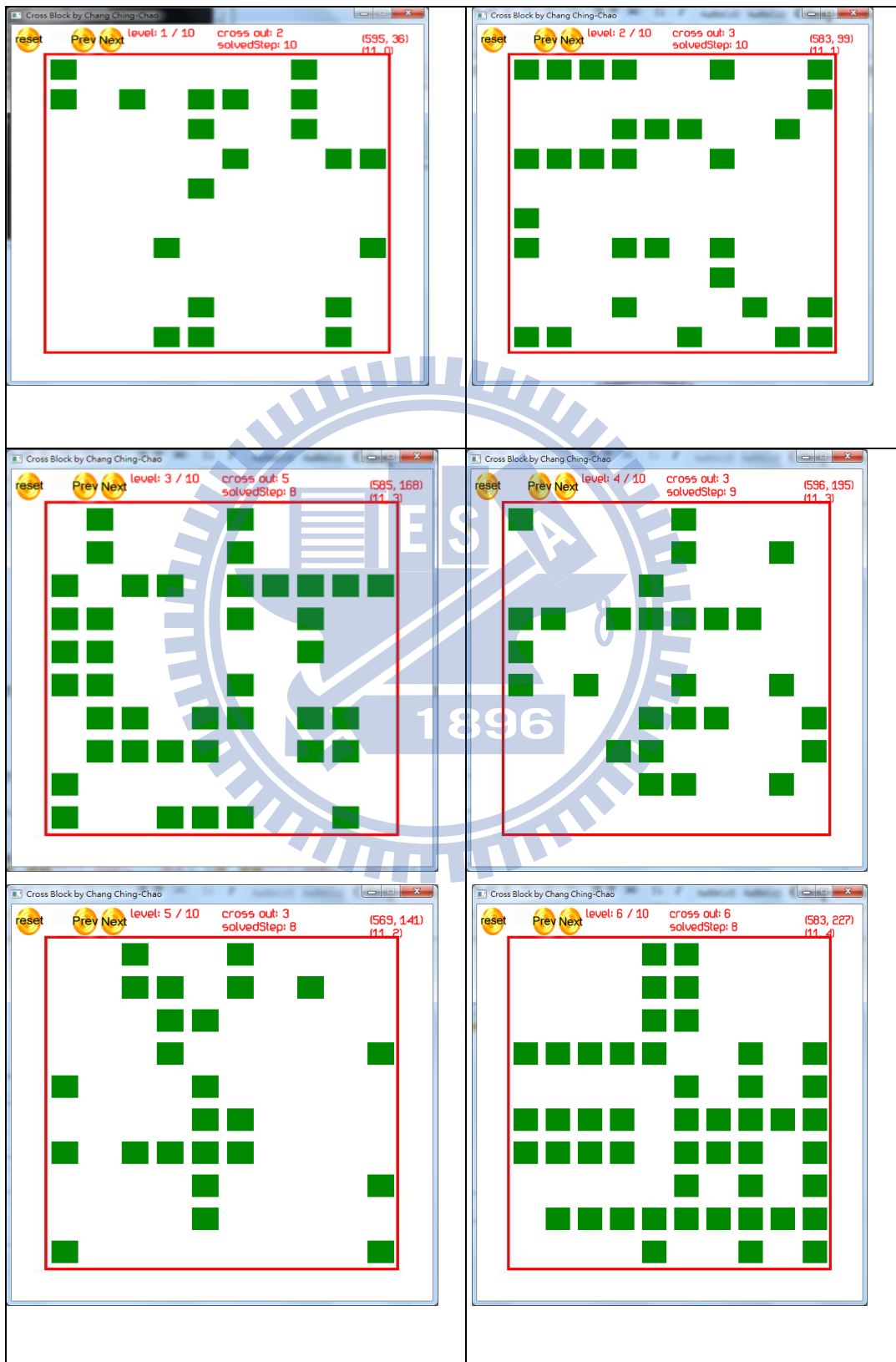


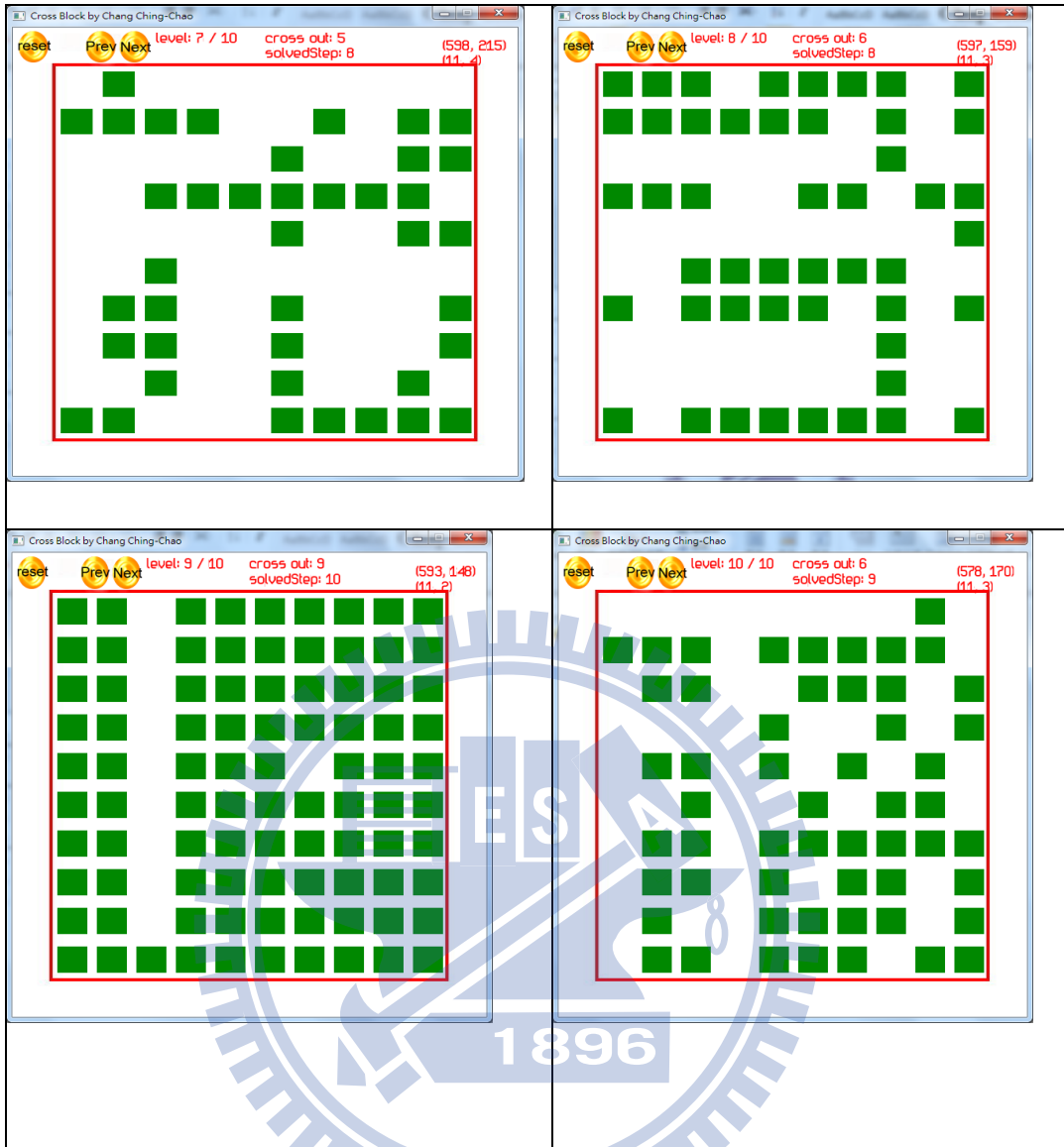
- 36:唯一解--( 2,3)=1,  
 37:宮攝餘解--( 6,2)=1,  
 38:唯一解--( 2,2)=9,  
 39:宮攝餘解--( 3,7)=9,  
 40:宮攝餘解--( 1,7)=1,  
 41:唯一解--( 2,7)=8,  
 42:宮攝餘解--( 1,1)=8,  
 43:唯一解--( 1,5)=7,  
 44:宮攝餘解--( 2,1)=7,  
 45:唯一解--( 3,1)=5,  
 46:數對唯餘解--( 3,6)=6,  
 47:數對唯餘解--( 6,6)=3,  
 48:數對唯餘解--( 2,6)=5,  
 49:數對唯餘解--( 4,5)=2,  
 50:數對唯餘解--( 4,9)=3,  
 51:數對唯餘解--( 5,9)=2,  
 52:數對唯餘解--( 5,1)=3,  
 53:唯一解--( 6,1)=2,  
 54:唯一解--( 6,5)=6,  
 55:唯一解--( 2,5)=3,  
 56:唯一解--( 3,5)=1,  
 [/solve sequence]



# Appendix F: Puzzles in Experiment Two

Following puzzle had used in experiment two:





# Reference

- Adamatzky, Andrew. (2010). *Game of Life Cellular Automata*: Springer-Verlag New York Inc.
- Amanita Design (Producer). (2009). machinarium. Retrieved from <http://machinarium.net/demo/>
- Apple. App Store Retrieved Jun 23, 2011, from <http://app-store.appspot.com/?url=viewGrouping%3Fmt%3D8%26id%3D25204%26ign-mscache%3D1>
- B. R. Clarke. (1994). *Puzzles for Pleasure*. Cambridge, England: Cambridge University Press
- Ben Weber. (2010). Infinite Mario with dynamic difficulty adjustment Retrieved April, 2011, from [http://users.soe.ucsc.edu/~bweber/dokuwiki/doku.php?id=infinite\\_adaptive\\_mario](http://users.soe.ucsc.edu/~bweber/dokuwiki/doku.php?id=infinite_adaptive_mario), <http://www.youtube.com/watch?v=kYbKNAmZ1z4>
- Bernard suits. (2005). *The Grasshopper: Games, Life and Utopia*: Broadview Press.
- C. Crawford. (1984). *Art of Computer Game Design*. New York: McGraw-Hill/Osborne Media.
- C. D. Güss, E. Glencross, Ma. T. Tuason, L. Summerlin, & F. D. Richard. (2004). *Task Complexity and Difficulty in Two Computer-Simulated Problems: Cross-cultural Similarities and Differences*. Paper presented at the Proc. 26th Annual Conf. Cognitive Science Society, Mahwah.
- C. E. Shannon. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 379-423, 623-656.
- C. Pedersen, J. Togelius, & G. N. Yannakakis. (2010). Modeling Player Experience for Content Creation. *IEEE Trans. Computational Intelligence and AI in Games*, 2(1), 54-67.
- Christopher G. Langton (Ed.). (1995). *Artificial Life: An Overview*: Cambridge: MIT Press.
- DJ Trousdale (Producer). (2009). Cross block. Retrieved from <http://ditrousdale.com/games/crossblock/>
- Elina M.I. Koivisto. (2006). *Mobile Games 2010*. Paper presented at the CyberGames '06: Proceedings of the 2006 international conference on Game research and development
- Erich Gamma, Richard Helm , Ralph Johnson, & John M. Vlissides. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley.

- Gerald M. Edelman, & Joseph A. Gally. (2001). *Degeneracy and complexity in biological systems*. Paper presented at the Proceedings of the National Academy of Sciences of the United States of America.
- H. Robin, & C. Vernell. (2004). *AI for dynamic difficulty adjustment in games*. Paper presented at the Proc. of the Challenges in Game AI Workshop, Nineteenth National Conf. on Artificial Intelligence, San Jose.
- Holling, C. S. (2001). Understanding the Complexity of Economic, Ecological, and Social Systems. *Ecosystems*, 4(5), 390-405. doi: 10.1007/s10021-001-0101-5
- J. Kim. (2005). *Task Difficulty in Information Searching Behavior: Expected Difficulty and Experienced Difficulty*. Paper presented at the Proc. 5th ACM/IEEE-CS Joint Conf., New York.
- James Paul Gee. (2005). *Why Video Games Are Good for Your Soul: Common Ground*
- Jeremy Campbell. (1982). *Grammatical Man: Information, Entropy, Language, and Life* New York: Simon & Schuster.
- Johan Huizinga. (1954). *Homo Ludens—Study of the play-element in culture*.
- John H. Holland. (1999). *Emergence: From Chaos to Order*: Basic Books
- Kai-Ju Chen, & Kou-Yuan Huang. (2007). Simulated Annealing for Pattern Detection and Seismic Application. *Proceedings of international Joint Conference on Neural Networks*.
- Kang. Yilin, & Tan. Ah-Hwee. (2010). *Learning Personal Agents with Adaptive Player Modeling in Virtual Worlds*. Paper presented at the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology.
- Katie Salen, & Eric Zimmerman. (2003). *Rules of Play: Game Design Fundamentals*. Cambridge: MIT Press.
- Kou-Yuan Huang, & Ying-Liang Chou. (2008). Simulated annealing for hierarchical pattern detection and seismic application. *International Joint Conference on Neural Networks(IJCNN 2008)*.
- Lennart E. Nacke, Anders Drachen, Kai Kuikkaniemi, Joerg Niesenhaus, Hannu J. Korhonen, Wouter M. van den Hoogen, . . . Yvonne A. W. de Kort. (2009). Playability and Player Experience. *Breaking New Ground: Innovation in Games, Play, Practice and Theory. Proceedings of DiGRA 2009*.
- Lennart Nacke, & Craig A. Lindley. (2008). Flow and immersion in first-person shooters measuring the player's gameplay experience. *Proceeding Future Play '08 Proceedings of the 2008 Conference on Future Play: Research, Play, Share*

- M. Csikszentmihalyi. (1998). *Finding Flow: The psychology of engagement with everyday life*. New York: Basic Books
- M. Sipser. (1997). *Introduction to the Theory of Computation*: PWS Publishing.
- Makoto Matsumoto, & Takuji Nishimura. (1998). Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS) - Special issue on uniform random number generation* 8(1).
- Mia Consalvo. (2007). *Cheating: Gaining Advantage in Videogames, Chap 4. Gaining Advantage: How Videogame Players Define and Negotiate cheating*.
- N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, & C. H. Papadimitriou. (1988). The Complexity of Searching a Graph. *Journal of the Association for Computing Machinery*, 35(1), 18-44.
- P. Robinson. (2001). Task Complexity, Task Difficulty, and Task Production: Exploring Interactions in a Componential Framework. *Applied Linguistics*, 22(1), 27-57.
- Penny Sweetser. (2007). *Emergence in Games*: Charles River Media.
- R. A. Hearn. (2006). *Games, puzzles, and computation*. Ph.D. dissertation, Massachusetts Institute of Technology, Massachusetts.
- R. E. Korf, M. Reid, & S. Edelkamp. (2001). Time complexity of iterative-deepening-A\*. *Artificial Intelligence*, 129(1-2).
- Ratan K. Guha, Erin Jonathan Hastings, & Kenneth O. Stanley. (2009). Automatic Content Generation in the Galactic Arms Race Video Game *Computational Intelligence and AI in Games, IEEE Transactions on* 1, 4.
- Robin Wauters. (2011). Boom - Apple's App Store Hits 10 Billion Downloads Retrieved April, 2011, from <http://techcrunch.com/2011/01/22/boom-apples-app-store-hits-10-billion-downloads/>
- Russell. S., & P. Norvig. (2002). *Artificial Intelligence: A Modern Approach Second Edition*: Prentice Hall.
- S. Kim. (2003). The art of puzzle design: The Puzzlemaker's Survival Kit Retrieved May, 2011, from <http://www.scottkim.com/thinkinggames/index.html>
- S. Kim. (2008). The art of puzzle design: Mathematics as a Creative Art Retrieved April, 2011, from <http://www.scottkim.com/thinkinggames/index.html>
- Sanjeev Arora, & Boaz Barak. (2009). *Computational Complexity: A Modern Approach*: Cambridge University Press.
- Sarah Perez. (2011). Mobile App Market: \$25 Billion by 2015 Retrieved April,

2011, from

<http://www.readwriteweb.com/mobile/2011/01/mobile-app-market-25-billion-by-2015.php>

Togelius, J., De Nardi, R., & Lucas, S. M. (2007). Towards automatic personalised content creation in racing games. *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on* (1-5 April 2007), 252 - 259.

W. Kuang-Chen (巫光楨). (2008). Taiwan Sudoku Association Retrieved May, 2011, from <http://sudoku.org.tw/>

Walter J. Savitch. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*.

