

THE PROCEDURAL GENERATION OF INTERESTING SOKOBAN LEVELS

Joshua Taylor, B.S., M.S.

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

May 2015

APPROVED:

Ian Parberry, Major Professor
Robert Akl, Committee Member
Armin R. Mikler, Committee Member
Robert Renka, Committee Member
Barrett R. Bryant, Chair of the
Department of Computer Science
and Engineering

Costas Tsatsoulis, Dean of the College of
Engineering and Interim Dean of
the Toulouse Graduate School

Taylor, Joshua. The Procedural Generation of Interesting Sokoban Levels. Doctor of Philosophy (Computer Science and Engineering), May 2015, 69 pp., 32 tables, 11 figures, references, 46 titles.

As video games continue to become larger, more complex, and more costly to produce, research into methods to make game creation easier and faster becomes more valuable. One such research topic is procedural generation, which allows the computer to assist in the creation of content.

This dissertation presents a new algorithm for the generation of Sokoban levels. Sokoban is a grid-based transport puzzle which is computationally interesting due to being PSPACE-complete. Beyond just generating levels, the question of whether or not the levels created by this algorithm are interesting to human players is explored. A study was carried out comparing player attention while playing hand made levels versus their attention during procedurally generated levels. An auditory Stroop test was used to measure attention without disrupting play.

Copyright 2015

by

Joshua Taylor

ACKNOWLEDGEMENTS

I would like to thank Marcus Hof, David Holland, Evgeny Grigoriev, David W. Skinner, and Rick Sladkey for giving me permission to use their Sokoban levels in my study. I would like to thank Dr. Thomas Parsons, Paeng Angnakoon, and Marvin Powell for their help in designing and analyzing my study. Finally, I would like to thank my wife for putting up with my extended tenure as a student.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS.....	iii
LIST OF TABLES.....	vi
LIST OF FIGURES	viii
CHAPTER 1 INTRODUCTION.....	1
1.1. Procedural Generation	1
1.2. Sokoban	5
1.3. Outline.....	6
CHAPTER 2 A SOKOBAN GENERATOR.....	7
2.1. Wall Templates.....	7
2.2. Goal Placement.....	10
2.3. Box Search.....	11
2.4. Optimizations.....	15
2.5. Run Time.....	17
CHAPTER 3 ONLINE STUDY	20
3.1. Database	20
3.2. Results	21
CHAPTER 4 LAB STUDY.....	23
4.1. Stroop Test	23
4.2. Study Design.....	24
4.3. Study Data.....	26
4.4. Data Transformations	28

CHAPTER 5 ANALYSIS.....	32
5.1. Independent Variables.....	32
5.2. Dependent Variables.....	32
5.3. Covariables	34
5.4. Statistical Models.....	38
5.5. Results	39
5.6. Conclusion.....	42
CHAPTER 6 FUTURE WORK.....	43
6.1. Algorithmic Details	43
6.2. Other Games	43
6.3. Statistics	44
APPENDIX A PSEUDOCODE	45
APPENDIX B PARTICIPANT SURVEY.....	50
APPENDIX C SURVEY RESULTS	55
APPENDIX D IRB	61
REFERENCES.....	66

LIST OF TABLES

	Page
2.1. The average run time to generate levels of varying sizes with two boxes	17
2.2. The average run time to generate levels of varying sizes with three boxes	18
2.3. The average run time to generate 2 x 2 levels with a varying number of boxes ...	18
3.1. The number of users that played a given number of sessions	21
3.2. Statistics collected from the Sokoban game released on Kongregate.....	22
4.1. Pitch and duration of the voice files used in the study.....	24
5.1. Component matrix for the difficulty covariables.....	36
5.2. Component matrix for the Attention Network Test covariables.....	36
5.3. Component matrix for the Sokoban practice covariables.....	37
5.4. Component matrix for the Stroop practice covariables.....	37
5.5. Component matrix for the gaming habits covariables.....	38
5.6. Component matrix for the subjective experience covariables.....	38
5.7. Log reaction time for the generated levels vs. the hand made levels.....	39
5.8. Log-odds of accuracy for the generated levels vs. the hand made levels	40
5.9. Log-odds for a missed prompt for the generated levels vs. the hand made levels	41
C.1. Participants' ages.....	56
C.2. Participants' gender	56
C.3. Participants' ethnicity	56
C.4. Participants' marital status	56
C.5. Highest level of education attained by participants.....	57
C.6. Participants' occupation	57

C.7.	Whether participants had a computer at their job	57
C.8.	Whether participants had a computer at home	57
C.9.	Hours per week participants spent on the computer.....	58
C.10.	Participants' competency in computer use.....	58
C.11.	Hours per week participants spent on various tasks on a computer.....	58
C.12.	Participants' impressions of various computer related activities.....	59
C.13.	Game genres enjoyed by participants	59
C.14.	Participants' dominant hand	59
C.15.	Participants needing corrective eyewear	59
C.16.	Participants diagnosed with some attention disorder.....	60
C.17.	Participants impression of the game	60

LIST OF FIGURES

	Page
1.1. A simple Sokoban level.....	5
2.1. Templates used in the implementation of this algorithm.....	8
2.2. Squares that a box cannot be pushed onto	10
2.3. A level that forces the player through a binary counter	12
2.4. One section from the level in Figure 2.3	12
2.5. A graphical summary of the run time experiments.....	19
4.1. One of the instruction sheets given to the study participants.....	25
4.2. Normal-QQ plots for the users' reaction times	30
5.1. Bean plots summarizing the raw data for the dependent variables.....	33
5.2. Scree plots of the principal components taken from each group	35
B.1. The survey used in the study.....	54

CHAPTER 1

INTRODUCTION

1.1. Procedural Generation

The term procedural generation refers to any method for the algorithmic creation of content, usually for video games, though the term would also cover the creation of content for other media as well. Content refers to any part of a game that is not code, from the graphics to the sound, the text, and even the layout of the levels.

Algorithms exist for the procedural generation of everything from rocks and rock piles (Peytavie et al. [36]), to trees (Stava et al. [41]), clouds (Harris et al. [14]), and entire landscapes (Doran and Parberry [7]). Some algorithms can be applied to the generation of multiple different types of content. For example, Wang tiles can be used for the generation of large, non-repeating textures (Cohen et al. [3]) or, as Wang cubes, for the generation of large models and architectures based on example models (Merrell [26]). In the other direction, almost any type of content can be generated in multiple ways. For example, landscapes can be generated through the use of agent-based methods (Doran and Parberry [7]) or through the simulation of some of the natural processes involved in the creation of real world terrain (Olsen [31]).

High-end video games continue to increase in size and complexity and thus cost. At the other end of the spectrum, independent games created by smaller teams are becoming more popular but often take a long time to develop. Both groups benefit from any new research that allows games to be created faster. The procedural generation of content is one such avenue of research as, once such an algorithm is developed, it can be used to create such content much more quickly than a human could.

1.1.1. A Good Generator

Doran and Parberry [7] defined five criteria that any good procedural generation system should possess. These are *novelty*, *structure*, *interest*, *controllability*, and *speed*.

1.1.1.1. Novelty

Novelty refers to the generator's ability to generate substantially different content with each run. A procedural generator meant to generate trees is more generally useful if it can generate more than one type of tree, and a terrain generator is more generally useful if it can generate deserts and mountains as well as forests and islands. The novelty of such a system must be judged within the scope of the type of content they are meant to generate. The natural range of variability of boulders is less than that of trees or buildings, and that must be accounted for when deciding whether or not a system is producing novel content.

1.1.1.2. Structure

Random numbers are easy to generate, but not very useful on their own. For a generator to have structure, its output must be something more than simply random. For example, to be useful as a tree generator, such a system must generate something that at least resembles a tree. Imposing some structure on even a simple grid of random numbers can make a useful procedural generator (Perlin [34]).

1.1.1.3. Interest

Interest refers to the generator's ability to produce content the end user finds interesting. This partially comes from a combination of novelty and structure. For generators meant to replicate something that exists in the real world, interest also partially comes from the accuracy of their representation. Interest is the most subjective of the five criteria.

1.1.1.4. Controllability

A good procedural generation system should offer the designer using it some control over the output. For example, a designer using a tree generator would want to be able to generate five examples of one type of tree rather than having to sift through many examples of all the types of trees the generator could produce. The type of content being generated has a large impact on how controllable a generator should be, as a rock generator calls for a different degree of control compared to a tree generator. In either case though, more control makes for a better generator. Controllability also encompasses the intuitiveness of

the controls since, for example, a slider that does something the designer does not understand is less useful.

1.1.1.5. Speed

Speed is meant in the subjective sense. A good procedural generator should be “fast enough” to satisfy the demands of the designer. What exactly that means depends on what is being generated and how the generator is being used. For a generator being used purely as a design tool, a run time of several days can be justifiable if that saves several weeks of design time. If the generator is being used to reduce the memory footprint of a game by generating a large amount of content on demand, the run time must be kept as low as possible.

1.1.2. Why Use Procedural Generation

There are four very general use cases for procedural generators: as a design tool, as a level-a-day feature, as a feature integrated into a finished game, and as a method of replacing a large amount of content in memory as the game is running. Each case can arise for any number of reasons. For example, a generator used as a design tool might have been created to produce rough base layers for a designer to build on, to produce several puzzle levels to help a designer learn what makes a good level when no such levels exist yet, or to produce complete levels to help a designer fill in less important areas of a larger map.

In the first case, the generator is used purely by the designer as another tool in their toolbox. In many such cases, the designer would take the results from the generator and work on them further or otherwise not include the results directly in the final game. This is perhaps the most common use of procedural generation but, from an end user’s point of view, the least visible. The primary reason such generators are so commonly used is due to the time savings they can provide. It might take a designer a week to place every tree in a forest level, while a procedural generator could do the same job in only a few minutes. The time saved could be used to improve parts of the game the player is likely to pay more attention to or simply to get the game to release sooner.

In the second case, the generator would be used to implement a level-a-day feature, or something similar. It would be required to generate one complete level within a 24-hour period, or otherwise reduce the workload on the designer to where they could continue to create a new level each day. This is probably the least common use for procedural generation systems as a level-a-day feature is not appropriate for every game.

In the third case, the generator is included as part of the final game. For example, players could access the generator through the game to provide them with a new character model, a new item, or a new level at any time. Since the designer would not be able to influence the results beyond setting the initial parameters for it, the generator would need to produce a complete piece of content that needed no further processing. For a level generator, for example, it would need to run within the same time frame as the average loading screen. With the success of games like Spelunky [40], Terraria [44], and Minecraft [27], this type of generator is currently the most popular and what most people think of when they hear about procedural generation. The idea goes back to much older games like Rogue [46] and NetHack [30]. The random items of Diablo [5] also fall into this category, while the infinite worlds of Minecraft actually fall more into the next.

The fourth case is where a very fast procedural generator is used to replace the content it would generate. For example, instead of storing the entirety of a very large world, the generator would create smaller pieces of it as needed. Since code usually has a much smaller memory footprint than the output it is capable of producing, this allows the designer to fit more content into the game than would otherwise be possible. Alternatively, the designer could use the same techniques to allow the game to run on weaker hardware. Often generators used this way allow designers to explore ideas that would otherwise be impractical.

These cases are not mutually exclusive; if a generator can be used for one category, in principal it can be used for any of the categories above that. For example, many people who make adventure maps for Minecraft use its level generator as a design tool, even though that generator falls into the fourth use case as it is intended to allow for larger worlds than would fit into memory or even on disk.

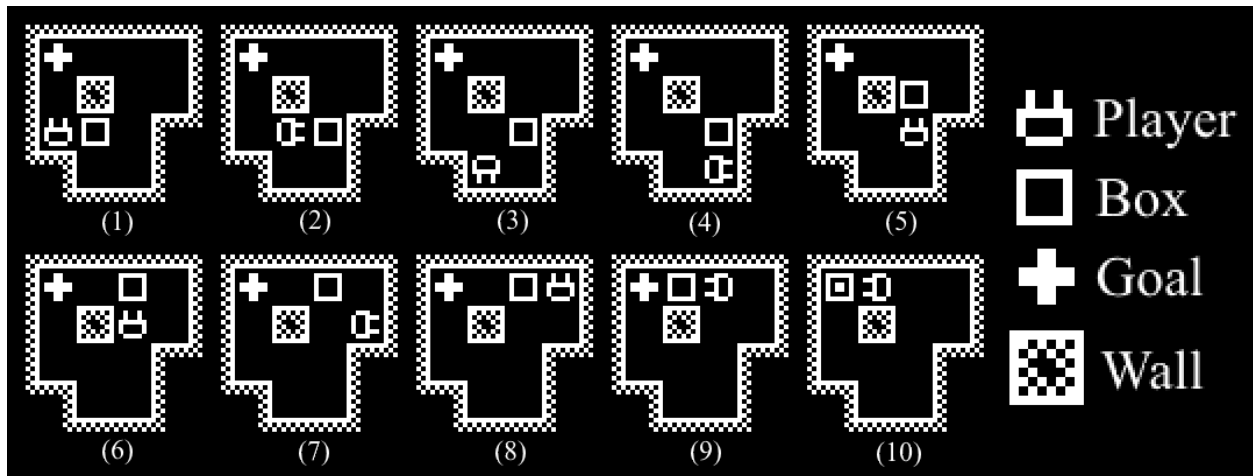


FIGURE 1.1. A simple Sokoban level solvable in 9 steps. This is the only way to solve this level as the player must get behind the box to push it.

1.2. Sokoban

Sokoban is a grid-based transport puzzle with the objective of moving a set of boxes on to a set of marked goal squares. The player controls an avatar within this grid to accomplish this objective. The avatar is limited to pushing the boxes, at most one at a time, and cannot manipulate the walls at all. Figure 1.1 shows a simple Sokoban level and its solution. The rules are very simple and yet the levels created within those rules can be very difficult.

Together, this makes Sokoban a good candidate for study. A random collection of walls, floors, and boxes is very unlikely to form a solvable Sokoban level, and even if it happened to be solvable, it would very likely be too easy to be interesting. Making a difficult, but not impossible, level is a challenge for both human authors and procedural generators.

1.2.1. Other Sokoban Generators and Related Work

Most of the academic interest in Sokoban is in its use as a difficult planning task (Junghanns and Schaeffer [19], Dor and Zwick [6], Jarušek and Pelánek [17]). One paper by Masaru et al. [24] describes a method that takes a large, easy Sokoban level and creates a more difficult level by carefully removing unnecessary floor spaces.

A second paper by Murase et al. [29] uses a simple system of templates laid on top of each other to produce the layout of the walls. They then place the goals and boxes

randomly, avoiding certain squares. Finally, they use a solver to ensure the level is solvable and a heuristic evaluator to make sure the level is not too simple.

Outside of academic papers, there have been a number of Sokoban generators written. Mühendisi [28] has a web page collecting a number of these. Most of these programs do not divulge exactly how they work, though a few are open source. Of particular relevance are SokEvo and YASGen, which use genetic algorithms to place the walls, goals, and boxes and then attempt to solve the generated levels in reverse.

Besides Sokoban, there has been some research on the generation of levels for other hard puzzle games. One paper on the generation of Go problems (Chou et al. [2]) takes the interesting approach of playing two unequal artificial players against each other. This approach would only work in a two player game though. Another paper (Servais [38]) explored the generation of Rush Hour levels through the use of binary decision diagrams. One program of particular interest, simply called Sliding Block Puzzle Solver [43], includes an option to search for the diameter of the state space of a given sliding block puzzle. This gives two end points that together make a complete puzzle. Since all moves in a sliding block puzzle are reversible, either end point can be considered the start or the finish.

1.3. Outline

The rest of this document describes a new approach to the generation of Sokoban levels and a study carried out to determine whether or not it satisfied the criteria of interest in the levels it generated. [Chapter 2](#) describes the generation algorithm in detail. [Chapter 3](#) describes the implementation of a level-a-day feature for an online implementation of a Sokoban game. [Chapter 4](#) describes the design of the study. [Chapter 5](#) contains the analysis of the data from the study. [Chapter 6](#) discusses several questions to be answered in future studies. Additionally, [Appendix A](#) gives a pseudocode listing for the complete algorithm, and [Appendix B](#) and [Appendix C](#) contain a copy of the participant survey and additional statistics collected from it.

CHAPTER 2

A SOKOBAN GENERATOR

The approach for the generation of Sokoban levels presented here can be broken down into three main steps: place the walls, place the goals, and finally, place the boxes and the player. The walls are placed according to a set of interlocking templates. The algorithm then places the goals in each possible configuration and, for each one, a search is carried out to find the best place to start the boxes and the player avatar. Additionally, an optional timer is included to terminate the generation if it exceeds a specified run time. After that time, it will finish the step it is working on and return the best results found so far. [Appendix A](#) gives a pseudocode listing for this algorithm.

2.1. Wall Templates

The first step of the algorithm is to place the walls and define the room within which the boxes and goals will later be placed. It begins with a blank grid of a designer-specified size divided into 3×3 regions. In each region, it tries to place walls and floors according to one of the available templates. Once the algorithm fills in each region, it performs a series of checks to make sure the resulting room meets some basic requirements, such as having enough floor squares, forming a single, contiguous room, and not containing any large, open spaces.

Templates are supplied by the designer and, as such, offer some measure of control over the final shape of the level. Each template consists of a 3×3 grid of walls and floors, surrounded by squares representing restrictions on what may border that template. For example, in [Figure 2.1](#), the template consisting of a narrow hallway between two walls requires that there be at least one more floor space on either end to prevent the creation of narrow dead ends. Anything outside the defined grid is considered to be filled with wall spaces.

It is possible for certain configuration of walls to allow the player's avatar to pass through without allowing any box to pass. If no precautions were taken, it would be possible

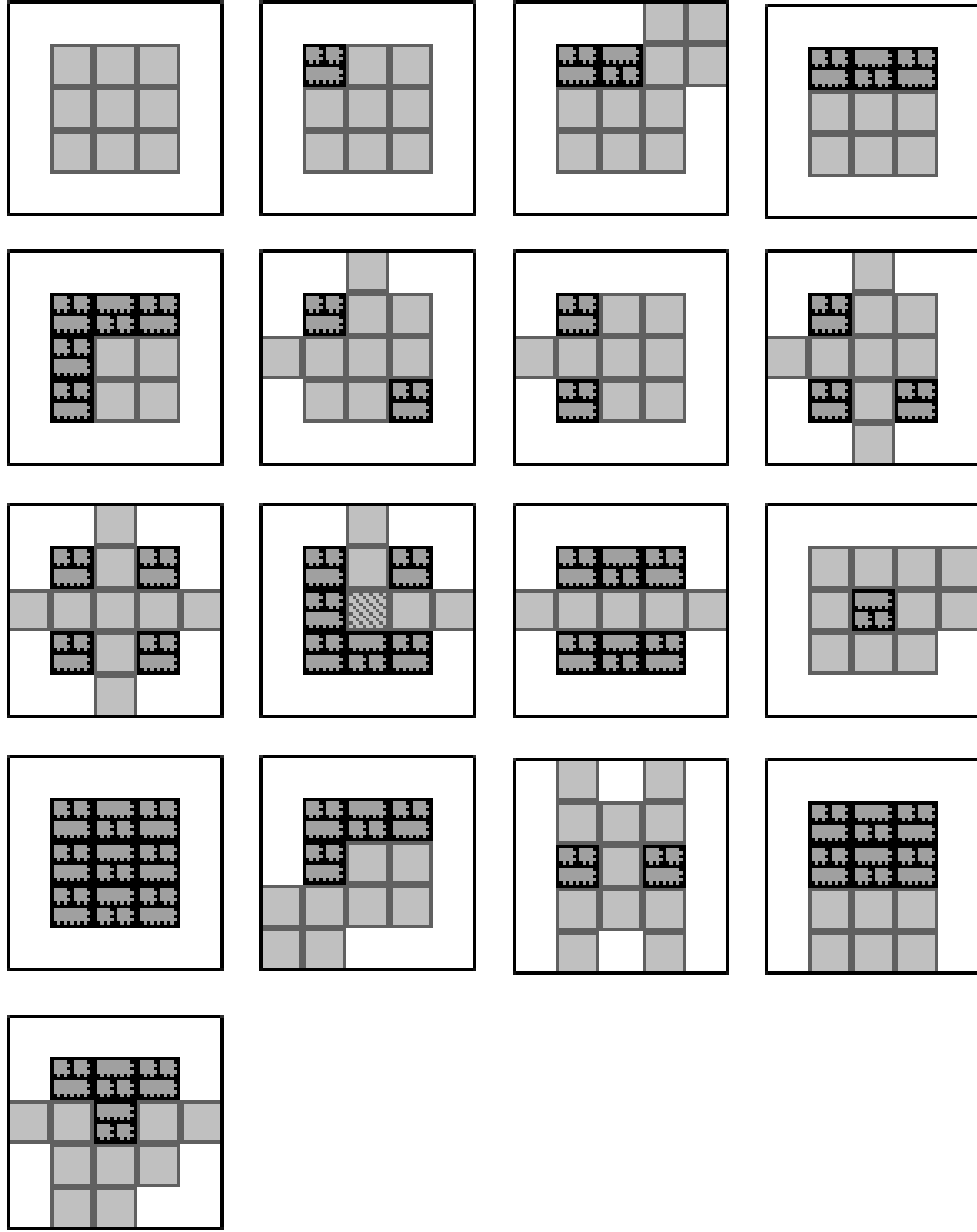


FIGURE 2.1. Templates used in the implementation of this algorithm. The brick pattern represents wall spaces while the gray squares represent floor spaces.

for such a template to divide the level into two separate sub-puzzles. One such template is the narrow hallway corner, which in [Figure 2.1](#) contains a specially marked square. That square is counted as a floor space in the final level, but does not count towards the contiguity

of the remaining floors. This allows the boxes to be pushed around that template so the level is not divided.

2.1.1. Template Placement

Templates are placed by a simple guess-and-check method. The algorithm picks a random template, rotates and reflects it randomly, and then attempts to place it in a random unused region. If this fails, due to the border restrictions, it picks a new template, a new rotation, and a new region. If it fails to place a template too many times, the algorithm starts over with a new, blank grid. While this is not the most efficient way to apply these templates, the running time of this step is insignificant compared to the remaining steps; therefore, optimizing this part of the code would not be beneficial.

2.1.2. Restrictions

The final checks are designed to ensure that the resulting room is compatible with the remaining steps of the algorithm. In a minor way, they also help with the aesthetics of the resulting puzzle. The first check is to make sure there are sufficient floor spaces in the room to fit the target number of boxes. The absolute minimum needed would be one floor space per box plus one for the player's avatar. The player would be unable to move in such a room though, so a target of three times the number of requested boxes was chosen.

The second check is to make sure the floor spaces form one contiguous room. Since neither the player's avatar nor the boxes can move through the walls, any cutoff areas would be wasted space. Additionally, it is possible to have areas that the player can move through but boxes cannot. To avoid accidentally dividing the level into separate, non-interacting sub-puzzles, such areas should not count towards the contiguity of the remaining floor space.

Large open spaces allow boxes to be pushed around one another, allowing multiple routes to the same end state. This causes the search space to increase in width but not in depth. At the same time, human players can recognize that any of those paths would lead to the same results. Therefore, such spaces make it harder for the computer to generate the level without making it more difficult for the player to solve. The size of the area needed for

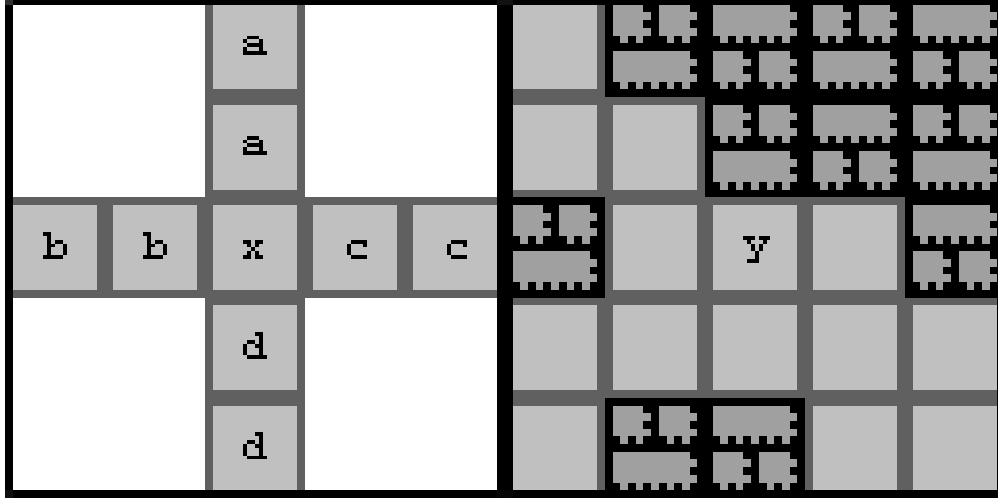


FIGURE 2.2. To be able to push a box onto square x , either box a squares, both b squares, both c squares, or both d squares must be floor spaces. For example, no box can be pushed on to square y .

this to become an issue depends on the number of boxes used. With more boxes, they would begin to get in each other's way, making it less likely there would be enough space in such an area to push them past one another. For a target of up to seven boxes, rooms containing a 3×4 rectangle of open space were forbidden.

Finally, rooms containing a floor space bordered on three sides by walls were rejected. Such spaces are rare in hand made levels as they tend to make the resulting level easier. If there is a goal in such a space, it is easy to move a box into that space after which it no longer interacts with the rest of the level, leaving more room in which to manipulate the remaining boxes. If there is not a goal in such a space, it is obvious that no box will ever need to be moved there, as there would be no way to remove it later.

2.2. Goal Placement

Once the walls are placed, the second step in the algorithm is to place the goals. The number of goals to be placed is set by the designer. The algorithm tries every combination of potential goal squares. Floor squares that a box could not be pushed onto are eliminated

from consideration. These are squares that do not have at least two colinear floor spaces to at least one side (see [Figure 2.2](#)).

To allow the generator to terminate early while still presenting a reasonable result, the configuration space is explored in a random order. This is accomplished by placing all the floor spaces into a list and then shuffling it. The goals are placed in order according to this list.

2.3. Box Search

For each arrangement of goal spaces, the algorithm performs a search for the starting state farthest from the ending state implied by the location of the goal spaces. What that state is depends on which distance metric is used.

2.3.1. Distance in Sokoban

There are four common ways of measuring distance in Sokoban: player moves, box pushes, box lines, and box changes (often abbreviated to moves, pushes, lines, and changes). A move is the simplest and is counted any time the player's avatar changes position. A push is counted whenever a box changes position.

A box line is counted whenever the player changes a box's position, but multiple consecutive pushes of the same box in the same direction only count as a single line. For example, pushing a box down a long hallway would count as one line, while pushing it around a corner would count as two. Pushing one box, then another, and then the first again would count as a total of three lines even if the first box was pushed further in the same direction.

A change is counted whenever the player begins pushing a different box, including the first box pushed in a level. For example, pushing a single box all the way around a maze would only count as one box change. Pushing one box, then another, and then the first again would count as three box changes regardless of how far or in which directions each box was pushed. By these definitions, any solution for a level will always require at least as many moves as pushes, at least as many pushes as lines, and at least as many lines as changes.

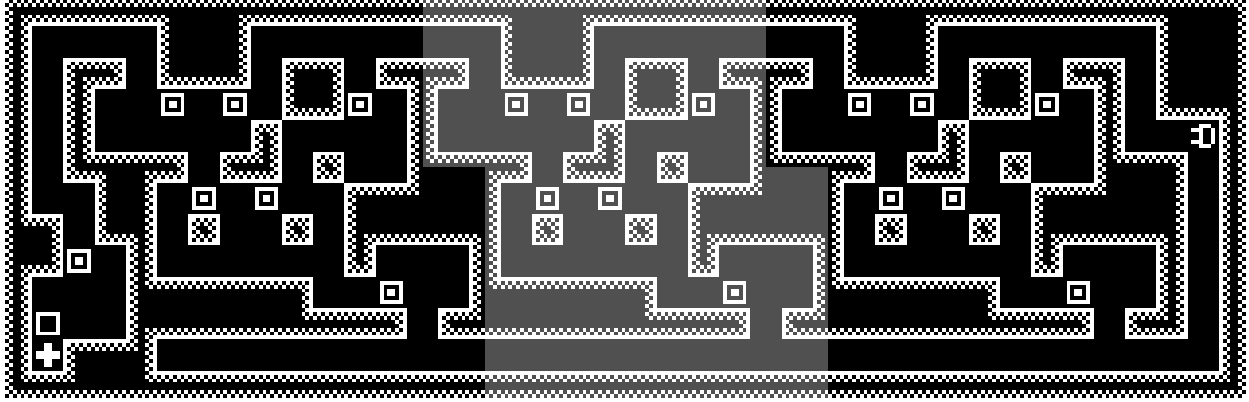


FIGURE 2.3. A level that forces the player through a binary counter. Each repetition of the highlighted section approximately doubles the length of the solution.

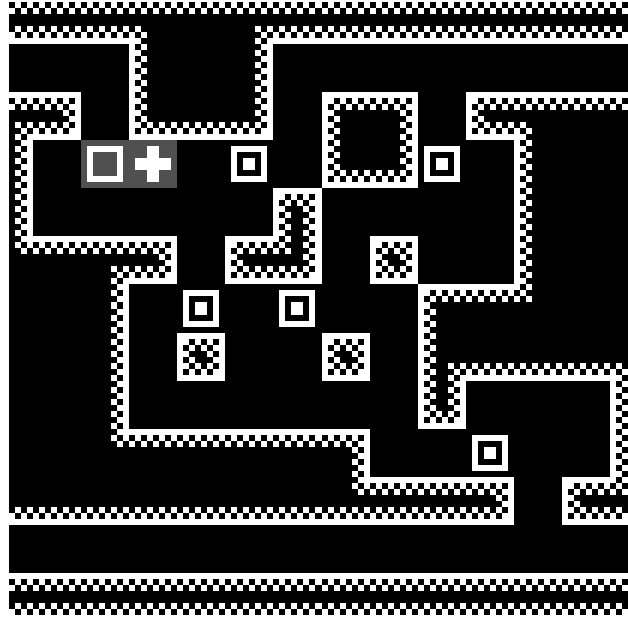


FIGURE 2.4. One section from the level in [Figure 2.3](#). The highlighted box stores one bit of the counter.

There is some research that suggests that, out of these four metrics, box changes most closely approximates the human-judged difficulty of a level ([Jarušek and Pelánek \[17, 16\]](#)). For computational reasons, the algorithm presented here uses box lines as the distance metric.

2.3.2. Longest Shortest Path

In graph theoretic terms, the distance to the farthest state from a given state is called the eccentricity of that state. The greatest eccentricity of any state is called the diameter of the graph. Computing the diameter of a graph can be done with an all-pairs shortest path algorithm, such as the Floyd-Warshall algorithm (Floyd [11]). However, a Sokoban level is a succinct representation of its state space graph. A succinct representation of a graph (Galperin and Wigderson [12]) is one where the size representation is a polylogarithmic function of the number of nodes in the graph it represents.

For Sokoban, this can be seen by noting the existence of a series of levels where the length of the solution grows exponentially with a polynomial increase in the size of the level. One example of such a level is given in Figure 2.3; for each replication of the highlighted area, the length of the solution is approximately doubled regardless of the distance metric used. Figure 2.4 shows one section of the level. To progress through each section, the player must enter from the top-right, travel downward, push the highlighted box off its goal, and then exit into the bottom tunnel. Once the box has been moved, the player can reenter the top-right of the section and exit through the top-left, moving the highlighted box back onto its goal in the process.

Once the player enters the final section, they cannot leave, which means that all other boxes must be on their goal spaces before the player can enter that section. Any other path will leave the level in an unwinnable state. To get to the entrance of a section other than the first, the player must go through the previous section twice, which means that in a level with n sections, the player will go through the first section 2^n times. These sections are adapted from the devices shown in Culberson [4], who used them to prove that the question “Is there a solution to a given Sokoban level?” is a PSPACE-complete decision problem.

2.3.2.1. PSPACE

PSPACE is the computational complexity class defined as all decision problems that can be answered with a correct “yes” or “no” using an amount of space that is a polynomial function of the size of the input regardless of run time. For PSPACE, there is no difference

between the use of a deterministic Turing machine and the use of a nondeterministic Turing machine (Hopcroft and Ullman [15]). In contrast, P is the class of all problems that can be answered in an amount of time that is a polynomial function of the size of the input using a deterministic Turing machine. It remains an open question whether the use of a deterministic Turing machine or a nondeterministic Turing machine (NP problems) has a categorical difference on the running time of such algorithms. It is also an open question as to whether or not any algorithm that can be run in polynomial space can also be run in polynomial time (Garey and Johnson [13]).

A decision problem that is part of a complexity class and can be used to solve all other problems in that class with only a polynomial increase in running time is called complete for that class. An example of a P-complete problem is whether or not a given string is a member of a given context-free grammar, and an example of an NP-complete problem is whether or not there exists a path through every node of a given graph that does not revisit any nodes. Many two-player games, such as Chess and Go, are PSPACE-complete due to the nature of turn-taking: The process of deciding if there is some winning move player one can make, regardless of any move player two could respond with, is analogous to the operation of an alternating Turing machine. A nondeterministic Turing machine can accept an input (return a “yes” answer) when there is at least one series of choices that leads to an accepting state (an NP problem) or when all series of choices leads to an accepting state (a co-NP problem). An alternating Turing machine alternates between these two modes of computation. PSPACE is equivalent to the set of problems that can be solved in polynomial time by an alternating Turing machine.

2.3.3. Farthest State

To find the most distant state, the algorithm begins from the ending state implied by the goal placement. Since the boxes are interchangeable, and every box must end on exactly one goal, this state is unique except for the final position of the player’s avatar. The algorithm generates each possible position of the player and then searches backwards through the state space to find the farthest starting state. By running the game backwards, every

state generated is guaranteed to be a valid candidate for the starting state. That is, no matter where the generator stops, the result will be a playable level.

To determine when the farthest state has been found, the algorithm performs a form of iterative deepening search. A standard breadth-first or iterative deepening search would not work on a desktop computer due to the limits of its memory. However, since not all moves are reversible, it is not sufficient to store only the frontier of the search in memory; a move could be generated that revisited a node last seen any number of moves prior. To prevent this, a search on a standard graph would simply keep track of every node that it visited. Since Sokoban is PSPACE-complete, the number of nodes that would need to be tracked would quickly grow beyond any reasonable bounds.

To avoid these problems, the algorithm presented here proceeds by layers, advancing the frontier of the search by one step in each layer. After each step, it begins a new search from the original set of ending states, and in the n th layer, advancing it $n - 1$ steps, removing any duplicates found from the main search.

This process is slower than a breadth-first search and is more comparable to an iterative deepening search. The advantage is that it uses much less memory than would be required to keep all visited nodes in memory. Since the secondary search does not need to recognize when it is revisiting nodes, the algorithm only needs to store a total of three frontiers in memory at any given time: the frontier of the main search, the frontier of the secondary search, and a workspace to advance one of the other two frontiers. Considering that some levels can take 50 or more box lines to solve, this is a huge savings.

2.4. Optimizations

The algorithm as presented in [Appendix A](#) includes two optimizations and one short-cut over what is described above. The first and most important optimization is to abstract out the player’s exact position. Because any move that does not change the position of the boxes can easily be undone, the player’s exact position is unimportant. All that matters is which boxes and which sides of those boxes can be reached from the player’s position. Calculating everywhere a player can reach without moving a box can be done by a simple

flood fill algorithm. As such, the algorithm only tracks which contiguous region of the floor the player is currently in. Since almost any level will require more moves than any other measure of solution length, this reduces number of states that need to be considered, often significantly. Additionally, it requires only a minor amount of extra work to calculate all the states reachable in one line over that required for one push. Instead of moving each box one space, the algorithm moves each box as far as possible in each direction, generating each intermediate state.

The other optimization is to mark certain squares as “slippery.” If the player pushes a box onto these squares, it will make no difference to the game state even if they continue to push the box farther in the same direction. This occurs in narrow hallways with no goal spaces or other boxes (specifically, when there are empty floors on two opposite sides of a square and walls on the remaining six spaces, including the four corners).

When the player pushes a box onto a slippery square, their only options to get the box out of that space are to push it all the way through the hallway, or to go around to the other side of the box and push it back out the way it came in. While one box is in such a hallway, no other box can be pushed in without putting the level in an unwinnable state. Additionally, the box in the hallway is not in the way of any other boxes in the level. Therefore, any box pushed part way into such a hallway may be pushed all the way to the other end without affecting the game state. The algorithm generates each move along each line, but any moves that leave a box on a slippery square are not added to the search space.

Finally, the algorithm takes one major shortcut: Instead of generating all the requested goals in one pass, it finds the optimal placement of just two or three. Then, with those fixed in place, it finds the optimal placement of another one or two. It repeats this until the requested total has been generated. This speeds up the level generation in two ways. First, it generates fewer configurations of goals. For example, placing six goals in a room with 20 floor spaces would require checking up to $\binom{20}{6} = 38,760$ configurations. Placing them two at a time would instead only require checking $\binom{20}{2} + \binom{18}{2} + \binom{16}{2} = 463$ configurations. In

Size	Moves	Time
1×2	26	< 1 sec
2×2	48	1.9 sec
2×3	60	16 sec
3×3	73	128 sec

TABLE 2.1. The running time to generate levels of varying sizes with two boxes. Each data point is the average of 10 runs.

addition to checking fewer ways of placing the goals, it also spends less time checking each configuration as there are fewer boxes to solve for in all but the final pass.

2.5. Run Time

If the final state is n steps away, the algorithm presented here will generate the m th layer $n - m + 1$ times. Since there are fewer states in the earlier layers, generating these positions multiple times is not overly expensive. Theoretically, given s floor spaces and b boxes, there are up to $\binom{s}{b}$ ways of placing the goals, up to $\binom{s}{b}$ ways of placing the boxes, and up to $3b + 1$ ways of placing the player. Each goal and each box can be placed on any square, although not all combinations will result in a playable level. The player can be placed in any contiguous section of the remaining floor space. A single box can divide the floor into at most four sections; for example, by placing it at the intersection of two narrow hallways. Each additional box can divide one of those regions into at most four subregions, creating three additional regions. This gives an upper bound on the number of states that might be searched: $O(b\binom{s}{b}^2)$.

Table 2.1, Table 2.2, and Table 2.3 show the results of several timing experiments using this algorithm. Each experiment was executed on an Intel Core i7 3.2 GHz quad-core processor with hyperthreading. The implementation used did not take advantage of the extra cores, but several independent copies of the code were run simultaneously, relying on the operating system to place each copy on a separate core. For each data point, 10 levels of the

Size	Moves	Time
1×2	38	58 sec
2×2	69	2.7 min
2×3	98	1.1 hr
3×3	115	24.5 hr

TABLE 2.2. The running time to generate levels of varying sizes with three boxes. Each data point is the average of 10 runs.

Boxes	Moves	Time
2	48	1.9 sec
3	69	2.7 min
4	100	3.4 hr
5	109	26 hr

TABLE 2.3. The running time to generate 2×2 levels with a varying number of boxes. Each data point is the average of 10 runs. All of the boxes were placed in one pass.

specified size and number of boxes were generated and the run times were averaged together. The size column in the tables refers to the number of template pieces used. The average number of player moves needed to solve each level set was also recorded for comparison.

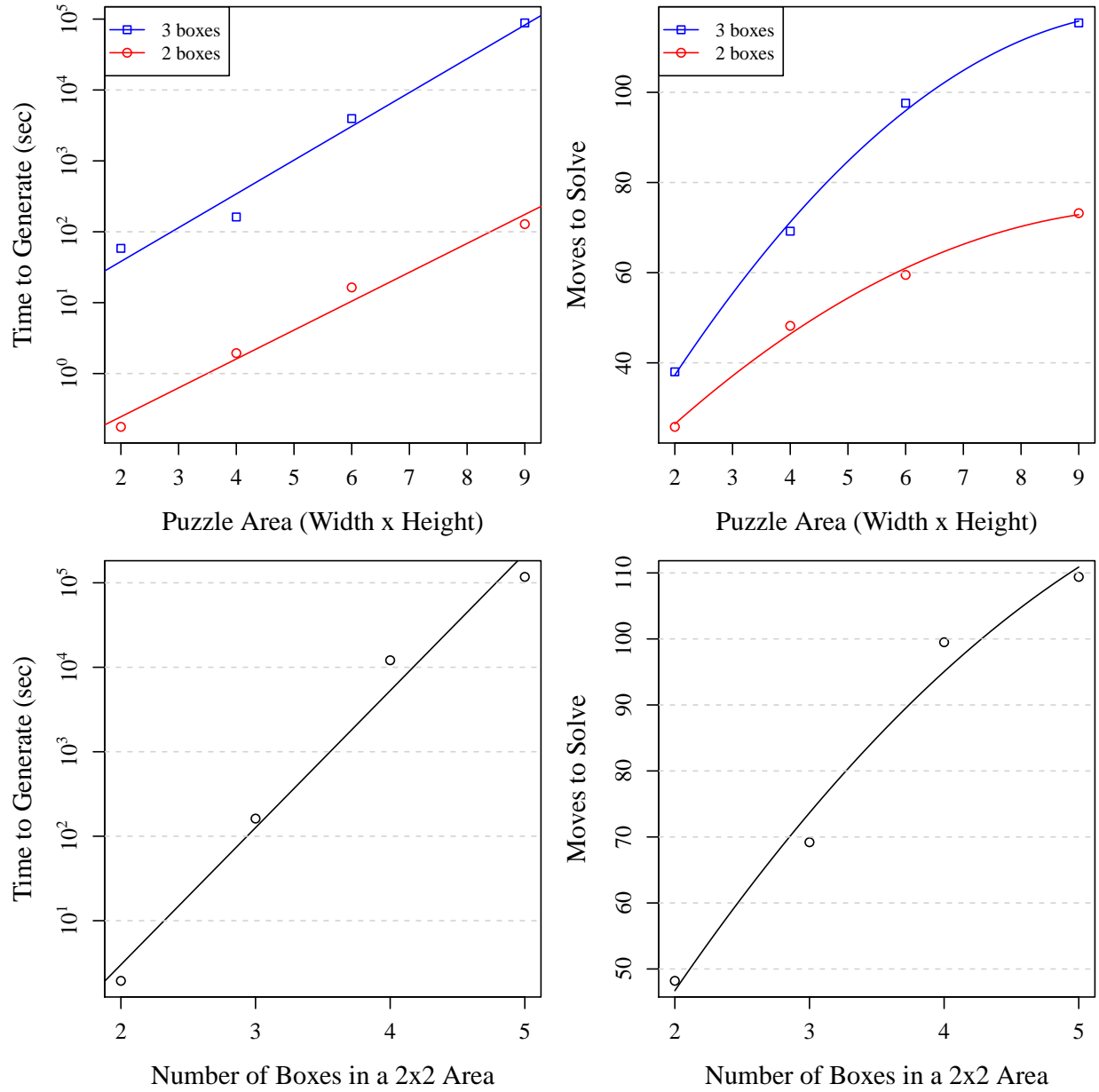


FIGURE 2.5. A graphical summary of the run time experiments. The graphs to the left show the average time needed to generate various levels along with best-fit exponential lines. The graphs to the right show the average number of moves needed to solve the generated levels along with best-fit quadratic lines.

CHAPTER 3

ONLINE STUDY

To test if the algorithm presented in [Chapter 2](#) and [Appendix A](#) was fast enough to support a level-a-day feature, a C++ implementation was written. It was run automatically once per day on a cloud server and was set to generate as many levels as it could within a 23-hour period. The size of the level to be generated was chosen randomly at the beginning of each day and was either a 2×3 or a 3×3 level using four to six boxes, a 2×2 or a 2×3 level using five to seven boxes, or a 2×2 level using six to eight boxes. An ActionScript implementation of Sokoban was written to provide access to these generated levels. This game was made available on the Flash portal Kongregate [22].

In addition to the level-a-day feature, the game included several static level sets. To provide a comparison to the procedurally generated levels, levels from five expert authors were collected, with their permission. A program was used to take the large collection of levels from each author (including a collection by the generator) and categorize them by their size, number of boxes, and number of moves needed to solve them. It then randomly selected 25 levels of a specified size and complexity from each collection (for a total of 150 levels) for inclusion in the game. Additionally, 25 smaller levels were generated to make an easy set as an introduction to Sokoban.

3.1. Database

The generator connected to a MySQL database running on the same cloud server. It used this database to store the levels it generated and to ensure that duplicate levels were rejected. A PHP page was created that accepted a level number and returned the corresponding level from the database. The game would connect to this page to give users access to the generated levels to provide the level-a-day feature.

The game also implemented an event logger that recorded data from the users' play sessions. One PHP page was created that took the user name provided by Kongregate and returned a session key. When a new session was started, the session key was recorded into the

database along with a timestamp and a hash of the user name. This was enough to register whether a user played the game multiple times while maintaining the users' anonymity. Another page recorded a session key and an event from the game's event logger. An event log was recorded whenever the user started, restarted, won, or quit a level. Along with each event, the number of moves and pushes up to that point was recorded.

3.2. Results

The first level the algorithm generated was created on March 30th, 2011. As of February 20th, 2015, a total of 13,377 levels were created, averaging just over nine levels per day. Since the game's release on Kongregate on September 11, 2012, the game had gathered approximately 1,200 plays according to their metrics. A play refers to any time a user loaded the page containing the game. Out of those approximately 1,200 plays, there were 328 sessions played by 258 users, where a session implies that the player pressed the start button on the screen informing them that the game would collect anonymous data. Out of the 258 users, 24 attempted levels from at least two different level sets, and of those seven attempted levels from more than two.

Sessions	1	2	3	4	6	8	12
Users	224	18	10	3	1	1	1

TABLE 3.1. The number of users that played a given number of sessions. 224 players played only a single session while 34 played at least two.

There were a total of seven level sets presented to the player. Each level set was by a different author with the exception of level set 4 and the easy set, which were both generated by the algorithm presented here using different settings. The easy set was always in the top-right corner of the selection screen with the other six sets presented below that in a random order. This order was generated based on the player's user name so that they would see the same order if they reloaded the game.

		Users	Attempted	Restarted	Solved	Quit	Time
Procedurally	Easy	200	1561	2551	1490	168	113,763 s
Generated	4	5	16	20	11	3	2,254 s
Hand Made	1	35	49	92	5	28	2,609 s
	2	5	11	29	6	2	2,324 s
	3	4	7	9	1	4	446 s
	5	5	14	17	7	8	898 s
	6	34	40	14	4	8	583 s

TABLE 3.2. Statistics collected from the Sokoban game released on Kongregate. Each column is a total over all sessions.

Table 3.1 shows the number of sessions per user as recorded by the game. The majority of players only played a single session. Table 3.2 shows statistics gathered from the play data. The *Users* column refers to the number of players that started at least one level within that set. *Attempted* shows the total number of levels attempted by the players. *Restarted* shows the total number of times the players pressed the restart button to get back to the beginning of a level. *Solved* shows the total number of levels solved by the players. *Quit* shows the total number of times the players quit a level; however, this only counts the times they quit via the keyboard. If they closed the window the game was running in, it would not have recorded that action. *Time* shows the total number of seconds spent on levels within each set.

CHAPTER 4

LAB STUDY

To determine if the levels generated by the algorithm presented in [Chapter 2](#) and [Appendix A](#) were interesting, an experiment was conducted with student volunteers from the Computer Science and the Psychology departments at the University of North Texas. The ActionScript game mentioned in [Chapter 3](#) was modified to record the players attention while playing specific Sokoban levels. The hypothesis was that there would be no difference between the players' attention levels while playing the procedurally generated levels versus their attention levels while playing the hand made levels. To test this, it was assumed that there was a difference and the experiment was conducted to measure that difference.

4.1. Stroop Test

A Stroop test is a common attentional test that measures a person's reaction to conflicting information. In the original test (Stroop [42]), the participants had to name the color of a printed word, ignoring what the word itself said. When the word used was the name of a color that did not match the printed color, subjects took longer to respond. When a matching word and color were used, or when all words were printed in black, subjects responded faster.

Since Sokoban is a visual game, requiring the player to look elsewhere would be too significant of a disruption to play. Instead, an auditory form of the Stroop test was used, where subjects were required to respond to the pitch of a voice. The voice played would either be in a high pitch or a low pitch, and would either say word "High" or "Low." Previous research (MacLeod [23]) has shown that this form of the Stroop test does trigger the same conflict response, but to a lesser degree.

The use of the Stroop test allows for the players' attention levels to be measured while playing the game. Attention is a finite resource in the sense that the more attention a person pays to one thing, the less attention they have to pay to another (Sinnott et al. [39]). With

Pitch \ Word	High	Low
High	238 Hz 0.249 sec	205 Hz 0.274 sec
Low	133 Hz 0.249 sec	114 Hz 0.275 sec

TABLE 4.1. Pitch and duration of the voice files used in the study.

each Stroop prompt, the players’ reaction time can be measured and used to infer whether or not they were paying more or less attention to the game at that point in time.

The voices were produced using text-to-speech software for consistency. The durations and pitches of the four files generated are shown in Table 4.1. The voices were presented to the players through a pair of headphones. They were asked to press one of two buttons depending only on the pitch of the word. Responses were recorded until the next word was spoken.

4.2. Study Design

A study was conducted with 40 student volunteers from the University of North Texas. Participants were asked to complete several tasks: Take the Attention Network Test (Fan et al. [9]), practice the Stroop test by itself for five minutes, practice Sokoban by itself for 20 minutes, play two specified sets of levels (one hand made and one procedurally generated) for 30 minutes each, and finally take a survey. Each subject completed the study in a single two-hour block.

Half of the subjects took the Attention Network Test before they played the game, and half took it afterwards. Half of them practiced the Stroop test first, and half practiced Sokoban first. Half played the hand made levels first, and half played the procedurally generated levels first. Finally, the subjects were assigned evenly to each of the five hand made level sets. All subjects took both of the practice sessions before playing the main game

Please don't close any windows. You can shrink them or move them out of the way.

- 1: Take the Attention Network Test. Use guest as the group name and your ID as the session number. Feel free to move things around to make the test more comfortable. Please read and follow the instructions given.
- 2: Start SokoKong
- 3: Practice the Stroop test (under Level Sets) for 5 minutes.
- 4: Read the How To Play screen, then practice on the Easy level set for 20 minutes.
- 5: Play level set 6 for 30 minutes. Feel free to play the levels in any order or switch levels if you get stuck. Try to answer the Stroop test quickly and accurately, but pay more attention to the game itself.
- 6: Play level set 4 for 30 minutes. Feel free to play the levels in any order or switch levels if you get stuck. Try to answer the Stroop test quickly and accurately, but pay more attention to the game itself.
- 7: Take the participant survey.

FIGURE 4.1. One of the instruction sheets given to the study participants

and took the survey last. This gave a total of 40 combinations of conditions with one subject per combination. Participants were given printed instructions stating which order to do each task. [Figure 4.1](#) shows a sample of these instructions.

4.3. Study Data

Each of the subtasks, and the statistics collected from them, are detailed below.

4.3.1. Attention Network Test

The Attention Network Test (ANT) is a standard attentional test. The software used in this study (Fan [8]) presents the user with a series of four 5-minute tests. The first test is a practice session and tells the user whether or not they have pressed the correct button, but otherwise, each test is in the same format. During a test, the user is presented with a left or right arrow and must respond by pressing the left or right arrow key on the keyboard. The arrow appears above or below a fixation point (a small cross the user is instructed to fix their gaze upon) in the center of the screen. Some of the time, the central arrow appears by itself (the neutral case). The rest of the time, this central arrow is surrounded by two pairs of other arrows, either pointing in the same direction as the central arrow (the congruent case) or in the opposite direction (the incongruent case). Some of the time, before the arrow appears, one or two asterisks will flash on the screen to cue the user. This cue can either be a single asterisk where the arrow will appear; a single asterisk in the center on top of the fixation point; or two asterisks, one above the fixation point and one below. In total, this gives 12 possible combinations, with reaction times and accuracy recorded for each. For this study, the reaction times and accuracy were used and were separated by the congruency of the arrows.

4.3.2. Stroop Practice

During the Stroop practice session, the users listened to the voice files through a pair of headphones. The only things on-screen during this time were the instructions for how to respond and a line of text indicating whether or not they had responded correctly. The players were instructed to take the Stroop test with their right hand on the keyboard number pad, pressing 8 if the voice was in a high pitch and 2 if the voice was in a low pitch.

A timestamped event was recorded when a vocal prompt was made and another when the user responded. The event recorded also tracked exactly which prompt was made and

whether or not the user responded correctly. From this series of data points, six statistics were extracted: the users' reaction times, their accuracy, and the percentage of times they failed to respond, each separated by the congruency of the prompt. Because users were not expected to be able to identify the pitches correctly on the first few tries, having no point of reference, the first minute of data from the Stroop practice was not included in further analysis.

4.3.3. Sokoban Practice

During the Sokoban practice, no sound was played. Users were given a brief tutorial on how to play Sokoban, showing them how their avatar moved and interacted with the level. Then they were asked to play the easy level set at their own pace for 20 minutes. They were told to play the game with their left hand using the W, A, S, and D letter keys to move.

A timestamped event was recorded each time the user started, restarted, won, or quit a level. Each event also recorded how many moves and pushes the user made up to that point. From this, four statistics were extracted: how many levels the users attempted, how many levels they completed, how many moves they made beyond the minimum required for each level they completed, and how many pushes they made beyond the minimum.

4.3.4. Sokoban Gameplay

After practicing both the Stroop test and the Sokoban gameplay, subjects were asked to play two specific sets of levels. They were not informed that one of these was procedurally generated and none mentioned the possibility. Subjects were told to play the levels within each set in any order and at their own pace. During play, they had to respond to the Stroop prompts.

All of the events that were recorded during the two practice sessions were also recorded during the gameplay. Both sets of events were recorded into a single event log, and all of the same statistics were calculated afterwards. During these level sets, some players did not complete any levels; therefore, the number of excess moves and pushes in the players'

solutions were not meaningful. One additional statistic was calculated: the number of times a player quit a level without returning to it.

4.3.5. Survey

The survey used included several demographic questions, questions about the participants' computer use and game playing habits, and questions about their experience with the study. The demographic questions were taken from the 2010 U.S. Census. The survey was created in HTML and presented from a local file through a browser. The entire survey and the statistics collected from it are presented in [Appendix B](#) and [Appendix C](#). For the analysis of this study, the statistics used were the number of hours the participants spent on the computer, the number of hours per week the participants spent playing games, how much they liked games in general, whether or not they liked puzzle games in particular, and their responses to the five questions about their subjective reactions to the game used in the study.

4.4. Data Transformations

Many statistical methods assume that the data used are normally distributed. For this study, data that severely violated this assumption (subjects' reaction times to the Stroop prompts and all percentile data) were transformed before any analysis. Such normalizing transformations are not strictly necessary, but some research suggests that they can improve the statistical power of the results (Altman [1], and Kirisci and Hsu [21]).

4.4.1. Probabilities

$$(1) \quad \text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Because percentile data, as probabilities, are bounded on both sides but a linear regression makes no such assumptions attempting to use them directly in a regression analysis can lead to nonsensical results such as predicting that an event would have a negative

probability of occurring. For this reason, the logit transformation (give by [Equation 1](#), see Fleiss et al. [\[10\]](#)) was applied to all percentile data.

A probability can also be presented as the odds of an event occurring versus not occurring. This is simply the ratio of the two probabilities, $\frac{p}{1-p}$. Taking the log of this ratio gives the log-odds of the probability. The log-odds is not bounded in either direction: as an event gets less and less (more and more) likely, its log-odds goes to negative (positive) infinity. An event with a 50% chance of occurring will have a log-odds of 0.

Log-odds cannot represent events with 0% or 100% probabilities. Some players answered all Stroop prompts they were presented with, which makes their chances of not answering a prompt 0%. The `car` package in the R programming language handles this case by slightly scaling the data so that the log-odds of such events are a large, but finite, negative number. It handles 100% probabilities symmetrically.

4.4.2. Reaction Times

Reaction times do not follow any simple distribution, but research suggests that a combination of an exponential distribution and an inverse Gaussian distribution provide a good approximation (Schwarz [\[37\]](#)). Such times are bounded below and positively skewed which suggests that a log transformation, or some similar transformation, would be appropriate. A Box-Cox transformation parameter (Osborne [\[32\]](#)) was calculated. When the parameter is exactly zero, the Box-Cox transformation reduces to a log transformation. Since the parameter that best normalized the Stroop reaction time data was close to zero (approximately -0.34), a simple log transformation was used for its interpretability.

A quantile-quantile plot, abbreviated QQ plot, is a graphical device used to compare a sample to a distribution. (A normal-QQ plot would be a QQ plot comparing data to a normal distribution.) One axis measures the ideal quantiles from the target distribution, while the other measures the actual quantiles from the data. When the data points are plotted on such a graph, they should form a straight line if they match the distribution exactly. No data will match exactly though, so a confidence interval is often calculated around the best-fit line. The more points that fall within this interval, the more likely it is

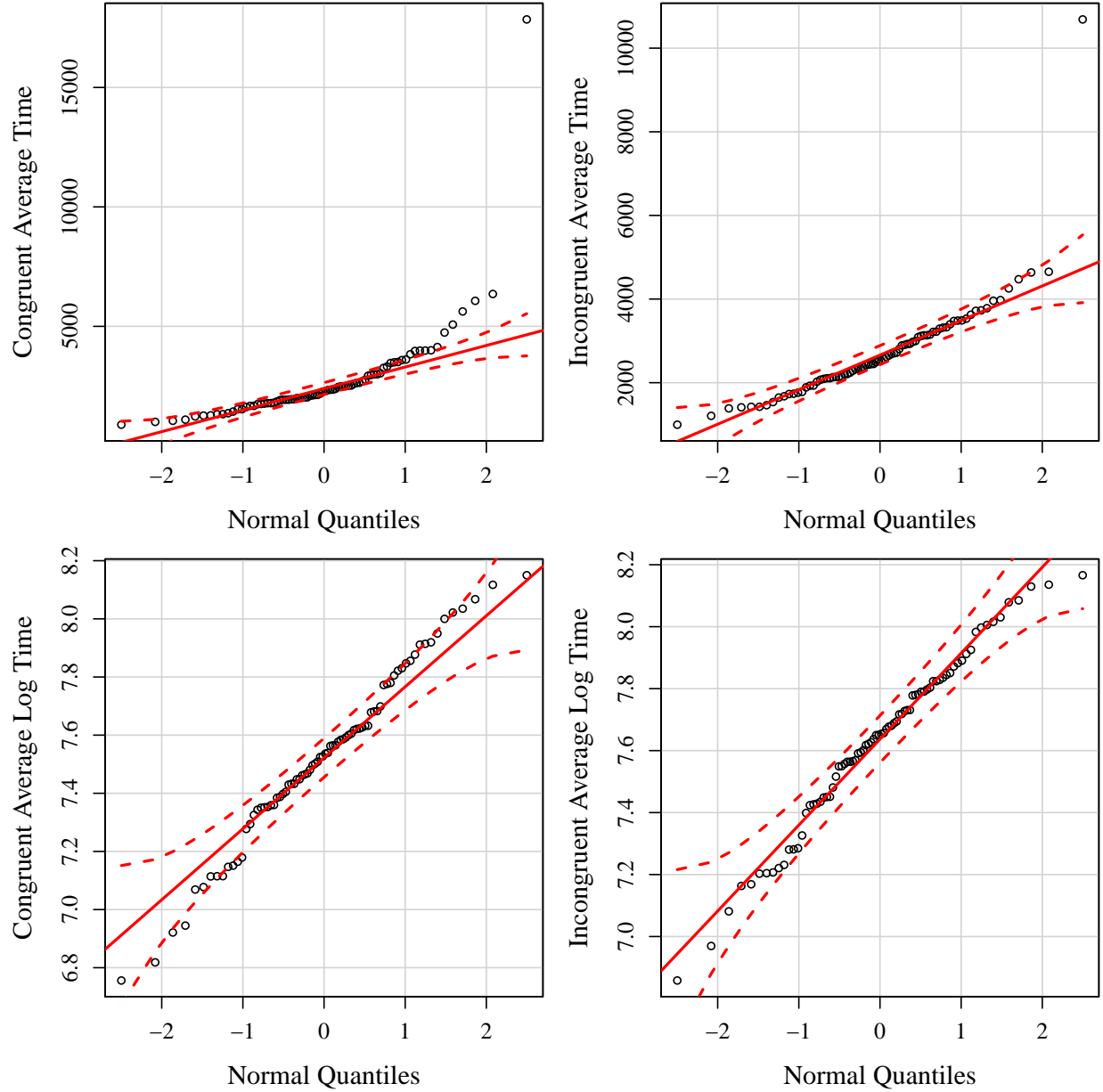


FIGURE 4.2. Normal-QQ plots for the users' reaction times to the congruent and incongruent prompts, with and without the log transformation. The dashed lines show 95% confidence intervals around the fitted lines.

that the points follow the target distribution. Figure 4.2 shows the QQ plots for the subjects' reaction times versus a normal distribution, before and after log transformation. The solid line shows the best-fit line, while the dashed lines show the confidence intervals about those

lines. While the figure shows the data separated by congruency, a single transformation was applied to the entire set of data points.

CHAPTER 5

ANALYSIS

5.1. Independent Variables

The study included five independent variables, four of which were between-subjects variables: the order in which the Attention Network Test was taken (before or after the main game), the order of the practice sessions, the order in which the two main level sets were played, and which human author's levels were played. The fifth independent variable was a within-subjects variable representing which level set was being played for each data point. Of these, the variable representing the level set being played was the only one relevant to the hypothesis being tested. The remaining four were evenly counterbalanced to remove any influence they might have had on the final results. Additionally, the interaction between the order of play and which level set was being played was tested to ensure that it did not interfere with the interpretation of the results.

5.2. Dependent Variables

The dependent variables taken from the Stroop test were the players' reaction times, their accuracy, and how often they failed to respond to a prompt, each separated into the congruent and incongruent cases for a total of six variables. [Figure 5.1](#) summarizes the raw results for each of these using bean plots, a form of kernel density plots.

A kernel density plot (Tukey [\[45\]](#)) is a type of smoothed histogram. For each data point, a smooth function is generated centered on that point. This function, called the kernel, is typically something similar to a Gaussian curve. The kernel density plot averages all the functions for each data point to produce an estimate of the distribution the data was drawn from. A bean plot presents two data sets side by side, in this case the data for the hand made levels versus that for the computer generated levels. For each set of data, it shows a kernel density plot along with a rug plot of the data (the short gray lines), the mean for each set (the long black lines), and the overall mean for the two data sets (the dotted line).

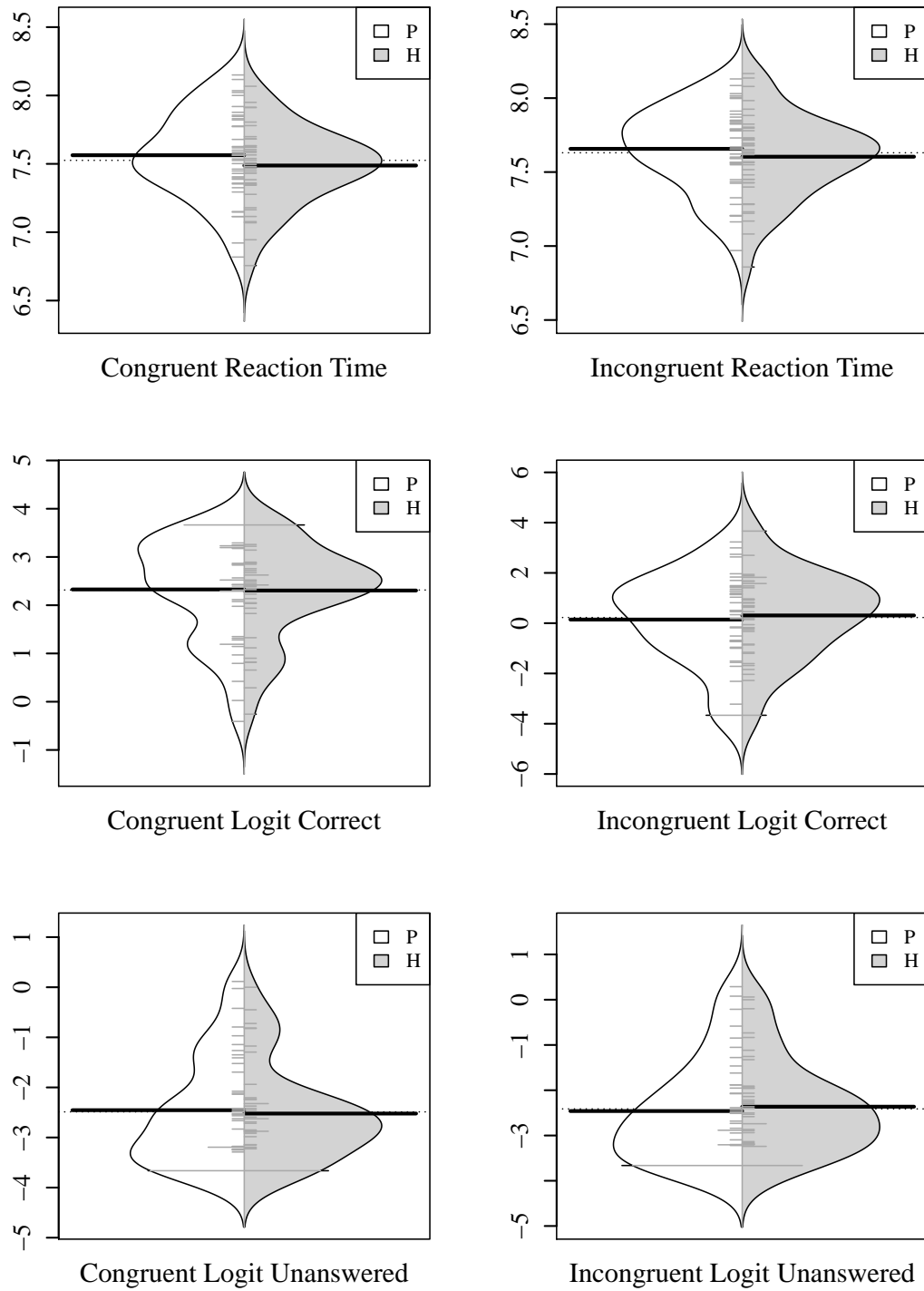


FIGURE 5.1. Bean plots summarizing the raw data for the dependent variables. The data for the procedurally generated levels are shown in white, and the data for the hand made levels are shown in gray.

5.3. Covariables

Few statistical methods work if there are more variables than subjects, and the few methods that do are typically less powerful. Rather than relying on such methods, the number of covariables was reduced to allow standard techniques to be used. Principal component analysis (Jolliffe [18]) was used for this reduction.

5.3.1. Principal Component Analysis

Principal component analysis (PCA) is a data reduction method that attempts to find a set of linear combinations of the variables such that each accounts for as much variance in the data as possible. These combinations are interpreted as underlying components that the data is composed of. It does this by computing the eigenvectors and eigenvalues of the correlation matrix of the data set. There is no hard rule for how many principal components should be calculated, but the most common rule is to keep the components whose eigenvalues are greater than one ([20]). Finally, those eigenvectors are rotated to normalize the results. Out of the multiple methods for accomplishing this, varimax rotation was used in this analysis. The rotated eigenvectors can then be used to transform the original variables into the principal components.

5.3.2. Covariable Groups

A separate principal component analysis was performed on each group of variables mentioned in [section 4.3](#). The survey data was split into two separate groups before this analysis: the answers to the computer use and game preference questions, and the answers to the subjective experience questions. [Figure 5.2](#) shows a scree plot, a plot of the sorted eigenvalues, for each of the six groups. The results of each of these analyses are given below. Values less than 0.1 have been omitted for clarity.

5.3.2.1. Difficulty

There were three within-subjects variables related to the difficulty of the levels being played. A PCA ([Table 5.1](#)) of these three variables resulted in two principal components that explained practically all (>99%) of the variance in this group.

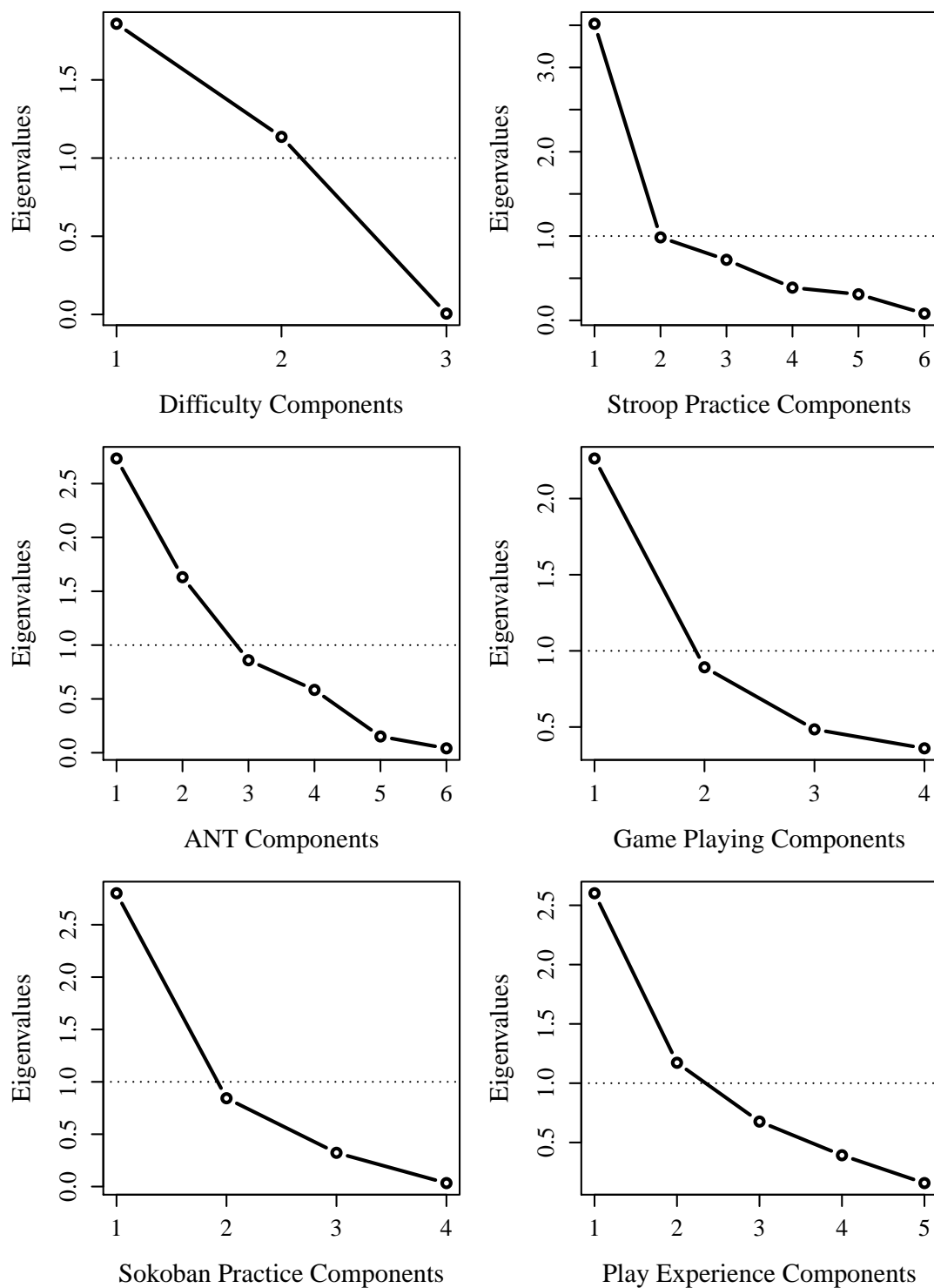


FIGURE 5.2. Scree plots summarizing the number of principal components taken from each group. Components with an eigenvalue above the dotted line were retained while the rest were discarded.

	Comp. 1	Comp. 2
Levels Attempted	0.969	0.241
Levels Solved	—	1.000
Levels Quit	0.957	-0.286

TABLE 5.1. The rotated component matrix for the difficulty covariables.

5.3.2.2. Attention Network Test

There were six variables related to the Attention Network Test results. The percentile data was transformed to log-odds before analysis. A PCA (Table 5.2) of these six variables resulted in two principal components that explained 73% of the variance.

	Comp. 1	Comp. 2
Congruent Reaction Time	0.962	—
Incongruent Reaction Time	0.868	0.239
Neutral Reaction Time	0.942	—
Congruent Accuracy	—	0.801
Incongruent Accuracy	0.258	0.728
Neutral Accuracy	—	0.706

TABLE 5.2. The rotated component matrix for the Attention Network Test covariables.

5.3.2.3. Sokoban Practice

There were four variables taken from the Sokoban practice session. Performing a PCA (Table 5.3) on this group resulted in one component that explained 70% of the variance.

5.3.2.4. Stroop Practice

There were six variables taken from the Stroop practice. A log transformation was applied to the reaction times and a logit transformation was applied to the percentile data

	Comp. 1
Levels Attempted	0.893
Levels Solved	0.899
Excess Moves	0.766
Excess Pushes	0.779

TABLE 5.3. The component matrix for the Sokoban practice covariables.

Since there is only one component, rotation does not change it.

before analysis. A PCA (Table 5.4) resulted in a single component that explained 59% of the variance of this group of variables.

	Comp. 1
Congruent Reaction Time	0.820
Incongruent Reaction Time	0.777
Congruent Accuracy	0.564
Incongruent Accuracy	0.467
Congruent Missed Prompts	-0.922
Incongruent Missed Prompts	-0.925

TABLE 5.4. The component matrix for the Stroop practice covariables.

5.3.2.5. Gaming Familiarity

There were four variables from the survey related to the players' computer and gaming habits. A PCA (Table 5.5) gave one principal component that explained 57% of the variance.

5.3.2.6. Study Experience

There were five variables related to the participants' subjective experience of the game. Two principal components (Table 5.6) explained 75% of the variance.

	Comp. 1
Computer Hours per Week	0.821
Hours per Week Gaming	0.832
Likes Games	0.822
Likes Puzzle Games	0.471

TABLE 5.5. The component matrix for the computer use and gaming habits covariables.

	Comp. 1	Comp. 2
Terrible vs. Wonderful	0.787	0.311
Difficult vs. Easy	-0.128	0.866
Frustrating vs. Satisfying	0.450	0.659
Dull vs. Stimulating	0.901	—
Boring vs. Fun	0.914	—

TABLE 5.6. The rotated component matrix for the subjective experience covariables.

5.4. Statistical Models

Three linear mixed models were constructed varying in the way the covariables were included in the model. In the first, the nine principal components from the groups mentioned above were used. In the second, all 28 covariables were analyzed using a single PCA, giving nine new components explaining 81% of the variance of the covariables. In the third, a single component was extracted from the 28 covariables (Parsons et al. [33]).

Each model was analyzed with the dependent variables and covariables as fixed effects and the subject IDs as a random effect with the assumption of a compound symmetric covariance structure. In each case, the adjusted scores for the procedurally generated levels minus that for the hand made levels were computed at a 95% confidence level. To get a final result for each variable, the most significant model was used. This method inflates the

Test	Model 1	Model 2	Model 3
Con.	-0.017 0.177 p = .103	0.028 0.182 p = .008	* 0.023 0.134 * * p = .006 *
Incon.	-0.045 0.149 p = .289	* 0.005 0.157 * * p = .036 *	-0.001 0.111 p = .052

TABLE 5.7. The log of the reaction time to the Stroop prompts during the procedurally generated levels minus the log of the reaction time during the hand made levels. The most significant results are highlighted.

significance of the final results, but since the objective is to show that no such difference exists, this only strengthens such a conclusion.

A linear mixed model (McCulloch [25]) is a linear regression model that includes both fixed and random effects. Mixed models are one of the more straightforward ways of handling repeated measures data, where multiple data points exist for each individual. In this case, two sets of dependent variables were measured for each player: the Stroop results during the procedurally generated levels and those during the hand made levels.

5.5. Results

Table 5.7, Table 5.8, and Table 5.9 summarize the results for the players' reaction times, accuracy, and rate of missed prompts respectively. For each category, the most significant result is highlighted. None of the models showed a strong interaction effect between the order of play and which level set was being played, which allows the separate interpretation of the effect of the level set.

5.5.1. Reaction Times

Reaction times were recorded as the log of the players' times in milliseconds. To get an interpretable answer, this transformation must be undone by exponentiating the raw differences. For example, in model three the upper bound for the raw difference for the congruent case is 0.134. Exponentiating this gives 1.143, which is an upper bound on the

Test	Model 1	Model 2	Model 3
Con.	* -0.202 0.700 *	-0.305 0.347	-0.281 0.264
	* p = .272 *	p = .899	p = .947
Incon.	-0.530 0.424	-0.602 0.185	* -0.537 0.019 *
	p = .823	p = .294	* p = .067 *

TABLE 5.8. Log-odds of the players' accuracy on the Stroop prompts during the procedurally generated levels minus that on the hand made levels. The most significant results are highlighted.

ratio between the times on the procedurally generated levels and those on the hand made levels. A slower reaction time implies that the players were paying more attention to the game and less to the Stroop test.

The third model gives the most significant results for the congruent times, and the second model gives the most significant results for the incongruent times. For the congruent reaction times, players were on average (95% confidence) took between 1.025 and 1.143 times as long to respond to the Stroop prompts during the procedurally generated levels. For the incongruent case, they took between 1.005 and 1.170 times as long. This implies that the players were paying more attention to the game during the procedurally generated levels compared to when they were playing the hand made levels. For completeness, the first model shows no significant difference for either the congruent case or the incongruent case, while the remaining model shows a less significant result in the same direction.

5.5.2. Accuracy

All percentile data was transformed to log-odds before analysis. To undo this transformation, these differences must be exponentiated, giving a ratio of the odds of the players answering the prompts correctly. A greater likelihood of getting a correct response implies that the players were paying less attention to the game and more to the Stroop test.

Test	Model 1	Model 2	Model 3
Con.	* -0.657 0.189 *	-0.295 0.323	-0.156 0.353
	* p = .271 *	p = .928	p = .438
Incon.	* -0.847 0.094 *	-0.446 0.243	-0.352 0.218
	* p = .114 *	p = .557	p = .639

TABLE 5.9. Log-odds of the chances of the players leaving a prompt unanswered during the Stroop test during the procedurally generated levels minus that on the hand made levels. The most significant results are highlighted.

The first model gave the most significant results for the congruent case, and the third model for the incongruent case; however, none of the models gave a result that was significant at the 95% level. The most significant of these models shows the players as between 0.817 and 2.014 times as likely to answer the Stroop prompts correctly during the procedurally generated levels (that is, between 18.3% less likely and 101.4% more likely). For the incongruent case, the most significant result shows the players as between 0.584 and 1.019 times as likely to answer correctly during the procedurally generated levels.

5.5.3. Missed Prompts

The data for the percentage of Stroop prompts left unanswered was also transformed to log-odds and must also be exponentiated. A higher likelihood of leaving a prompt unanswered implies that the player was paying more attention to the game and less to the Stroop test. The first model gave the most significant results for both the congruent and incongruent cases, but none of the results were significant at the 95% confidence level. Players were between 0.518 and 1.208 times as likely to leave a congruent Stroop prompt unanswered during the procedurally generated levels, and between 0.429 and 1.099 times as likely to leave an incongruent prompt unanswered.

5.6. Conclusion

While the reaction times suggest that the players were paying more attention to the procedurally generated levels, the overall results were mostly insignificant even with the inflation caused by choosing the most significant of the three models. Additionally, the effect size from the reaction times was quite small, with players only responding 17% slower as an upper bound. On balance, it seems that players were paying about as much attention to both types of levels, and therefore, found both types of levels about equally interesting.

CHAPTER 6

FUTURE WORK

6.1. Algorithmic Details

There are several unexplored avenues of research towards improving the generator further. The second step in the algorithm, placing the goals, currently tries every combination of legal goal squares. One possibility would be to have it generate only contiguous groupings of goals; something human designers often do for aesthetic purposes anyway. Without restricting the goals to one contiguous group, it might still be possible to find some commonality among the patterns of goal spaces used by human authors. Another possibility would be to apply the results from the theory of optimal stopping (Peskir and Shiryaev [35]) to allow the program to stop early while still having a high chance of returning a good result.

Another avenue of research relates to the aesthetics of the generated levels. Those created by the procedural generation algorithm often look noticeably different than those created by human authors. Partially, this is due to the generator creating certain configurations of walls forming rooms that are not used in the solutions. Whether or not these rooms serve to make the resulting level more difficult for humans by acting as red herrings is an open question.

6.2. Other Games

Another open question is how to expand the procedural generation algorithm to allow it to make levels for other games. Sokoban is a great experimental platform for computational and psychological purposes. Because the players' moves are not important unless they move a box, the players' exact position does not need to be tracked, nor does the exact path they take from one box to another need to be calculated. Additionally, Sokoban is deterministic when played in reverse, apart from the players' inputs. This is not true for many other games.

For example, if there was a second, computer controlled character within the game, the number of moves the player made getting from one point to another could change the

number of moves the second character would make, meaning the exact path taken by the player would need to be computed. Similarly, the player might have moved their avatar up, but knowing that would not be enough to reverse the move as the second character could potentially have reached their current position from multiple previous positions.

6.3. Statistics

Another avenue of research would be to perform this experiment again using a larger number of participants. More participants would allow for the detection of smaller differences or for the inclusion of more covariables. Relatedly, further experiments are needed to determine which covariables are the most useful in this type of study.

The principal component analysis of the users' responses to survey questions related to their subjective experience of the game ([section 5.3.2.6](#)) suggests that the difficulty of the levels they were playing and their enjoyment of those levels are not strongly related. Further research is needed to determine whether there is a connection at all and how it might vary from player to player.

APPENDIX A

PSEUDOCODE

This appendix contains pseudocode for the algorithm with all of the optimizations and shortcuts discussed in [Chapter 2](#).

GENERATEWALLS(w, h)

```
1  // Returns an empty room of  $(3 \cdot w) \times (3 \cdot h)$  squares
2  Initialize board to a blank grid of  $w \times h$  regions
3  count = 0
4  fails = 0
5  while count <  $w \cdot h$ 
6      Choose a random template
7      Rotate and reflect template randomly
8      Choose a random region
9      if template can be placed in region
10         Place template on board
11         count = count + 1
12     else
13         fails = fails + 1
14         if fails > maxFails
15             count = 0
16             fails = 0
17             Clear board
18 return board
```

```

ADVANCE(frontier)
1  // Returns the set of states reachable from frontier in one box line
2  results = EMPTY SET
3  for each frontier.state in frontier
4      for each box reachable by the player in frontier.state
5          for each temp.state reachable by pulling box along a straight line
6              if box is not on a slippery tile in temp.state
7                  Add temp.state to results
8  return results

```



```

SEARCH(board)
1  // Returns the farthest states from board
2  start = EMPTY SET
3  for each goal on board
4      Place a box on goal
5  for each contiguous section of floor on board
6      Add board with player in section to start
7  frontier.main = start
8  layer = 0
9  repeat
10     temp = ADVANCE(frontier.main)
11     if temp == EMPTY
12         return frontier.main
13     frontier.main = temp
14     frontier.second = start
15     frontier.main = frontier.main - frontier.second
16     for each i in 0 to layer - 1
17         frontier.second = ADVANCE(frontier.second)
18         frontier.main = frontier.main - frontier.second
19     layer = layer + 1

```

GOALS(*board*, *n*.boxes)

```
1  // Returns an n-box puzzle within board
2  count = 0
3  while count < n.boxes
4      Erase player and boxes from board
5      floors = EMPTY LIST
6      for each floor.tile on board
7          Add floor.tile to floors
8      Shuffle floors
9      best = EMPTY SET
10     for each pair of floor.tile in floors
11         Temporarily mark both floor.tile as goals in board
12         temp = SEARCH(board)
13         if length of solutions in temp > length of solutions in best
14             best = temp
15     Choose one level from best
16     board = level
17     count = count + 2
18 return board
```

MAIN(*w*, *h*, *n*)

```
1  // Outputs a new  $w \times h$  puzzle with n boxes
2  board = GENERATEWALLS(w, h)
3  board = GOALS(board, n)
4  Output board
```

APPENDIX B

PARTICIPANT SURVEY

Research Participant Survey

Enter your ID number.

What is your date of birth?

 - -

What is your gender?

- ☐ Male
☐ Female

What is your race/ethnicity?

- ☐ White
☐ Black, African American or Negro
☐ American Indian or Alaska Native
☐ Asian Indian ☐ Japanese ☐ Native Hawaiian
☐ Chinese ☐ Korean ☐ Guamanian or Chamorro
☐ Filipino ☐ Vietnamese ☐ Samoan
☐ Other Asian ☐ Other Pacific Islander
☐ Other (Please specify)

What is your *current* marital status?

- ☐ Single / Never married
☐ Married / Living with partner
☐ Separated
☐ Widowed
☐ Divorced

What is the highest level of education you have attained?

- ☐ High school, GED or equivalent
☐ Specialized training after high school
☐ Associate's degree
☐ Bachelor's degree
☐ Master's degree
☐ Doctorate

What is or was your main occupation? _____

Do you or did you use a computer at your job? _____

- ☐ Yes
☐ No

Do you have a computer at home? _____

- ☐ Yes
☐ No

On average how many hours a week do you currently spend on a computer? _____

- ☐ 0 - 5 hours
☐ 5 - 10 hours
☐ 10 - 20 hours
☐ 20 - 40 hours
☐ 40+ hours

How would you rate your competency in terms of knowing how to use a computer? _____

- ☐ Completely inexperienced
☐ Inexperienced
☐ Somewhat experienced or inexperienced
☐ Experienced
☐ Very experienced

On average how many hours a week do you spend on the following computer activities? _____

Word processing	<input type="text"/>
Programming	<input type="text"/>
Game playing	<input type="text"/>
Data entry / processing	<input type="text"/>
Graphics design / art	<input type="text"/>
Surfing the internet	<input type="text"/>
Emailing	<input type="text"/>
Socializing	<input type="text"/>
Other	<input type="text"/>

Rate how you feel about each item or activity below.

	Dislike Greatly			Neutral			Enjoy Greatly	No Opinion
Computers in general	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Computer games	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Programming	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Word processing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Data processing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

What (if any) genres of video games do you enjoy?

- ☐ Action
- ☐ Adventure
- ☐ Role playing
- ☐ Simulation
- ☐ Strategy
- ☐ Puzzle
- ☐ Sports
- ☐ Other

What is your dominant hand?

- ☐ Left
- ☐ Right
- ☐ Neither / ambidextrous

Do you require any corrective eye-wear?

- ☐ Yes
- ☐ No

Have you been diagnosed with ADHD or a similar attention disorder?

- ☐ Yes
- ☐ No

Describe your overall reaction to this game by selecting the adjectives that most closely match the way you feel. Use the intermediate boxes to indicate, for example, "somewhat terrible."

Terrible	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Wonderful
Difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Easy
Frustrating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Satisfying
Dull	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Stimulating
Boring	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Fun

Do you have any other thoughts or comments?

Save

FIGURE B.1. A reproduction of the survey from the study. These images are screenshots taken from the HTML survey. Participants were presented the survey as one continuous file rather than in separate pages.

APPENDIX C

SURVEY RESULTS

Age	18-21	22-24	25-30	31-36	37-42
Count	22	6	8	2	2

TABLE C.1. Participants' ages.

Male	21
Female	19

TABLE C.2. Participants' gender.

White	23
Black	3
Indian	1
Chinese	2
Korean	1
Filipino	1
Vietnamese	1
Other Asian	1
Other	1
Multiple	3

TABLE C.3. Participants' ethnicity. *Multiple* includes all cases where more than one box was checked.

Single	34
Married	6
Separated	0
Widowed	0
Divorced	0

TABLE C.4. Participants' marital status.

High School	24
Specialized Training	0
Associate's Degree	8
Bachelor's Degree	7
Master's Degree	1
Doctorate	0

TABLE C.5. Highest level of education attained by participants.

Student/none	19
Other	20
No response	1

TABLE C.6. Participants' occupation. *Student/none* includes all responses that included "student." *Other* includes all responses that were given by only one participant.

No	14
Yes	25
No answer	1

TABLE C.7. Whether or not participants had a computer at their job.

No	0
Yes	40

TABLE C.8. Whether or not participants had a computer at home.

0-5	2
5-10	11
10-20	10
20-40	8
40+	9

TABLE C.9. Hours per week participants spent on the computer.

Completely inexperienced	0
Inexperienced	2
Somewhat experienced or inexperienced	10
Experienced	19
Very Experienced	9

TABLE C.10. Participants' estimation of their competency in the use of a computer.

	0	1-5	5-10	10-20	20-40	40+	No answer
Word processing	2	20	8	6	1	0	3
Programming	23	2	3	4	5	0	3
Game playing	14	10	2	10	1	0	3
Data entry	25	8	2	0	1	0	4
Graphics design	25	8	1	3	0	1	2
Surfing	0	10	10	9	8	2	1
Emailing	2	25	3	4	3	0	3
Socializing	6	16	6	5	3	1	2
Other activities	12	8	4	4	1	0	12

TABLE C.11. Hours per week participants spent on various tasks on a computer.

	Dislike Greatly			Enjoy Greatly				No opinion
Computers in general	0	0	0	4	3	6	27	0
Computer games	1	3	2	6	7	3	18	0
Programming	3	2	3	10	5	3	7	6
Word processing	0	1	7	19	5	3	3	2
Data processing	2	4	7	15	3	2	2	5

TABLE C.12. Participants' impressions of various computer related activities.

Action	21
Adventure	26
Role playing	22
Simulation	18
Strategy	23
Puzzle	17
Sports	11
Other	10

TABLE C.13. Game genres enjoyed by participants.

Left	3
Right	37

TABLE C.14. Participants' dominant hand.

No	19
Yes	21

TABLE C.15. Participants needing corrective eyewear.

No	38
Yes	2

TABLE C.16. Participants diagnosed with some attention disorder.

Terrible	2	2	4	13	14	5	0	Wonderful
Difficult	5	16	11	6	1	1	0	Easy
Frustrating	5	10	11	10	4	0	0	Satisfying
Dull	4	3	7	9	8	8	1	Stimulating
Boring	5	1	9	9	8	7	1	Fun

TABLE C.17. Participants subjective impression of the game.

APPENDIX D

IRB

This appendix contains a copy of the proof of the University of North Texas Institutional Review Board's approval to perform a study using human subjects. It also contains a copy of the informed consent form given to the participants.



A green light to greatness.™

Office of the Vice President of Research and Economic Development

May 2, 2013

OFFICE OF RESEARCH SERVICES

Supervising Investigator: Dr. Ian Parberry
Student Investigator: Joshua Taylor
Department of Computer Science
University of North Texas

Re: Human Subjects Application No. 13248

Dear Dr. Parberry:

As permitted by federal law and regulations governing the use of human subjects in research projects (45 CFR 46), the UNT Institutional Review Board has reviewed your proposed project titled "Measuring Interest in Puzzle Levels." The risks inherent in this research are minimal, and the potential benefits to the subject outweigh those risks. The submitted protocol is hereby approved for the use of human subjects in this study. **Federal Policy 45 CFR 46.109(e) stipulates that IRB approval is for one year only, May 2, 2013 to May 1, 2014.**

Enclosed is the consent document with stamped IRB approval. Please copy and **use this form only** for your study subjects.

It is your responsibility according to U.S. Department of Health and Human Services regulations to submit annual and terminal progress reports to the IRB for this project. The IRB must also review this project prior to any modifications. **If continuing review is not granted before May 1, 2014, IRB approval of this research expires on that date.**

Please contact Shelia Bourns, Research Compliance Analyst at extension 3940 if you wish to make changes or need additional information.

Sincerely,

Patricia L. Kaminski, Ph.D.
Associate Professor
Department of Psychology
Chair, Institutional Review Board

PK/sb

UNIVERSITY OF NORTH TEXAS™

1155 Union Circle #305250 Denton, Texas 76203-5017
940.565.3940 940.565.4277 fax <http://research.unt.edu>

University of North Texas Institutional Review Board

Informed Consent Form

Before agreeing to participate in this research study, it is important that you read and understand the following explanation of the purpose, benefits and risks of the study and how it will be conducted.

Title of Study: Measuring Interest in Puzzle Levels

Student Investigator: Joshua Taylor, University of North Texas (UNT) Department of Computer Science. **Supervising Investigator:** Dr. Ian Parberry.

Purpose of the Study: You are being asked to participate in a research study which involves measuring how attentive people are to different levels of a puzzle game. We hope to use this information to determine which levels are more interesting.

Study Procedures: You will be asked to take a standard attention test, a demographic survey and play a puzzle game and respond to an auditory signal, first separately and then simultaneously. This will take a total of about two hours of your time. During this time, the researcher administering the test will be on hand to answer any procedural questions and to observe how you play the game.


Foreseeable Risks: No foreseeable risks are involved in this study.

Benefits to the Subjects or Others: This study is not expected to be of any direct benefit to you, but we hope to learn more about what makes some levels more interesting than others.

Compensation for Participants: You will receive a class grade for participating in this study. If you choose not to participate, there is an alternate assignment available for the same grade.

Procedures for Maintaining Confidentiality of Research Records: The only personally identifiable information collected will be your signature on this form which will be stored in a locked office. All other information will be recorded using an anonymous ID number. The confidentiality of your individual information will be maintained in any publications or presentations regarding this study.

Questions about the Study: If you have any questions about the study, you may contact Joshua Taylor at JoshuaTaylor@my.unt.edu, Ian Parberry at ian@cs.unt.edu or Thomas Parsons at Thomas.Parsons@unt.edu.

APPROVED BY THE UNT IRB
FROM 5/2/13 TO 5/1/14


Office of Research Services
University of North Texas
Last Updated: July 11, 2011

Review for the Protection of Participants: This research study has been reviewed and approved by the UNT Institutional Review Board (IRB). The UNT IRB can be contacted at (940) 565-3940 with any questions regarding the rights of research subjects.

Research Participants' Rights: Your signature below indicates that you have read or have had read to you all of the above and that you confirm all of the following:

- Joshua Taylor has explained the study to you and answered all of your questions. You have been told the possible benefits and the potential risks and/or discomforts of the study.
- You understand that you do not have to take part in this study, and your refusal to participate or your decision to withdraw will involve no penalty or loss of rights or benefits. The study personnel may choose to stop your participation at any time.
- Your decision whether to participate or to withdraw from the study will have no effect on your grade or standing in any UNT course.
- You understand why the study is being conducted and how it will be performed.
- You understand your rights as a research participant and you voluntarily consent to participate in this study.
- You have been told you will receive a copy of this form.

Printed Name of Participant

Signature of Participant

APPROVED BY THE UNT IRB
FROM 5/2/13 TO 5/1/14
JB

Date

For the Student Investigator: I certify that I have reviewed the contents of this form with the subject signing above. I have explained the possible benefits and the potential risks and/or discomforts of the study. It is my opinion that the participant understood the explanation.

Signature of Student Investigator

Date

REFERENCES

- [1] Douglas G. Altman, *Practical statistics for medical research*, CRC Press, 1990. 28
- [2] Ping-Chiang Chou, Shi-Jim Yen, Cheng-Wei Chou, Ching-Nung Lin, Chang-Shing Lee, Olivier Teytaud, and Hassen Doghmen, *A simple tsumego generator*, GPW, 2012. 6
- [3] Michael F Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen, *Wang tiles for image and texture generation*, ACM SIGGRAPH 2003, vol. 22, ACM, 2003. 1
- [4] Joseph Culberson, *Sokoban is PSPACE-complete*, Proceedings of the International Conference on Fun with Algorithms, 1998, pp. 65–76. 13
- [5] *Diablo III*, <http://us.battle.net/d3/en/>. 4
- [6] Dorit Dor and Uri Zwick, *Sokoban and other motion planning problems*, Computational Geometry 13 (1999), no. 4, 215–228. 5
- [7] Jon Doran and Ian Parberry, *Controlled procedural terrain generation using software agents*, IEEE Transactions on Computational Intelligence and AI in Games 2 (2010), no. 2, 111–119. 1
- [8] Jin Fan, https://www.sacklerinstitute.org/cornell/assays_and_tools/ant/jin.fan/. 26
- [9] Jin Fan, Bruce D. McCandliss, Tobias Sommer, Amir Raz, and Michael I. Posner, *Testing the efficiency and independence of attentional networks*, Journal of Cognitive Neuroscience 14 (2002), no. 3, 340–347. 24
- [10] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik, *Statistical methods for rates and proportions*, John Wiley & Sons, 2013. 29
- [11] Robert W Floyd, *Algorithm 97: shortest path*, Communications of the ACM 5 (1962), no. 6, 345. 13
- [12] Hana Galperin and Avi Wigderson, *Succinct representations of graphs*, Information and Control 56 (1983), no. 3, 183–198. 13
- [13] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979. 14

- [14] Mark J Harris, William V Baxter, Thorsten Scheuermann, and Anselmo Lastra, *Simulation of cloud dynamics on graphics hardware*, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Eurographics Association, 2003, pp. 92–101. 1
- [15] John E Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Pearson Education India, 1979. 14
- [16] Petr Jarušek and Radek Pelánek, *Difficulty rating of Sokoban puzzle*, Proceedings of the Fifth Starting AI Researchers’ Symposium, 2010. 12
- [17] ———, *Human problem solving: Sokoban case study*, Tech. Report FIMU–RS–2010–01, Faculty of Informatics, Masaryk University Brno, 2010. 5, 12
- [18] Ian Jolliffe, *Principal component analysis*, Wiley Online Library, 2002. 34
- [19] Andreas Junghanns and Jonathan Schaeffer, *Sokoban: A challenging single-agent search problem*, Proceedings of the IJCAI Workshop on Using Games as an Experimental Testbed for AI Research, 1997, pp. 27–36. 5
- [20] Henry F Kaiser, *The application of electronic computers to factor analysis.*, Educational and psychological measurement (1960). 34
- [21] Levent Kirisci and Tse-Chi Hsu, *The effect of the multivariate Box-Cox transformation on the power of MANOVA.*, (1993), Annual Meeting of the American Educational Research Association. 28
- [22] Kongregate, <http://www.kongregate.com/>. 20
- [23] Colin M. MacLeod, *Half a century of research on the Stroop effect: An integrative review.*, Psychological Bulletin 109 (1991), no. 2, 163. 23
- [24] Odawara Masaru, Kaneko Tomoyumi, and Kawai Satoru, *A method of automatic creation of goal-area in Sokoban maps*, Joho Shori Gakkai Shinpojiumu Ronbunshu (2003), 67–74. 5
- [25] Charles E McCulloch and John M Neuhaus, *Generalized linear mixed models*, Wiley Online Library, 2001. 39

- [26] Paul Merrell, *Example-based model synthesis*, Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM, 2007, pp. 105–112. 1
- [27] *Minecraft*, <https://minecraft.net/>. 4
- [28] Makina Mühendisi, *Sokoban Level Generators (A to Z)*, <http://www.erimsever.com/sokoban7.htm>. 6
- [29] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga, *Automatic making of Sokoban problems*, PRICAI'96: Topics in Artificial Intelligence (Norman Foo and Randy Goebel, eds.), Lecture Notes in Computer Science, vol. 1114, Springer, 1996, pp. 592–600. 5
- [30] *NetHack*, <http://www.nethack.org/>. 4
- [31] Jacob Olsen, *Realtime procedural terrain generation*, (2004). 1
- [32] Jason W. Osborne, *Improving your data transformations: Applying the Box-Cox transformation*, Practical Assessment, Research & Evaluation 15 (2010), no. 12, 1–9. 29
- [33] Thomas D. Parsons, Albert R. Rizzo, Cheryl van der Zaag, Jocelyn .S McGee, and J. Galen Buckwalter, *Gender differences and cognition among older adults*, Aging, Neuropsychology, and Cognition 12 (2005), no. 1, 78–88. 38
- [34] Ken Perlin, *An image synthesizer*, ACM Siggraph Computer Graphics 19 (1985), no. 3, 287–296. 2
- [35] Goran Peskir and Albert Shiryaev, *Optimal stopping and free-boundary problems*, Springer, 2006. 43
- [36] Adrien Peytavie, Eric Galin, Jérôme Grosjean, and Stephane Merillou, *Procedural generation of rock piles using aperiodic tiling*, Computer Graphics Forum, vol. 28, Wiley Online Library, 2009, pp. 1801–1809. 1
- [37] Wolfgang Schwarz, *The ex-Wald distribution as a descriptive model of response times*, Behavior Research Methods, Instruments, & Computers 33 (2001), no. 4, 457–469. 29
- [38] Frédéric Servais, *Finding Hard Initial Configurations of Rush Hour with Binary Decision Diagrams*, Master's thesis, Université libre de Bruxelles, Faculté des Sciences, 2005. 6

- [39] Scott Sinnett, Albert Costa, and Salvador Soto-Faraco, *Manipulating inattentional blindness within and across sensory modalities*, The Quarterly Journal of Experimental Psychology 59 (2006), no. 8, 1425–1442. 23
- [40] *Spelunky*, <http://www.spelunkyworld.com/>. 4
- [41] Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, R Měch, Oliver Deussen, and Bedrich Benes, *Inverse procedural modelling of trees*, Computer Graphics Forum, vol. 33, Wiley Online Library, 2014, pp. 118–131. 1
- [42] J. Ridley Stroop, *Studies of interference in serial verbal reactions*, Journal of Experimental Psychology 18 (1935), no. 6, 643. 23
- [43] Taniguchi, *Taniguchi’s programs*, http://homepage2.nifty.com/yuki-tani/index_e.html. 6
- [44] *Terraria*, <https://terraria.org/>. 4
- [45] John W. Tukey, *Exploratory data analysis*, Addison-Wesley, 1977. 32
- [46] Glenn R. Wichman, *A brief history of “rogue”*, <http://www.wichman.org/roguehistory.html>. 4