The College of Wooster
Open Works

Senior Independent Study Theses

2020

# A Mathematical Analysis of the Game of Santorini

Carson Clyde Geissler
*The College of Wooster*, cgeissler20@wooster.edu

Follow this and additional works at: https://openworks.wooster.edu/independentstudy

Part of the Artificial Intelligence and Robotics Commons, Discrete Mathematics and Combinatorics Commons, Other Mathematics Commons, and the Theory and Algorithms Commons

Recommended Citation
Geissler, Carson Clyde, "A Mathematical Analysis of the Game of Santorini" (2020). *Senior Independent Study Theses.* Paper 8917.

This Senior Independent Study Thesis Exemplar is brought to you by Open Works, a service of The College of Wooster Libraries. It has been accepted for inclusion in Senior Independent Study Theses by an authorized administrator of Open Works. For more information, please contact openworks@wooster.edu.

© Copyright 2020 Carson Clyde Geissler

# A Mathematical Analysis of the Game of Santorini

Independent Study Thesis

Presented in Partial Fulfillment of the
Requirements for the Degree Bachelor of Arts in
the Department of Mathematical and
Computational Sciences at The College of Wooster

by
Carson Geissler

The College of Wooster
2020

**Advised by:**

Dr. Nathan Fox

# Abstract

*Santorini* is a two player combinatorial board game. *Santorini* bears resemblance to the graph theory game of Geography, a game of moving and deleting vertices on a graph. We explore *Santorini* with game theory, complexity theory, and artificial intelligence. We present David Lichtenstein's proof that Geography is PSPACE-hard and adapt the proof for generalized forms of *Santorini*. Last, we discuss the development of an AI built for a software implementation of *Santorini* and present a number of improvements to that AI.

# Acknowledgements

I would first like to thank my adviser, Dr. Nathan Fox, for his patience, his wit, and his candor throughout our time working together. I also would like to thank my parents and grandparents for supporting me in everything I do and for impressing upon me the value of education and a dedicated work ethic. Lastly, I would like to acknowledge my many siblings for instilling in me an ardent curiosity, a passionate love of board games, and a fierce competitive spirit.

# Contents

**Abstract** **iii**

**Acknowledgements** **v**

**1 Introduction** **1**

**2 *Santorini* and its Rules** **3**

**3 Relevant Mathematical Theory** **9**

3.1 Introduction to Graph Theory . . . . . . . . . . . . . . . . . . . . . . 9

3.2 Introduction to Game Theory . . . . . . . . . . . . . . . . . . . . . . 13

3.2.1 Brief History of the Field . . . . . . . . . . . . . . . . . . . . 13

3.2.2 Terminology Relevant to *Santorini* . . . . . . . . . . . . . . . 14

3.2.3 Game Trees . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

**4 Game Theory As it Applies to *Santorini*** **21**

4.1 Generalizing the Gameplay Elements of *Santorini* . . . . . . . . . . 21

4.2 Brute-Forcing Small Cases . . . . . . . . . . . . . . . . . . . . . . . . 24

**5 Introduction to Complexity Theory** **27**

5.1 Bachmann-Landau Notation . . . . . . . . . . . . . . . . . . . . . . . . 29

5.2 Complexity Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

5.3 Reductions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

5.4 Completeness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

**6 Proving P-SPACE Completeness** **39**

6.1 QSAT Game and the Game of Geography . . . . . . . . . . . . . . . 39

6.2 Proof that Geography is PSPACE-hard . . . . . . . . . . . . . . . . . 40

6.3 Bridging the Gap to *Santorini* . . . . . . . . . . . . . . . . . . . . . . 42

6.4 Proof that *Santorini* is in PSPACE . . . . . . . . . . . . . . . . . . . 51

**7 AI Development Practices** **55**

7.1 Brief History of AI Development . . . . . . . . . . . . . . . . . . . . 55

7.2 Look-Ahead Algorithms and Heuristics . . . . . . . . . . . . . . . . 56

**8 AI in *Santorini*** **61**

8.1 Limitations of Look-ahead . . . . . . . . . . . . . . . . . . . . . . . . 61

8.2 Valuable Heuristics in *Santorini* . . . . . . . . . . . . . . . . . . . . . 63

8.3 Improvements Using a Genetic Algorithm . . . . . . . . . . . . . . . 67

**9 Conclusion** **69**

# List of Figures


2.1 The board and pieces of the 2017 re-release [10] . . . . . . . . . . . 4

2.2 A guide to Santorini's building pieces and a depiction of a winning move [14] . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.3 A visualization of a worker's move options [14] . . . . . . . . . . 5

2.4 A visualization of a worker's move options with an unreachable adjacent space [14] . . . . . . . . . . . . . . . . . . . . . . . . 5

2.5 A visualization of a worker's build options [14] . . . . . . . . . . . 7

3.1 A simple graph with 5 vertices . . . . . . . . . . . . . . . . . . . . . 10

3.2 The board of *Santorini* in the form of a graph . . . . . . . . . . . . 11

3.3 The complete graph on 4 vertices, $K_4$ . . . . . . . . . . . . . . . . . 12

3.4 A game tree for a particular board state of Tic-Tac-Toe [5] . . . . . 17

4.1 The cycles on 5 and 6 nodes, $C_5$ and $C_6$ . . . . . . . . . . . . . . . . 26

5.1 A representation of the tape of a Turing machine. The location of the head of the machine is bold [17] . . . . . . . . . . . . . . . . . . 32

5.2 An illustration of the hierarchy of the complexity classes [4]. . . . 34

6.1 The graph of the original game of Geography being played with city names [11]. . . . . . . . . . . . . . . . . . . . . . . . . . . 40

6.2 A graph for a game of Geography that has an equivalent winning strategy to the quantified boolean formula game [11] . . . . . . . 41

6.3 A graph for a game of partizan Geography that has an equivalent winning strategy to the quantified boolean formula game . . . . . 44

6.4 A graph for a game of Geography with *Santorini*'s building rules that has an equivalent winning strategy to the quantified boolean formula game . . . . . . . . . . . . . . . . . . . . . . . . . 46

6.5 A graph for a game of partizan Geography with *Santorini*'s building rules that has an equivalent winning strategy to the quantified boolean formula game . . . . . . . . . . . . . . . . . . . . . . . 48

7.1 A tree on which the minimax algorithm is being executed [1] . . . 57

7.2 An example where the passed alpha-beta values allow the pruning of a node [1] . . . . . . . . . . . . . . . . . . . . . . . . . . 58

7.3 An example where the passed alpha-beta values allow the pruning of a subtree [1] . . . . . . . . . . . . . . . . . . . . . . . . 59

8.1 An example *Santorini* board where Player 1 (pieces represented by A and B) has a height sum of 4 while Player 2 (pieces represented by X and Y) has a height sum of 2. The evaluation of this board for Player 1 is thus $4 - 2 = 2$. . . . . . . . . . . . . . . . . . . . 64

8.2 An example *Santorini* board where Player 1 (pieces represented by A and B) has a centricity sum of 3 while Player 2 (pieces represented by X and Y) has a centricity sum of 1. The evaluation of this board for Player 1 is thus $3 - 1 = 2$. . . . . . . . . . . . . . . . 65

8.3 An example *Santorini* board showcasing the distances between pieces. We can evaluate Player 1's position by subtracting from 8 the sum of the minimum of the blue distances and the minimum of the red distances: $8 - (2 + 1) = 5$. . . . . . . . . . . . . . . . . . . . 66

# 1

# Introduction

> *"The core of mathematics is problem solving. Games are the most joyous excuse for problem solving."*
>
> – Dr. Gordon Hamilton, *Creator of* Santorini

Everyone loves solutions. Solutions mean answers. Solutions mean results. In board games, solutions mean winning. This paper is born out of a desire to solve a complex board game, the game of *Santorini*. However, solving board games can be notoriously difficult. While Tic-Tac-Toe is solvable by any well-meaning, methodical individual with a piece of scratch paper and a few minutes to spare, solutions become quite a bit more complicated with only small increases in game complexity.

Despite the far-reaching goal of attaining a solution to *Santorini* being fairly infeasible, this paper analyzes *Santorini* from three distinct mathematical approaches. First, this paper explores the game theory concepts necessary to understand *Santorini* and draws parallels between *Santorini* and other games

studied by mathematicians and game theorists. This exploration includes an overview of elementary graph theory concepts, definitions of a variety of game theory terms, and a discussion of simplified variants of *Santorini*. Second, this paper analyzes *Santorini* from a complexity theory standpoint, both giving a brief introduction to the field of complexity theory and proving an upper bound on the space (memory) complexity of finding a winning strategy for *Santorini*. The complexity theory analysis also includes a number of proofs that are indicative of the computational complexity of solving several restricted, generalized variants of *Santorini*. Third, this paper explores various artificial intelligence development practices and expounds on their use in developing an AI for *Santorini*. It goes further to discuss this project's accompanying software implementation of *Santorini* and the AI developed in accordance with it.

The purpose of this paper is to introduce one to the framework with which board games are discussed and analyzed. Such a purpose stems from a profound love of board games, and it is hoped that there is as much joy and discovery in reading this paper as there was in writing it.

# 2

# *Santorini* and its Rules

*Santorini* is an abstract strategy board game by designer and mathematician Gordon Hamilton. *Santorini* was tested and revised over thirty years by Hamilton [21] before settling into its final state in 2004. This first edition was released with simple, abstract pieces and few thematic tie-ins. The game was re-released through Roxley Games in early 2017 as a result of a highly successful Kickstarter campaign, raising over $700,000 from 7,100 backers [21]. This new release brought in a lot of publicity and support for the game, and has landed *Santorini* just outside of the board gaming website BoardGameGeek's fabled top 100 games, coming in at 108th overall [10].

*Santorini* was designed to be simple to learn and difficult to master. The game is played on a $5 \times 5$ grid of open spaces where each space is considered to be adjacent to the eight surrounding spaces. Each player has two identical workers who serve as the focal points for moving, building, and winning. The game begins with each player placing their workers, each on unique empty spaces. From there, the players alternate turns being the active player. On any

Figure 2.1: The board and pieces of the 2017 re-release [10]

given turn, a player must move one of their workers and then build with that moved worker [13]. The game ends immediately, and the active player wins when the active player manages to get one of their workers to the top of a building of exactly height three. The game can also end if the active player is unable to move [13]. In this case, the active player immediately loses. To

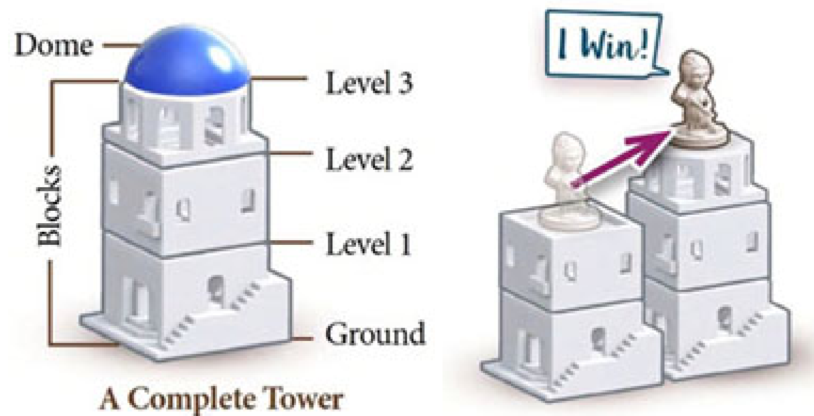


Figure 2.2: A guide to Santorini's building pieces and a depiction of a winning move [14]

move, the active player takes one of their workers and moves it to any of the

adjacent spaces, provided that the space does not already contain a worker and that that space does not contain a building of height four [13].

Figure 2.3: A visualization of a worker's move options [14]

As well, the space that the worker moves to cannot have a height that is more than one greater than the worker's initial height [13]. In practice, this means that a worker can only travel upwards a single unit of height in a single move but may always move to the same height level or jump down to any lower height level in a single move. An important note is that buildings do not

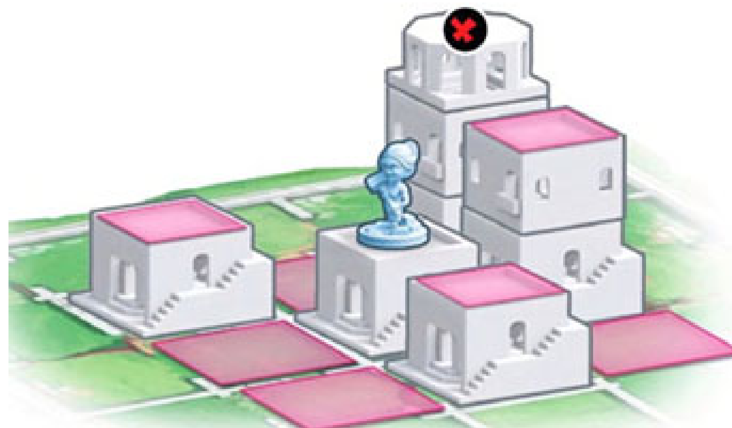

Figure 2.4: A visualization of a worker's move options with an unreachable adjacent space [14]

block movement along a diagonal. Consider this example: A given worker at

height zero is being prevented from moving north or east because the buildings to the north and east are of height two. While playing a physical copy of the game, these buildings will appear to be blocking the ability to move northeast, but, provided that the space to the northeast has a building of height one or less, the northeast space is still a legal move for the worker. It is still possible for a worker to be incapable of movement. If all adjacent spaces (including diagonals) are either occupied (they contain a worker or a building of max height) or have buildings with height at least two greater than the workers current height, than the worker is incapable of movement. A worker cannot be used if it cannot move, and thus it cannot build either [13]. As previously mentioned, if both of the active players workers are incapable of movement, the active player loses. After moving, the active player must then build with the same worker who just moved. To build, the active player chooses one of the adjacent spaces of the moved worker and raises the height of that adjacent spaces building by one. One cannot choose to build upon a space that contains a worker or upon a building of height four [13]. Unlike moving, building is not restricted by height differences. A worker on a space at height zero is allowed to build on empty (does not contain a worker or a building of height four) adjacent spaces, even if the height difference between the worker's space and the build space is greater than one. As a clarification, it is impossible for a worker to be capable of moving and yet incapable of building. If a worker is capable of moving, it will always at least be able to build on the space it just vacated, as that space is guaranteed to be free of workers, not of maximum height, and adjacent to the space the worker moved onto. After building, the turn is over and the other player becomes active.

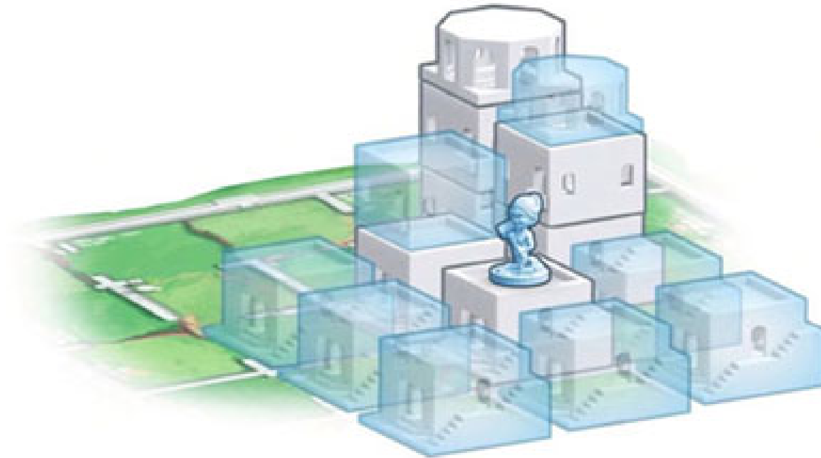

Figure 2.5: A visualization of a worker's build options [14]

Play continues until either a player wins by reaching the required height with a worker on their turn or a player loses because both of their workers cannot move.

*Santorini* is normally played on a $5 \times 5$ grid where any surrounding spaces are adjacent. This could be varied to provide different gameplay. The game could be played on a $2 \times 2$ grid, or an $n \times n$ grid. Even more interestingly, the game could be played with a different underlying structure than a grid. The number of workers and the values for the winning height and the maximum height of buildings could be varied as well. Each of these various factors could dramatically influence the winning strategies and produce different gameplay.

# 3

# Relevant Mathematical Theory

With a firm understanding of the rules, there are now several important mathematical avenues that we must explore to analyze *Santorini*. Graph theory and game theory are two realms with pertinent information that we must study.

## 3.1 Introduction to Graph Theory

Necessary to a more complete understanding of *Santorini* and the board upon which it is played is an understanding of several elementary graph theory concepts. Graph theory is, naturally, the study of graphs, and thus we are brought to our first concept.

**Definition 1.** *A* graph *is a set of points (which are interchangeably referred to as vertices or nodes) connected by lines (which are referred to as edges or arcs) [2].*

Graphs come in many forms, but we are mostly concerned with simple graphs.

**Definition 2.** *A* simple graph *is a graph with vertices where each pair is connected by at most one edge [2].*

There exist multigraphs and pseudographs which allow multiple edges between vertices and self-loop edges between a vertex and itself respectively, but these are not necessary to an understanding of *Santorini* [2]. Two vertices

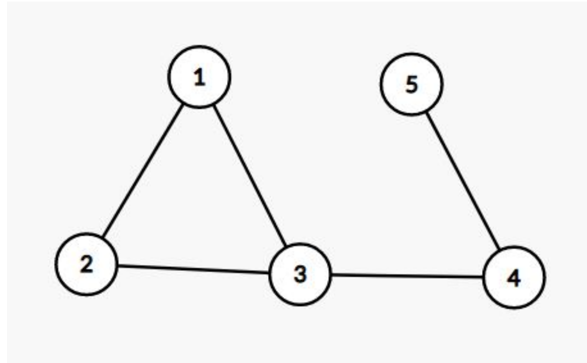


Figure 3.1: A simple graph with 5 vertices

in a graph are said to be *adjacent* if there is an edge connecting them [2]. This is useful because it allows us to turn the board of *Santorini*, a $5 \times 5$ grid of spaces where orthogonal and diagonal spaces are said to be "adjacent," into a graph. If we treat each space as a vertex, then the edges between them represent the adjacencies, and we obtain the grid-like graph in Figure 3.2, which will be henceforth referred to as a *grid-graph*. There are a few other interesting graph concepts that pertain to *Santorini*'s grid-graph playing space.

**Definition 3.** *The* complete graph *on n vertices, denoted $K_n$, is a simple graph in which there is an edge connecting every distinct pair of vertices. [2]*

The structure of the complete graph $K_4$ is very similar to that of *Santorini*'s grid-graph. $K_4$ is a $2 \times 2$ grid graph, and if we affix many copies of it to itself in

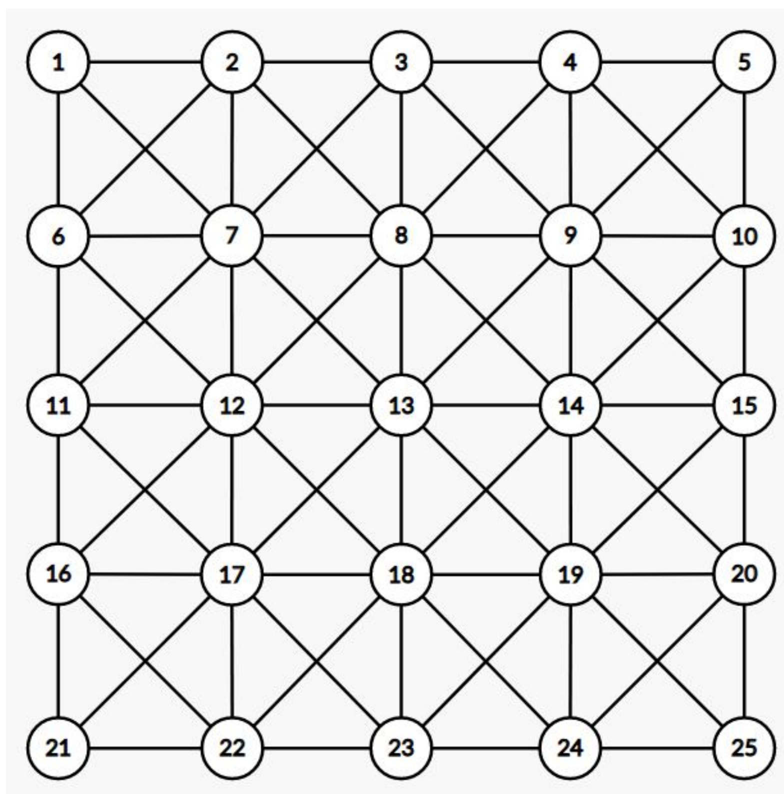


Figure 3.2: The board of *Santorini* in the form of a graph

a grid pattern, we obtain *Santorini*'s grid-graph. As previously mentioned, variants of *Santorini* can be generated by using a different underlying graph to represent the adjacencies between spaces. Relevant to these underlying graphs is the concept of a subgraph.

**Definition 4.** *A* subgraph *of a graph is a graph whose vertex set is a subset of the larger graph's vertex set and whose edge set is a subset of the larger graph's edge set. Each edge in the subgraph must connect vertices in the subgraph [2].*

$K_4$ is a subgraph of the $5 \times 5$ grid-graph that *Santorini* is played on.

Relevant to a later proof is the concept of a matching.

**Definition 5.** *A* matching *of a graph $G$ is a subgraph of $G$ where no two edges share*

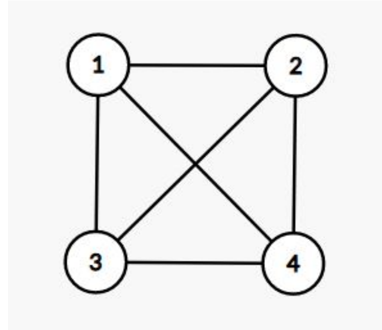


Figure 3.3: The complete graph on 4 vertices, $K_4$

*a vertex [2].*

Matchings can be represented as sets of edges. Consider Figure 4. An edge can be written as $e_{1,3}$, denoting that it connects vertices 1 and 3. The set $e_{1,2}, e_{3,4}$ is a matching of $K_4$, but the edge set $e_{1,3}, e_{3,4}$ is not a matching because the edges share vertex 3.

**Definition 6.** *A matching is a* maximum *matching of a graph G if the matching contains the largest possible number of edges [2].*

There can be many different possible maximum matchings, so long as they all have the same largest number of edges. The *matching number* of G is the size of a maximum matching on G. On $K_4$, the edge set $e_{1,2}, e_{3,4}$ is a maximum matching. Note that since each edge matches two vertices, the matching number for any graph is always less than or equal to half the number of vertices.

## 3.2 Introduction to Game Theory

### 3.2.1 Brief History of the Field

Game theory is a relatively recent addition to the fields under the beautiful umbrella of mathematics. It likely was not until 1928 that it even existed by name, when John von Neumann published *On the Theory of Games of Strategy* [18]. This paper proposed the fundamental theorem of game theory, today known as the Minimax Theorem, which, informally stated, explains that for any zero-sum, two-player game of finite length and complete information, it is always possible to find an equilibrium set of strategies that neither player should deviate from. Game theory's true claim to fame came with von Neumann's following publication in 1944, *Theory of Games and Economic Behaviour* [18]. This text is considered the birth of modern game theory, and the field seemed to explode with life into the 1950s and the following decades.

These following decades saw rise to several big names in game theory, most notably John Horton Conway, Richard K. Guy, and Elwyn Berlekamp. The three collaborated often and introduced the concept of partizan (also written as *partisan*) games in the 1960s, in contrast to the impartial two-player games considered by John von Neumann [18]. Conway published his own field defining classic, *On Numbers and Games* [8], in 1976. This book introduced surreal numbers and their generalization to games. Together, the three published *Winning Ways For Your Mathematical Plays* in 1982, which introduces game analysis techniques and implements them on a variety of games [7].

### 3.2.2 Terminology Relevant to *Santorini*

In their book, *Winning Ways For Your Mathematical Plays*, Berlekamp, Conway, and Guy discuss and analyze a number of games that satisfy all or most of the following eight conditions [7]:

1. There are two players, often called Left and Right.
2. There are several, usually finitely many, positions, and often a unique starting position.
3. There are clearly defined rules that specify the moves that either player can make from a given position to its options.
4. Left and Right move alternately in the game as a whole.
5. Both players know what is going on, i.e. there is complete information.
6. There are no chance moves such as rolling dice or shuffling cards.
7. In the *normal play convention*, a player unable to move loses.
8. The rules are such that play always comes to an end because some player is be unable to move.

Berlekamp et al. outline a clever proof that, for games satisfying these eight conditions, there must exist a winning strategy for either Right or Left. The game of *Santorini* can be viewed in such a way that it complies with the eight conditions, and thus it can be studied in a fashion similar to that used to study the games in *Winning Ways*. Note that although *Santorini* has a win

condition unrelated to a player being unable to move, if we add a simple addendum to the rules with no influence on gameplay, it complies with this condition. All we must say is that a player is unable to move if their opponent has a worker on a building of winning height.

Conditions 5 and 6 are the requirements for a game to be considered combinatorial.

**Definition 7.** *A* combinatorial game *is a two-player game that satisfies the following conditions:*

1. *The game is **deterministic**, meaning that there are no elements of chance or randomization.*
2. *There is **perfect information**, meaning that both players know all information about the state of the game and that nothing is hidden [7].*

The well-known games of chess, Checkers, and Tic-Tac-Toe are combinatorial games. They each are two-player deterministic games with perfect information. A game of Texas Hold 'Em, on the other hand, is not combinatorial as it fails both requirements. In Texas Hold 'Em, the deck of cards is shuffled and dealt randomly, so it is not deterministic. As well, the players' hands and the remaining cards in the deck are kept private, so neither player knows all the information about the state of the game.

*Santorini* is a combinatorial game, as it lacks chance elements and hidden information. Every game that Berlekamp et al. analyze in *Winning Ways For Your Mathematical Plays* is also combinatorial.

As mentioned earlier, the trio of collaborating game theorists Berlekamp,

Conway, and Guy jointly introduced the concept of partizan games in the 1960s. Partizan games are simply games that are not impartial.

**Definition 8.** *An impartial game is one where the set of options from any given position is the same for all players [7].*

As such, partizan games are ones where the players have different sets of options from a given position. For example, *Santorini* is partizan because each player is only allowed to move their own pieces, so the sets of options from the same board state are different. In fact, most two-player games played recreationally are partizan, including chess, Checkers, Go, and Tic-Tac-Toe. Examples of impartial games include the game of Nim, the game of Geography [7], and many other mathematically interesting games.

### 3.2.3 Game Trees

Many of the games studied in *Winning Ways* can be represented effectively with the use of a game tree. The game tree depicts the board state at each node of the tree, and the children of each node hold the board states that are reachable from the parent node through a single move. When constructed in this fashion for a turn-based game, each level of the game tree represents all of one player's options for a given turn. For example, in Figure 3.4, one can see that the root node illustrates the board state on X's turn. The three following child nodes from that root represent the board states resulting from the three moves that X can make. On that second level, any of the given board states would be O's turn, and the following level of the tree shows each of O's legal moves from each of those positions. This figure also depicts values assigned to

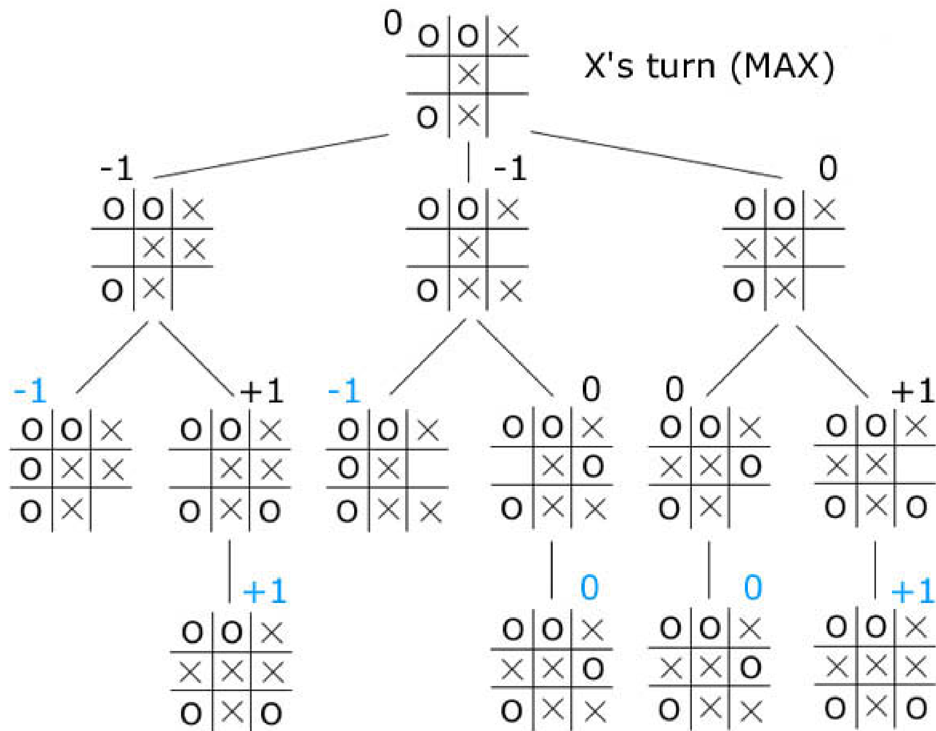


Figure 3.4: A game tree for a particular board state of Tic-Tac-Toe [5]

each position. These values are assigned by giving a value of positive 1 to states which result in X winning and giving a value of negative 1 to states which result in O winning. Draws are given a value of 0. To achieve this assignment across the whole tree, one must start from the bottom of the game tree and work upwards. Positions where the game ends are assigned the relevant value. Then, moving upwards to the previous level, the active player finds either the max or the min (in this case, the max for X, the min for O) and uses that max or min to represent the value of the current space. This is repeated until reaching the top, dictating whether the position is a winning or losing (or in the case of Tic-Tac-Toe, tying) position. In essence, this assignment is a simplified form of the minimax algorithm which originates

from John von Neuman's Minimax Theorem. The algorithm (in pseudocode) is found in Listing 3.1:

Listing 3.1: Minimax Algorithm [1]

```
minimax(player,board)
    {
    if(game_over_in_current_board_position)
    {
        return winner
    }
    children =
       all_legal_moves_for_player_from_this_board
    if(max's_turn)
    {
        return maximal_score_of_calling_minimax_on_all_the_children
    }
    else (min's turn)
    {
        return minimal score of calling minimax on all
           the children
    }
}
```

This algorithm is effective and easy to apply to a small game like Tic-Tac-Toe. Since Tic-Tac-Toe has so few possible board states, this algorithm can quickly evaluate the "score" of any starting position and can predict who will win (or whether they will tie) provided that players play optimally. In fact, the game tree for Tic-Tac-Toe can be memorized and optimally navigated by a few simple principles. This algorithm encounters trouble when we move to more complex games like chess and Go. This is because the depths of the game trees for chess and Go are much, much larger and there are not any rules or procedures to easily simplify them. The computation time is incredibly

high for games with deep game trees. We can't possibly completely search the game tree at the start of a game of chess because, at our current rate of computation, "by the time we finish analyzing a move the sun will have gone nova and the earth will no longer exist" [5]. This is why any real AI only looks a few moves ahead rather than searching to the end of the game tree. For a game as deep as chess where victory is often many moves away, it is important for an effective AI to be able to evaluate the board state so as to select better moves along the way.

# 4

# Game Theory As it Applies to *Santorini*

## 4.1 Generalizing the Gameplay Elements of *Santorini*

*Santorini*'s core gameplay arises from its intuitive rules for moving and building. However, finding a winning strategy in *Santorini* depends heavily on the more contextual elements of the game: the number of workers, the maximum height of the towers, the winning height for which one must move up to, and the shape of the underlying graph structure that defines moving and building. Each of these more contextual elements can be altered to form variants of *Santorini*. Studying these variants can improve our understanding of the base game to allow more reliable evaluation of given board states.

Let us first define a variant of *Santorini* that is impartial. *Impartial Santorini*

is identical to the base game, except that ownership of the four workers is shared. A player still wins if they are able to move onto a tower of the appropriate winning height, and a player still loses if they are unable to move. The only difference is that both players are allowed to make moves with any of the four workers. As such, it becomes remarkably more difficult to set oneself up to win without setting one's opponent up. By similar logic, it is more difficult to trap one's opponent and make them incapable of moving, because doing so also restricts one's own movements.

A second avenue of variance is that of the number of workers. Regular partizan *Santorini* has four workers, but it may be of interest to change this number. Let *n-worker Santorini* be defined as the game played with $n$ workers. In partizan forms, each player has $n/2$ workers, and in impartial forms, the $n$ workers are shared. This allows for 1-worker impartial *Santorini*, a form which bears resemblance to another combinatorial game, Geography, which is discussed more explicitly in Chapter 5.

A third avenue of variance is twofold: that of the maximum tower height and that of the necessary winning height. In terms of maximum height, things are fairly straightforward. Let *n-height Santorini* be defined as the game played with a maximum tower height of $n$. In terms of winning height, we will only be considering variants where the winning height is either null or one less than the maximum height. The in-between cases (where the winning height differs from the max height by two or more) are frequently uninteresting and differ from regular *Santorini* greatly enough that strategies do not translate back effectively. As such, we treat the winning height as a binary variable called *Tower Win*. A *Tower Win* game of *Santorini* allows players to win when

moving onto a tower of one less than maximum height. A *No Tower Win* game of *Santorini* only allows players to win by forcing their opponents into positions where they can no longer move.

An additional line of variance we explore is that of the underlying graph structure. Standard *Santorini* is played on a $5 \times 5$ undirected grid graph. For an underlying directed graph structure, where adjacencies are one-way relationships, we must be a little bit more precise in how we define the building rules. Moving translates to directed graphs as one would expect; a player can only move to an adjacent node following the directed edges of the graph (and obeying the standard moving rules with regards to height restrictions). Building on a directed graph is slightly counter-intuitive; a player can only build on nodes that they could have moved from to reach their current node. This mimics regular *Santorini*, where one is able to build on any of the nodes adjacent to one's current node, regardless of their height, as one could have moved from them. As such, when building in directed *Santorini*, one must follow along the edges opposite of their direction. Alternatively, there exists a building stipulation for directed graphs that makes *Santorini* more closely match the game of Geography. This building stipulation requires that only the vertex that was just moved from be built on. As such, the inclusion of this building stipulation is another variation on directed graphs.

A final variable rule in *Santorini* is the conditional movement based on height. Regular *Santorini* requires that a piece's destination cannot be more than one height taller than the piece's initial space. We can relax this requirement and allow pieces to move to any adjacent, empty (not containing a worker or a tower of max height) space as an additional variant of *Santorini*.

For future reference, we represent any given variant of *Santorini* by the abbreviation $nW_imH_jkS$, where $n$ is the number of workers (W), $i$ is either $I$ or $P$ dictating either impartial or partizan, $m$ is the maximum height (H), $j$ is either $TW$ or $NTW$ dictating the binary tower winning condition Tower Win or No Tower Win, $k$ is either $D$, $D^*$, $U$, or $U^*$, representing directed graphs, directed graphs with the building stipulation, undirected graphs, and undirected graphs with the building stipulation, and $S$ stands for *Santorini*. We also allow for $S_{FM}$ which refers to *Santorini* with free movement, meaning that the height restriction on movement is removed. As such, regular *Santorini* would be abbreviated as $4W_P4H_{TW}US$, whereas directed, impartial, one worker, two height *Santorini* without tower wins, with free movement, and with the building stipulation for directed graphs would be abbreviated as $1W_I2H_{NTW}D^*S_{FM}$.

## 4.2 Brute-Forcing Small Cases

This section results from fully exploring the game trees of variants of *Santorini* on select small graphs and determining which player can force a win in each case. Several of the simplified variants of *Santorini* are easy to solve in certain circumstances.

For example, $nW_ImH_{TW}US_{FM}$ is an easy game. This is undirected, impartial *Santorini* with Tower Wins and free movement, and the first player to build to height $m-1$ (the winning height) immediately loses as their opponent can instantly move atop the freshly constructed tower. This simplicity is true regardless of the graph that is being played on, the number

of workers that are shared (unless it is equal to the number of vertices in the graph), or the maximum height. This same principle applies to $nW_I2H_{TW}US$ on any graph as well, as the first player to build simply loses. These two cases apply to partizan *Santorini* on complete graphs as well. For $nW_P2H_{TW}US$ on complete graphs, Left always loses, and for $nW_PmH_{TW}US_{FM}$ on complete graphs, the first player to build to height $m-1$ loses.

Cycle graphs result in fairly simple games as well. $1W_I1H_{NTW}DS$ on a cycle graph is won based on the parity of the number of nodes in the cycle. Left wins even cycles, Right wins odd cycles. Note that the building stipulation and the free movement rule have no affect on games played on directed cycle graphs. If we increase the maximum height to an even value, then play loops over the cycle an even number of times, turning odd cycles effectively into even cycles so that Left always wins. If we allow tower wins on cycle graphs then the win conditions simply flip. Right wins even cycles, Left wins odd cycles, and Right always wins at odd maximum heights (heights where the winning height becomes even).

Two worker partizan *Santorini* games on directed cycle graphs are also easy to resolve. For max height 1, Right wins so long as they start on a node less than or equal to the $\lfloor \frac{n}{2} \rfloor + 1$ node where $n$ is the number of nodes in the cycle and the first node is where Left starts. Increasing max height flips the win conditions for each increase of one, and adding tower wins at the same max height maintains the status quo.

For example, in Figure 4.2, for $C_5$, Right wins when starting at nodes 2 and 3 for odd maximum heights both with and without tower wins. For the 6 cycle $C_6$, Right wins under the same conditions except that they can also start at

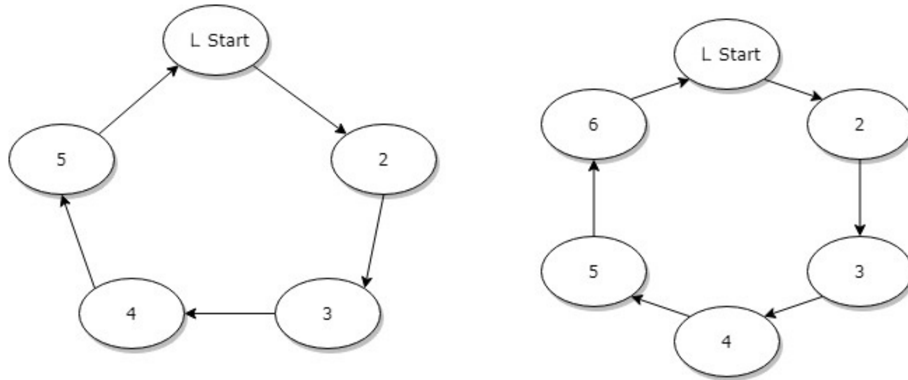


Figure 4.1: The cycles on 5 and 6 nodes, $C_5$ and $C_6$

node 4.

Any variant of *Santorini* on directed paths is also easy. At max height greater than 1 (with or without tower wins), the player further from the end of the path wins. At max height 1, the player further from the end of the path will potentially be impeded by the builds left behind by the player closer to the end of the path. In that instance, we must compare the closer player's distance to the end of the path with the further player's distance to the closer player's starting position. Whoever has the larger distance in this instance wins (taking turn order into account).

# 5

# Introduction to Complexity Theory

As stated by Weizmann Institute of Science professor Oded Goldreich, "Complexity theory is concerned with the study of the *intrinsic complexity* of computational tasks" [12]. The field of study aims to determine the complexity of given tasks but also to compare complexities and understand the relations in complexity between various computational phenomena. There has been little success in discovering absolute answers for the complexity of specific computational phenomena, but there has been significantly more success in identifying relations between computational phenomena [12]. In short, the field has struggled to make definitive statements about a given phenomenon's complexity alone, but it is capable of making statements about the phenomenon's complexity relative to another phenomenon.

Consider the Boolean satisfiability problem, often simply called SAT. This problem simply asks if the variables of a given Boolean formula can be assigned with the values TRUE and FALSE to make the overall formula evaluate to TRUE. A formula that has such an assignment is called *satisfiable*,

whereas a formula without one is called *unsatisfiable*. For example, the formula $a \wedge \bar{b}$ is satisfiable with the assignment that $a = TRUE$ and $b = FALSE$. However, the formula $a \wedge \bar{a}$ is unsatisfiable because there does not exist an assignment that makes the formula evaulate to TRUE. In general, the overall problem asks whether or not a given Boolean formula is satisfiable. This task is of some computational difficulty. While the field does not have an answer as to the specific and absolute value of the complexity of this task, it is able to compare its complexity to other computational tasks, which, in a grander scheme of things, might be more informative. For example, SAT has been proven to have the same (in a sense) computational complexity as the problem of determining whether the vertices of a graph can be colored red, green, and blue so that no two adjacent vertices have the same color, called a *proper 3-coloring* [12]. As such, SAT and the 3-coloring problem are considered to be of the same complexity.

Both the 3-coloring problem and the Boolean satisfiability problem are decision problems, meaning that they have a binary output [6]. For the 3-coloring problem, the input is a given graph and the output is *yes* if the graph has a three-coloring, and *no* otherwise. The Boolean satisfiability problem takes a Boolean formula as an input and gives an ouput based on whether or not it can be satisfied. Decision problems can also be represented as formal languages. Inputs to the problem are encoded using an alphabet of symbols, where the inputs that produce the *yes* output are considered members of the language, and those inputs that produce the *no* output are not members. Some kind of algorithm is used to determine whether the input is accepted or not, and we can analyze and make judgments about those

algorithms in order to compare the relative complexities of various problems. Necessary to understanding the various complexity classes that arise from this trail of analysis is an understanding of how we compare the functions that define our algorithms.

## 5.1 Bachmann-Landau Notation

Necessary to understanding these function comparison tools is an understanding of suprema and infima. Suprema are least upper bounds, and infima are greatest lower bounds. Formally, supremum is defined as follows:

**Definition 9.** *An upper bound of a subset $A$ of a $\mathbb{R}$ is an element $x \in \mathbb{R}$ such that $x \geq a \forall a \in A$. An upper bound $x$ of $A$ is a supremum if for all upper bounds $y$ of $A$ in $\mathbb{R}$, $x \leq y$.*

The definition of infimum is analogous for lower bounds, being the greatest of the lower bounds for a set. With this understanding of bounds, we can approach Bachmann-Landau notation.

Bachmann-Landau notation, more commonly known as big-$O$ notation, is a relation on functions that is used as a theoretical measure of the execution of an algorithm (in terms of run-time or memory space) for a given problem of size $n$ [16]. Formally, it is defined as follows:

**Definition 10.** *[16] $f(n) = O(g(n))$ if $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$*

An equivalent definition that is more common among computer scientists is $f(n) = O(g(n))$ if there exists $c > 0$ and there exists $N$ such that for all

$n > N$, $f(n) \leq c \cdot g(n)$ [16]. In practice, this notation sets upper bounds on the growth of a function, and is used to describe the efficiency of algorithms. It is relatively easy to compute the simplest big-$O$ form of a given function. One only needs to consider the fastest growing term and can drop any attached constants [16]. For example, the function $f(n) = 3n^2 + 2n$ is of similar order to $g(n) = n^2$, and thus it is expressed that $f(n) = O(n^2)$. Specifically, we can prove that $3n^2 + 2n$ is $O(n^2)$. We must simply find a constant $c$ and a specific $n$ beyond which $cn^2 \geq 3n^2 + 2n$. If we let $n \geq 1$, then for $c = 5$, $cn^2 \geq 3n^2 + 2n$. Thus $3n^2 + 2n$ is $O(n^2)$. This ability to drop constants has an additional implication that may not be immediately apparent. It allows us to also ignore the base of any logarithms. For example, consider the following: $f(n) = \log_2(n) = \frac{\log_{10}(n)}{\log_{10}(2)} = \left(\frac{1}{\log_{10}(2)}\right)\log_{10}(n)$. Since $\frac{1}{\log_{10}(2)}$ is constant, $\log_2(n)$ and $\log_{10}(n)$ differ by a constant factor and are thus big-$O$ of each other. Thus, when simplifying a function to its bound, logarithm bases can be completely disregarded, and both $\log_2(n)$ and $\log_{10}(n)$ are simply referred to as being $O(\log(n))$.

Big-$O$ notation is a part of a larger family of asymptotic notation invented and improved upon by mathematicians Paul Bachmann, Edmund Landau, and Donald Knuth [16]. This notation family also includes $o(g(n))$, $\Omega(g(n))$, $\omega(g(n))$, and $\Theta(g(n))$, each of which is defined as follows:

**Definition 11.** *[16] $f(n) = o(g(n))$ if* $\limsup\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Little-$o$ serves as a stricter requirement than big-$O$, requiring not only that $f$ be bounded above asymptotically by $g$, but that $f$ be dominated by $g$ asymptotically. The difference between big-$O$ and little-$o$ is similar to the

difference between ≤ and <. For example, the function $2n^2$ is both $o(n^3)$ and $O(n^3)$, but it is $O(n^2)$, but not $o(n^2)$.

**Definition 12.** *[16]* $f(n) = \Omega(g(n))$ *if* $\liminf_{n\to\infty} \frac{f(n)}{g(n)} > 0$

Big-Omega is the corresponding form of big-$O$ from below. It states that $f$ is asymptotically bounded below be $g$. As such, the function $2n^2$ is $\Omega(n^2)$ but also $\Omega(n)$.

**Definition 13.** *[16]* $f(n) = \omega(g(n))$ *if* $\liminf_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Little-omega is the parallel form of little-$o$, requiring that $g$ be dominated by $f$. Thus the function $2n^2$ is $\omega(n)$ and yet is not $\omega(n^2)$.

**Definition 14.** *[16]* $f(n) = \Theta(g(n))$ *if* $f(n) = O(g(n))$ *and* $f(n) = \Omega(g(n))$

$\Theta$ is an equivalence relation and requires that $f$ be bounded asymptotically both above and below by $g$. Our example function $2n^2$ is thus $\Theta(n^2)$, and it is not $\Theta(n)$ or $\Theta(n^3)$.

## 5.2 Complexity Classes

A complexity class is defined by a model of computation, a resource, and a function (the complexity bound) for that resource [6]. We are concerned with the complexity classes defined by a machine-based model of computation, that of the Turing machine [3]. The Turing machine is a hypothetical device imagined by Alan Turing in 1936 [17]. The Turing machine presented here (there are many equivalent variants) consists of an infinitely long piece of tape upon which data is stored and a head which highlights a single square of the

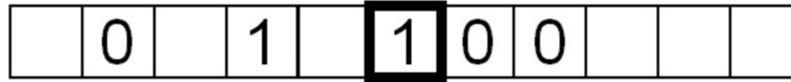


Figure 5.1: A representation of the tape of a Turing machine. The location of the head of the machine is bold [17]

tape [17]. The Turing machine is capable of three different operations. It can read the symbol under its head, write a new symbol or erase the symbol currently under its head, or it can move the tape beneath it left or right one square so it can read and write on a neighboring square [17]. The machine simulates computation by following a set of instructions depending on its state and on what symbol is read [17]. The machine is not always limited to the symbols 1 and 0, but it is frequently of interest to consider the computations capable by a Turing machine of a set number of symbols. Despite its simplicity, given enough states, time, and tape, the Turing machine is capable of fully simulating any computer algorithm [17]. Complexity classes defined by a machine-based model of computation are defined in relation to the two fundamental resources of the Turing machine: time and space. The time used by a Turing machine is reflected in the number of operations (read, write, move) it must do in order to complete a computation. The space used by the Turing machine is reflected in the number of squares on the infinite tape that it must use to store data to complete a computation. P is the class of formal languages such that an input of size $n$ can be solved (found to be either a member of the language or not) by an algorithm in time $O(n^k)$ for some fixed $k$ [6]. As such, it is said that problems in this complexity class can be solved in polynomial time. NP is nondeterministic polynomial time, and it is the class of

problems for which a potential solution can be shown to be correct in polynomial time, regardless of whether there exists a deterministic method for generating solutions [6]. SAT and the 3-color problem are both in NP, and they are in fact both NP-Complete, a property that we discuss later [12]. A natural deduction from these definitions is that $P \subseteq NP$. Given a problem that is solvable in polynomial time and thus in P, a solution can be verified in polynomial time by simply solving the problem. This means that the problem is in NP, and thus that $P \subseteq NP$ [6]. One of the most important questions in complexity theory deals with the question as to whether $P = NP$. If true, this would indicate that any problem with a solution verifiable in polynomial time would be fully solvable in polynomial time [6]. This problem is one of the seven Millennium Prize Problems proposed by the Clay Mathematics Institute and has a million dollar reward attached to proving it one way or the other [6]. It is widely believed that $P \neq NP$, and it seems to follow some intuitive logic that checking if a solution is correct is easier than finding the solution from scratch in the first place.

An additional complexity class can be defined in terms of the space used by a deterministic Turing machine. This class, PSPACE, is the class of formal languages such that an input of size $n$ can be solved by an algorithm using space $O(n^k)$ for some fixed $k$ [6]. Complexity classes go beyond PSPACE. There exists the class EXP (or EXPTIME) which solves problems in $2^{O(n^k)}$ time [6]. Similarly, EXPSPACE solves problems in $2^{O(n^k)}$ space [6]. Between them is NEXP (or NEXPTIME), which is analogously related to EXP as NP is to P [6]. Encompassing all of these is the class of decision problems that are decidable at all, which are proven to have an algorithm that provides an answer [4].

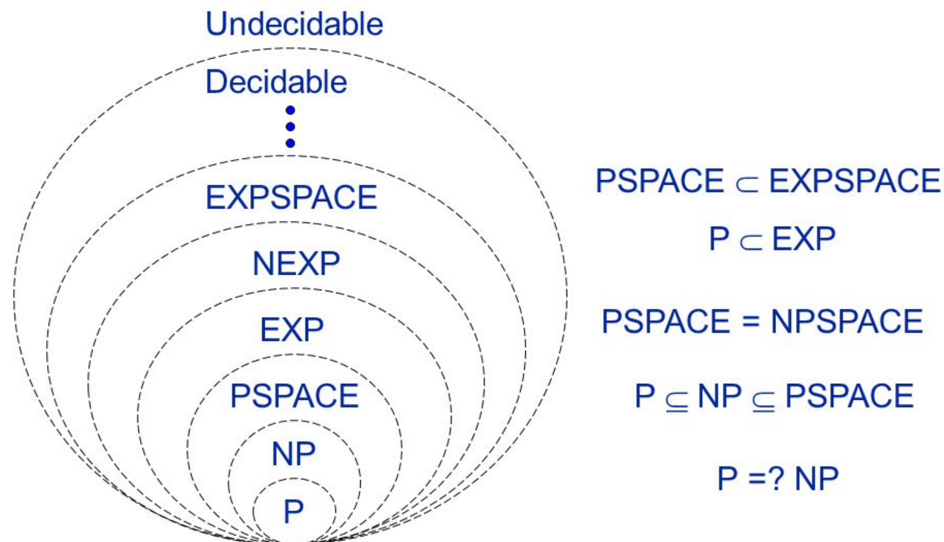


Figure 5.2: An illustration of the hierarchy of the complexity classes [4].

Obviously beyond that is the class of decision problems that are undecidable, where it has been proven that it is impossible to construct an algorithm that returns an answer [4]. For the purposes of this thesis, we need only concern ourselves with P, NP, and PSPACE. It has been proven that $P \subseteq NP \subseteq PSPACE$, but it is still even possible that P = PSPACE [4]. This would of course imply that P = NP and is thus even less likely to be true.

## 5.3 Reductions

An important accompaniment to the concept of complexity classes is the notion of reducibility. Essentially, a problem Q can be reduced to the problem

Q' if any instance of Q can be "easily rephrased" as an instance of Q' [9]. When that rephrasing is possible, the solution to the instance of Q' provides the solution to Q [9]. For example, the problem of solving linear equations can be reduced to the problem of solving quadratic equations. The generalized instance $ax + b = 0$ can be re-imagined as $0x^2 + ax + b = 0$, and the solution to the quadratic equation provides the solution to the linear one [9]. As such, when a problem Q is reduced to a second problem Q', it is insinuated that the first problem is not any harder to solve than the second problem (as long as the reduction is sufficiently simple). In terms of formal languages, it is said that a language $L_1$ is *polynomial-time reducible* to a language $L_2$ (written $L_1 \leq_P L_2$) if there exists a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$ [9]. Polynomial-time reductions provide the framework for comparing the difficulties of different problems.

## 5.4 Completeness

The notion of completeness is important to complexity classes in that it provides an understanding of equality of difficulty between various problems. Informally stated, for a given complexity class $C$, a problem $Q$ is said to be $C$-complete if the problem $Q$ is in $C$ and the problem $Q$ is as "hard" as any problem in $C$. A problem satisfying only the second requirement is said to be $C$-hard. Using the framework of reductions, we can compare the difficulties of problems and thus make statements about the second requirement for completeness. If there exists a polynomial-time reduction from some initial

problem $Q_i$ (that has already been shown to be $C$-hard) to the problem $Q$, then it can be said that $Q$ is $C$-hard [9]. Because the technique of reduction relies on having a problem already known to be $C$-hard in order to prove that a different problem is $C$-hard, it is necessary to have an initial problem that is known to be $C$-hard. The complexity classes NP and PSPACE have known natural complete problems that are used in any such proof [9].

The known natural complete problem for NP is given by the Cook-Levin theorem, which states that any problem in NP can be polynomial time reduced by a deterministic Turing Machine to the previously mentioned Boolean satisfiability problem (SAT) [11].

The quantified Boolean satisfiability problem (QSAT) is a generalization of the Boolean satisfiability problem (SAT). In this generalization, existential and universal quantifiers can be applied to each variable [15]. For example, the following is a quantified boolean formula: $\exists x \forall y (x \vee \bar{y}) \wedge (\bar{x} \vee y)$. This formula asks if there exists a value for $x$ such that for all values of $y$ the formula is true. Whether a fully quantified Boolean formula is true or not is the prototypical complete problem for PSPACE. Any fully quantified Boolean formula can be rearranged into prenex normal form in polynomial time, which has all the quantifiers at the front and has them alternate between existential and universal [15]. This takes the form of $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \mathrm{Q}_n x_n \Phi(x_1, x_2, \ldots, x_n)$. There additionally exists a polynomial time reduction which shows that satisfying a three conjunctive-normal form (3CNF) Boolean formula is PSPACE-complete, meaning that problems can be reduced to the 3CNF QSAT problem to prove PSPACE-completeness. A formula in 3CNF is a conjunction of *clauses* where each clause is a disjunction of three *literals*. For $x_1, x_2, x_3, x_4$, a

formula in 3CNF might be $(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee \bar{x_3} \vee x_4)$.

# 6

# Proving P-SPACE Completeness

## 6.1 QSAT Game and the Game of Geography

There exists a two-player game interpretation of QSAT, where players A and B alternate turns assigning values to the variables. Player A assigns the value of the existentially quantified variables and player B assigns the value of the universally quantified variables. Player A wins if the formula is true, meaning that the there is a strategy for assigning the existential variables such that the formula is always satisfied. Player B wins if the formula is false. The formal language that decides if player A wins the game is PSPACE-complete [15].

With a gamified version of the problem, it is one step closer to a reduction to *Santorini*. An intermediate step is through a reduction to Generalized Geography. The original game of Geography is played by two players naming cities in the world. Play alternates, and each player must name a different city in the world that begins with the same letter that the previously named city ended with. The player unable to name such a city loses. This game (which

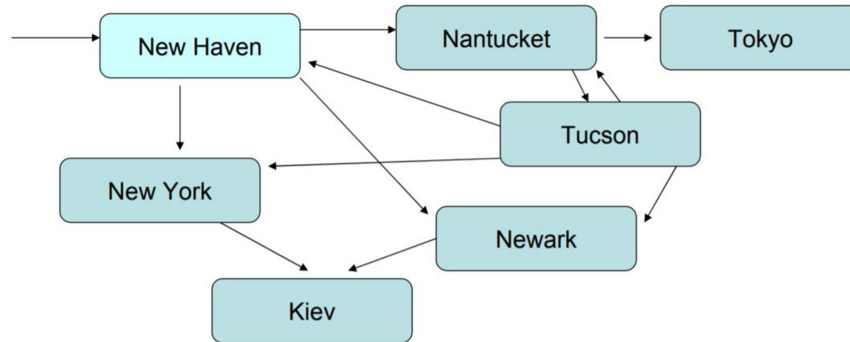


Figure 6.1: The graph of the original game of Geography being played with city names [11].

would usually end to due a player's lack of knowledge) can be represented by a directed graph. Each node is a city, the last-letter to first-letter one way relationship is depicted by the directed edges, and each node is deleted after it is visited. In the generalized graphical form of the game, the player unable to move to a new node loses.

## 6.2 Proof that Geography is PSPACE-hard

**Theorem 1.** *[11] The game of Generalized Geography is* PSPACE-*hard.*

*Proof.* We will show the polynomial-time reduction from the quantified boolean formula game to the game of Generalized Geography. The quantified boolean formula can be assumed to be in prenex normal form, and the interior formula $\Phi(x_1, x_2, \ldots, x_n)$ can be assumed to be in 3CNF. At this point, we construct a directed graph for the game of Geography that emulates the quantified boolean formula game. Each diamond structure in the graph represents a player choosing the truth value of a variable. Players alternate

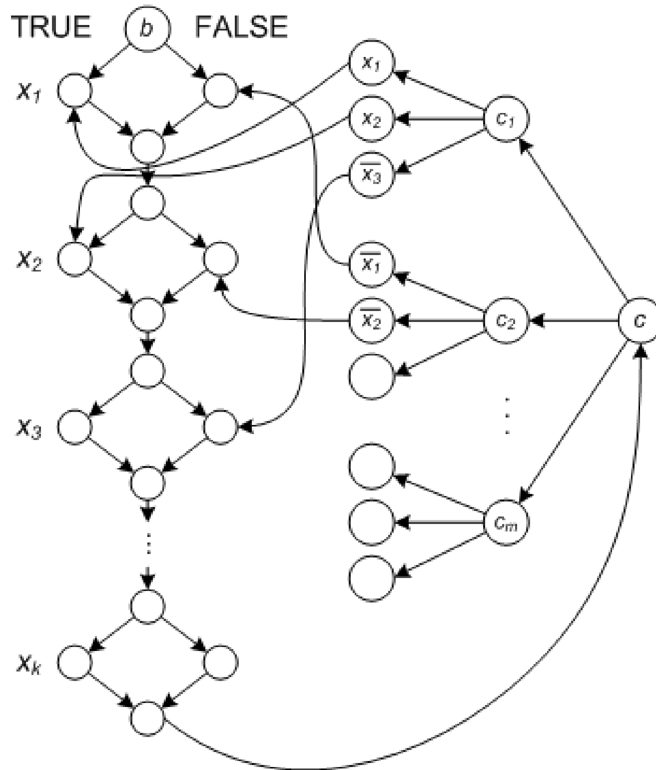


Figure 6.2: A graph for a game of Geography that has an equivalent winning strategy to the quantified boolean formula game [11]

these choices. At the bottom of the graph, each variable has been dictated, and we move to the second portion of the quantified boolean formula game. In this portion, Player 1 wins if $\Phi(x_1, x_2, \ldots, x_n)$ is true. Since $\Phi$ is in CNF, it is composed of conjunctions of a number of clauses. As such, Player 1 wins only if every clause is true. A clause is composed of disjunctions of literals. If any of the literals are true, the clause is true. Thus, player 1 wins if there is at least one true literal in every clause. Player 2 wins if there is a single clause in which all the literals are false. This is exactly represented by the right side of the graph. Player 2 makes the choice from the node $c$ to a node $c_i$. The $c_i$ nodes represent the clauses. Each clause is connected to nodes representing its literals. Player 1

thus makes a choice of node literal to move to. Each node literal points back to the corresponding variable from the left diamond structure of the graph. As nodes are deleted after they are visited, if a variable was selected as true then any corresponding literal on the right side of the graph is a terminal node. Thus Player 1 wins if they are able to select a node literal that was established as true in the first part of the game, as Player 2 will be unable to move. If Player 2 can choose a clause where all literals are false, then Player 1 will not be able to select a node literal that is a terminal node, and thus Player 2 will be able to make an additional move after Player 1's selection which then results in Player 1 being unable to move. As such, Player 2 wins the game of Geography if there exists a clause for which all the literals are false. Player 1 only wins the game of Geography if every clause has at least one true literal, because Player 2 is unable to select a clause with only false literals. This exactly corresponds to the winning conditions of the formula game, and thus there is a polynomial time reduction from the formal language that decides who wins the formula game to the formal language that decides who wins the game of Geography. Therefore, Generalized Geography is *PSPACE*-hard. □

## 6.3 Bridging the Gap to *Santorini*

Our goal is to perform a similar reduction for *Santorini*, taking a specific instance of *Santorini* and showing that the winning strategy for player one also serves as a winning strategy for player one in QSAT. If necessary, this can be a sequence of reductions, from QSAT to Geography and then to *Santorini*. In pursuit of this goal, we make reductions to variants of Geography that

incorporate gameplay elements from *Santorini*. Consider that the game of Geography is identical to one worker, impartial, one max height, No Tower Win, directed *Santorini*, denoted $1W_I1H_{NTW}D^*S$. As such, we want to explore the Geography reduction by altering the aforementioned gameplay elements to bring Geography closer to the game of *Santorini*. If we change the maximum height to 2 and allow tower wins, then the reduction remains nearly identical. As the shared piece progresses through the initial diamond chain structure, nodes of height one are left behind (rather than effectively being deleted). When play loops around back to the initial node literals after Player 2 has chosen the clause and Player 1 has chosen the node literal, the player who normally would have lost (due to being trapped) instead wins because they are able to move onto a tower of height one. This results in exactly flipping the win condition, which can be easily flipped back by adding a buffer node at the end of the loop that reverts the turns and ensures that Player 1 wins only in the appropriate context. Reductions stem from a specific instance of one problem being able to solve any form of the other problem. As such, this maximum height/Tower Win variant of Geography can be produced at any max height, provided that we assume that all the nodes start at max height minus two, and that the winning height is maximum height minus one. This means that the game of $1W_I4H_{TW}D^*S$ is *PSPACE*-hard.

Consider another variant: two worker partizan Geography. In this variant of Geography, players still win when their opponent can no longer move, vertices are still deleted after they are moved from, but each player controls their own worker and a single node cannot be shared by multiple workers. This form of Geography is equivalent to $2W_P1H_{NTW}D^*S$. The graph structure

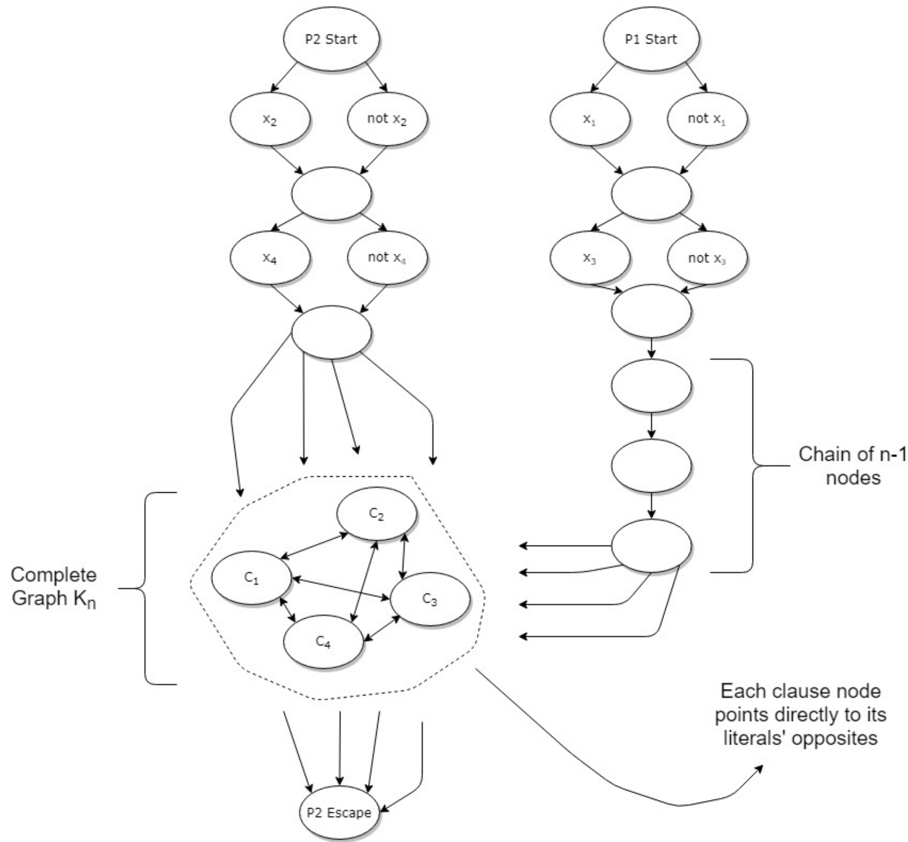


Figure 6.3: A graph for a game of partizan Geography that has an equivalent winning strategy to the quantified boolean formula game

for partizan Geography holds several key differences from the regular Geography reduction. Each player has their own diamond chain structure to allow for the assignment of the boolean variables. This assignment is still done in a proper turn sequence. At the bottom of these diamond chains is a joining mechanism that allows Player 2 to select a single clause that Player 1 must then move through. For $n$ clauses, there is a subgraph of $n$ nodes that forms the complete graph $K_n$. The bottom of player 2's chain connects to each of the $n$ nodes in the complete subgraph. Player 1 has a chain of $n-1$ nodes at the bottom of their diamond structure that delays their entry into the complete

subgraph by $n - 1$ turns, giving Player 2 the opportunity to eliminate all of the $n$ clauses except one. This is equivalent to choosing which clause Player 1 must move through. The timing of this is tight. Immediately after Player 2 moves onto the second to last clause node, Player 1 is forced to move onto the remaining clause node. Each clause node in the complete subgraph points to an escape node that is intended for Player 2. Each clause node also points directly to its literals' opposite nodes in the diamond structures. For example, a clause $x_1 \vee \bar{x_2}$ points to $\bar{x_1}$ and $x_2$ nodes. At this juncture, it is Player 2's turn and both Player 1 and Player 2 are on clause nodes. Player 2 must then use the labeled escape node or lose. If they choose to go to one of their clause's literals' opposite nodes, then Player 1 can use the escape node and Player 2 will be trapped first. This self-entrapment happens similarly if Player 2 attempts to exit the complete subgraph before selecting all clause nodes except one. After Player 2 moves to the escape node, Player 1 loses if all the literals for their clause have been chosen to be false (because then all of his clause's literals' opposite nodes in the diamond structures will have already been deleted). Player 1 wins if any have been chosen to be true, because then they can move to the appropriate opposite node which has not been deleted and Player 2 is subsequently trapped on their escape node. As such, Player 1 wins if there exists an assignment so that each every clause has at least one true literal. This exactly corresponds to the winning conditions of the formula game, and thus partizan Geography ($2W_P1H_{NTW}D^*S$) is *PSPACE*-hard.

Let us now discuss the building stipulation. We can remove this stipulation by allowing a worker to build on any node it could have moved from to reach its current node. For the game of Geography, this translates to

deleting any of the nodes that point to the node one moved onto, not always the node one moved from. This is essentially following the directional edges backwards for building (deleting) purposes. At first glance, this appears to throw a wrench into the regular Geography reduction. If one can delete any node that points to the node one moved onto, then one can prematurely delete nodes necessary for the second portion of the game. The variable $x_i$ nodes are

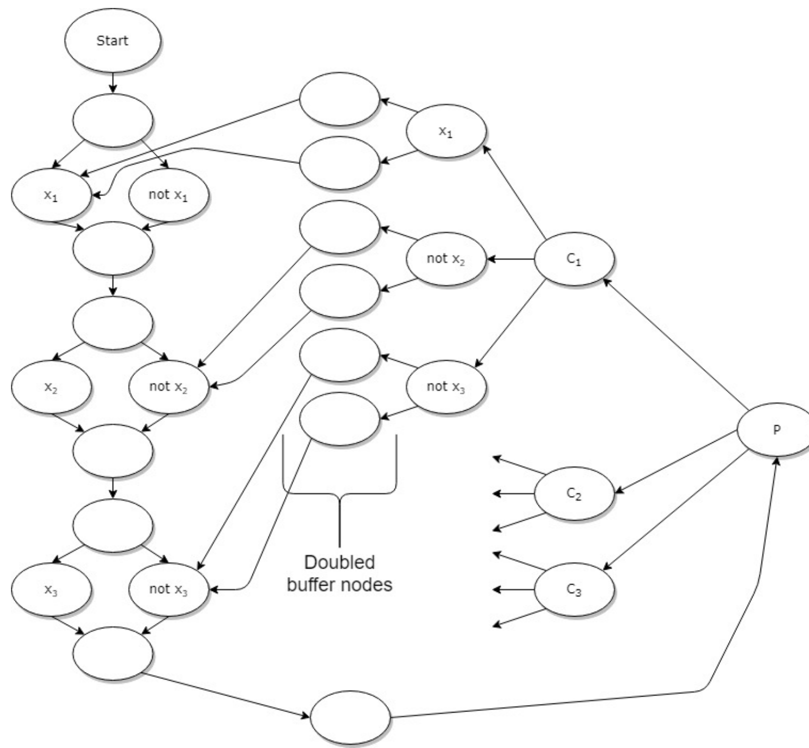


Figure 6.4: A graph for a game of Geography with *Santorini*'s building rules that has an equivalent winning strategy to the quantified boolean formula game

pointed to by the clause nodes in regular Geography. This means that when moving onto a variable node in the assignment phase of the game in the diamond structure, one could delete a clause node. A second problem is that the truth assignment decision is delayed by a turn. When Player 1 moves onto

a variable node initially, their choice is irrelevant because the deletion occurs when Player 2 moves to the bottom of the diamond and gets to choose which variable node to delete (as they could have come from either). This second problem is easily rectified by two simple parity switches possible through buffer nodes: one at the start of the game and one before the clause selection phase. The first problem is rectified by introducing a buffer chain between the final variable selection nodes and the assigned variable nodes from the first phase. This buffer has doubled nodes that each point to the assigned variable to prevent all paths from being deleted, ensuring that the games plays out as normal. The decisions and win conditions thus remain identical to the regular Geography reduction, and thus $1W_I1H_{NTW}DS$ is *PSPACE*-hard.

We can extend this removal of the building stipulation to partizan geography as well with slight modifications to the proof. The game begins as normal: each player has a chain of diamonds that is used to decide the truth values for the literals. Similar to the previous building stipulation proof, that actual decision of truth assignment is not made while moving from the top of the diamond but rather when moving onto the bottom of the diamond and building on one of the two previous nodes. It is also necessary for each variable chain (depicted on the right) to have a doubled end to prevent players from restricting access to the node literals, similar to the doubled ends in the previous proof. Player 1 once again has a delay chain of $n-1$ nodes, where $n$ is the number of clauses, preventing them from picking a clause node before Player 2 has eliminated all except one. Unlike the previous partizan Geography proof, Player 2 does not have a complete graph to move through, and instead has a chain of similar length to Player 1's. Each clause node at in

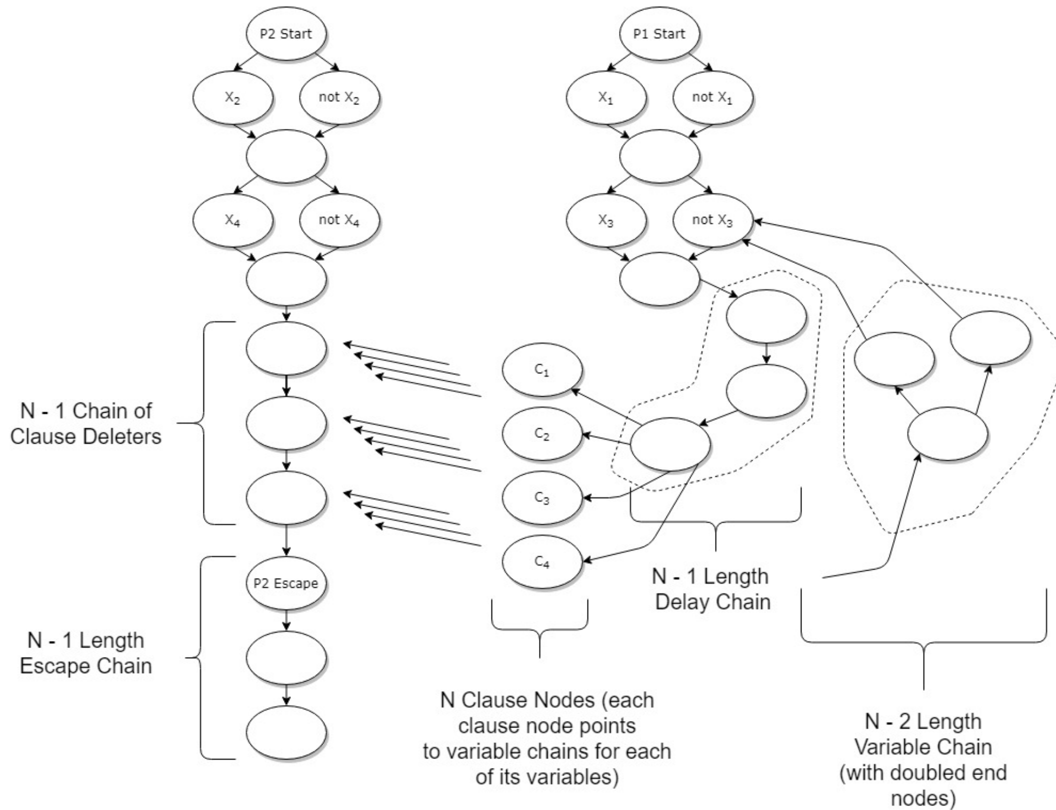


Figure 6.5: A graph for a game of partizan Geography with *Santorini*'s building rules that has an equivalent winning strategy to the quantified boolean formula game

the fan at the end of Player 1's delay chain points to each node in Player 2's clause deletion chain. This allows Player 2 to build on the clause nodes as they move through the clause deletion chain, without actually being able to reach the clause nodes. Player 2 can choose not to build on the clause nodes and to instead build on the vertices in their chain, but this only decreases their chances of winning by allowing Player 1 more clauses to choose from, and thus will never be done. At the end of Player 2's clause deletion chain, Player 1 steps onto the remaining clause (effectively chosen by Player 2) and Player 2 moves onto their escape chain. The escape chain is also of $n - 1$ length to

prevent Player 1 from moving onto Player 2's clause deletion chain from their clause node, as the length of the escape chain guarantees that Player 1 will run out of moves first if they attempt this. Due to this necessity, each variable chain must be of length $n - 2$ to ensure the proper victor after a variable is selected. Despite the figure's limitations, each clause node has variable chains leading to each node literal of the clause. With all of these modifications, the players once again decide back and forth the truth values of the literals, Player 2 selects a clause, and then Player 1 selects a variable and wins if the variable is true. Thus the partizan form of Geography with the building stipulation ($2W_P1H_{NTW}DS$) is PSPACE hard.

Potentially the largest gulf in gameplay to overcome between Geography and regular *Santorini* is the jump from directed graphs to undirected graphs. *Santorini* is strange in that is normally played on a grid-graph where edges between nodes are conditionally directional. The grid-graph is, for the most part, undirected, but this changes as the heights of the nodes change. When the heights of two nodes differ by more than one, the edge between them becomes directed (from the higher node to the lower node, of course). Implementing this element is troublesome for the Geography reduction, as a large part of the proof is predicated on the fact that the workers must travel along a set path and cannot go backwards. Even more troubling is the fact that undirected Geography is in $P$. In a game of undirected Geography (denoted as $1W_I1H_{NTW}U^*S$), Player 1 has a winning strategy if and only if every maximum matching of the graph covers the starting node [11]. The following is a proof by contradiction.

*Proof.* [11] Suppose that $M$ is a maximum matching of $G$ that covers $v$, the current vertex. Player 1's strategy is to move along an edge in $M$, which is guaranteed to be possible because $M$ covers $v$, so an edge in the matching has $v$ as one of its vertices. If Player 1 were to lose, then there would be a sequence of edges $e_1, f_1, e_2, f_2, ..., e_k, f_k$ such that $v \in e_1$, $e_i \in M$, $f_i \notin M$, and $f_k = (x, y)$ where $y$ is the current vertex and it is not covered by $M$. The $e$ edges are Player 1's moves and the $f$ edges are Player 2's moves. This essentially is presuming that there must exist a sequence of moves such that it is Player 1's turn at a vertex that is not covered by $M$. If this sequence of moves exists however, then if we let $A = e_1, e_2, ..., e_k$ and $B = f_1, f_2, ..., f_k$, then $(M - A) \cup B$ is a maximum matching of $G$ that does not cover $v$. This is a contradiction as all maximum matchings are assumed to cover $v$.

Suppose now that $M$ is a maximum matching but it does not cover $v$. Player 1's move is $(v, w)$, and $w$ is guaranteed to be covered by $M$ (otherwise $M$ is not a maximum matching, as the edge $(v, w)$ could be in it). Thus it becomes Player 2's turn at $w$ and $w$ is covered by $M$. Player 2's strategy is to move along an edge in $M$, and an analogous contradiction exists here to show that Player 2 must have the winning strategy.

The problem of determining whether all maximum matchings of $G$ cover $v$ is in P by virtue of a fairly simple test. The size of a maximum matching can be found in $O(n^3)$ time (where $n$ is the number of vertices). If the matching number of $G$ does not equal the matching number of $G - v$, then $v$ must be covered by every maximum matching of $G$. □

We can apply a similar proof to $1W_I 1H_{NTW} US$, the undirected form of

Geography without the building stipulation, again meaning that the vertex moved from is not necessarily deleted, but rather any adjacent vertex to the node that is moved onto. Player 1 simply adjusts their winning strategy to be to move and delete a vertex along an edge in the maximum matching $M$. When Player 2 goes, they must move along a nonmatched edge (just as before), but they can potentially delete along a matched edge or along a nonmatched edge. If they delete along a matched edge, it is guaranteed that they did not delete the vertex they just moved from. In this case, Player 1 simply moves back to the previous vertex and removes the vertex they leave behind. This forces Player 2 into effectively the same position as before. If Player 2 on their turn had deleted along a nonmatched edge, then Player 1 is guaranteed to be able to move and delete along a matched edge, and does so. This allows Player 1 to win under the same conditions as before, meaning that $1W_I1H_{NTW}US$ without the building stipulation is also in P.

## 6.4 Proof that *Santorini* is in PSPACE

**Theorem 2.** *For fixed parameters, all discussed variants of Santorini are in PSPACE, where the size of the underlying graph is the input size to the resolving algorithm.*

*Proof.* The building nature of the game prevents any previous board state from occurring again. Given the height restriction on nodes and the fact that a node of maximum height cannot be moved or built onto, the game is of fixed length, with an obvious upper bound on game length being $h \times n$, where $h$ is the max height and $n$ is the number of nodes in the graph.

Let $S = \{\langle G, x_1, x_2, ..., x_i, y_1, y_2, ..., y_i, h, L\rangle | \ P_1$ has a winning strategy for the game *Santorini* played on the given instance of a graph $G$ with max height $h$ with node height values described by $L$ and $P_1$'s pieces starting at nodes $x_1$ through $x_i$ and $P_2$'s pieces starting at nodes $y_1$ through $y_i\}$. Let $B_n$ describe the board state given by $G, x_{1_n}, x_{2_n}, ..., x_{i_n}, y_{1_n}, y_{2_n,...,y_{i_n},h,L_n}$. To show that $S \in$ PSPACE, consider the following polynomial-space recursive algorithm $M$ which determines which player has a winning strategy. This algorithm works whether the graphs are directed or not, and whether the building stipulation is in place or not.

$M(B_{start})$ :

1. If tower wins are enabled, check to see if nodes $x_{1_{start}}$ through $x_{i_{start}}$ have height $h-2$ or $h-1$. If any do (or if free movement is enabled), check their respective adjacent empty nodes to see if any have height 3. If any do, return *accept*, as the active player has a winning move.

2. Construct a list of all possible board states reachable from $B_{start}$ by one ply (a move and build from the active player): $B_1, B_2, ..., B_k$. This list includes less than $i \times n^2$ board states, where $i$ is the number of workers the player has and $n$ is the number of nodes in the graph, since every piece has, at most, $n-1$ move options with an accompanying maximum $n-1$ build options. So $k \leq i \times n^2$. If the list is empty, return *reject* because the active player has no move options and thus loses.

3. For each $B_j$ in the list $B_1, ..., B_k$, call $M(\langle G, y_{1_j}, y_{2_j}, ..., y_{i_j}, x_{1_j}, x_{2_j}, ..., x_{i_j}, L_j\rangle)$.

4. If all of these calls return *accept*, then no matter what $P_1$ does, $P_2$ has a

winning strategy, so return *reject*. If any of the calls return *reject*, then $P_1$ has an option to deny $P_2$ any winning strategies, so return *accept*.

The algorithm $M$ decides $S$. The algorithm's input contains several terms to consider. Included is a graph G on $n$ nodes, with a memory complexity of maximum $O(n^2 \log n)$, needed to store the adjacency list of each of the labeled $n$ nodes along with their labels. The next $2i$ terms, $x_1, x_2, ..., x_i, y_1, y_2, ..., y_i$, are simply positions of the workers, each of which can be represented by a binary encoding of some number less than or equal to the number of nodes $n$, meaning that each requires space bounded by $O(\log n)$. The final term, $L$, is a representation of each node's height and can be bounded by the number of nodes multiplied by the space required to store their height, so $O(n \log h)$. Considering all the input terms, we have $O(n^2 \log n + i \log n + n \log h)$. Given that the number of workers per player $i$ and the maximum height $h$ are fixed per game, this simplifies to $O(n^2 \log n)$.

No operations within the algorithm besides its recursion use any significant amount of space. The recursion, given the game's maximum length of $h \times n$ moves, has depth at most $h \times n$. Since the space complexity of a recursive algorithm is its depth multiplied by its input space, we find that the space complexity of the entire algorithm is bounded by $O(n^3 \log n)$. As such, $S \in$ PSPACE.

This algorithm can be simplified to resolve impartial games as well with minimal changes. Simply omit the pieces represented by the $y$ variables and when recursively calling the algorithm $M$ in the third step, make no changes to the order of the pieces.

□

# 7

# AI Development Practices

## 7.1 Brief History of AI Development

In 1956, John McCarthy first used the term artificial intelligence in an academic conference that he convened on the subject [20]. Since then, the term has risen to popular usage and is often used to refer to machines that are capable of approximating humans' natural intelligence: that is, machines which are capable of learning and problem solving [20]. An interesting philosophical question is frequently posed about machines with artificial intelligence. It is often asked as to whether they are capable of truly understanding a subject. No one can refute a computer's ability to process logic, but many have been skeptical of a computer's ability to *think*. It is argued that since computers are always applying some form of rote fact lookup, they are incapable of *thinking*. The imprecise definition of the word and the different expectations on artificial intelligence become important in determining whether a machine is "intelligent" or not.

This question was originally tackled by Alan Turing in a 1950 paper titled *Computing Machinery and Intelligence*, just a few years before McCarthy's conference [20]. Within this paper is the definition of the famous Turing test, an initial metric used to determine if a machine is sentient or not. The test is a game of imitation, built on the premise that a machine capable of imitating sentient human behavior would therefore itself be sentient [20]. The test involves three participants: the machine, an interrogator, and a competing person. The three are separated and are only allowed to communicate with typed notes. The interrogator is allowed to pose questions to the human and to the machine and is supposed to deduce which is which from the responses he receives. There are problems with Turing's test; opponents have claimed that imitating a human is not a proof of intelligence, but rather just a difficult problem [20]. Others have cited that intelligence may be possible without being able to pass the Turing test. In either case, Alan Turing's hope that the test would be passed within 50 years did not come to pass [20]. The year 2000 came and went, and we have still not found success in constructing a machine capable of passing the test.

## 7.2 Look-Ahead Algorithms and Heuristics

In other realms of "intelligence", machines have had more success in the intervening decades. Claude Shannon initially wrote about chess AI, dividing them into two distinct categories. Type-A programs used brute force, examining thousands of moves and utilizing a min-max algorithm similar to the one explained in Chapter 3 [22]. Type-B programs used specialized

heuristics and were supposed to be the more intelligent, "strategic" AI. Type-B programs were initially favored in the 50s and 60s due to hardware limitations, but as machines grew faster and capable of storing more memory, the "dumber" Type-A machines took over [22]. The famous Type-A program Deep Blue coded by developers at IBM challenged and defeated chess world champion Gary Kasparov in 1997 [19]. Deep Blue evaluated 200 million positions a second and averaged an 8-12 turn search depth [19]. Brute force programs like Deep Blue use a modified version of the minimax algorithm that was discussed in conjunction with game trees. This modification, called alpha-beta pruning, increases the depth to which a given machine can search in a given amount of time [1]. At every level of the game tree, the regular minimax algorithm explores a number of nodes that it does not need to. Consider the game tree in 7.2. The minimax algorithm will explore the tree left

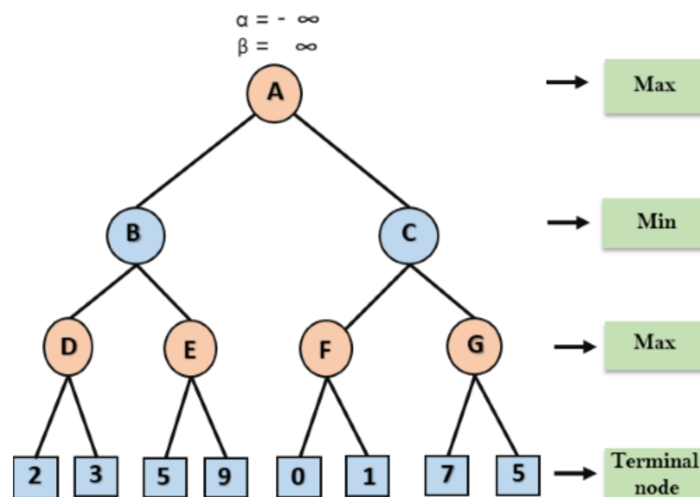


Figure 7.1: A tree on which the minimax algorithm is being executed [1]

to right. It searches down to node D, which returns the maximum of its two terminal child nodes. Normally, the minimax algorithm would continue and explore the rest of the tree. With pruning in place, however, that maximum, 3, is handed back to node B as a range limiter. Node B is trying to find the minimum of its children, yet its children are finding maximums from their respective children. Since node E is finding a maximum, if the value of the first child E searches is greater than 3, then E would return a value greater than 3. Node E's second child does not need to be searched in this case, as node B always returns the smaller value, which is thus guaranteed to be 3. These maximum and minimum range values are handed back up the tree between nodes (and reverse at each level of the tree) and are called alpha and beta values (hence the name alpha-beta pruning) [1]. On the right side of this tree,

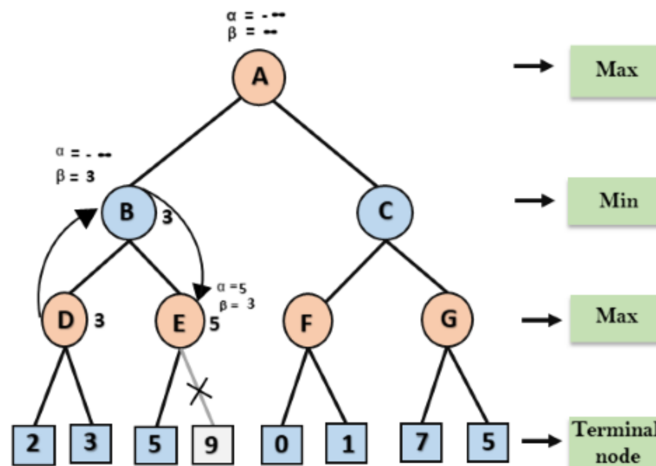


Figure 7.2: An example where the passed alpha-beta values allow the pruning of a node [1]

the pruning algorithm eliminates the search of a larger subtree. Node F returns the maximum value of its children, 1 and passes that back to node C. Node C, which is attempting to find minimum values, thus guarantees that the value it passes back to A will be less than or equal to 1. Since A is a max node and B has already been found to return 3, node G does not even need to be explored at all. A would return 3 regardless of G's contents. It is possible that the

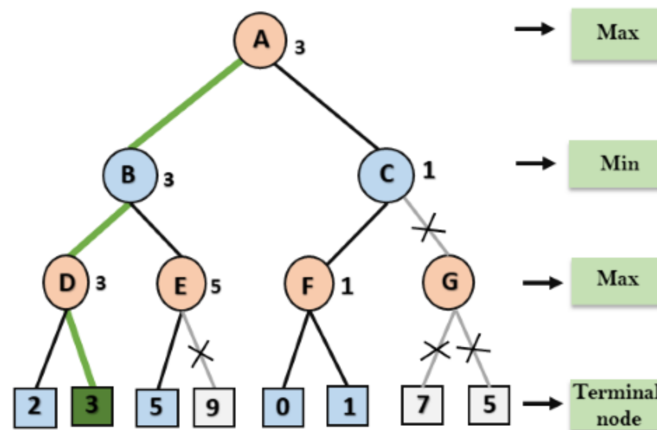


Figure 7.3: An example where the passed alpha-beta values allow the pruning of a subtree [1]

pruning algorithm does not prune any subtrees and that it runs in time comparable to the standard minimax algorithm [1]. This only occurs when the nodes are explored in the worst possible order, meaning that the best nodes are found on the right side of the tree. On the other hand, there is also potential that lots of pruning occurs and that the nodes are explored in the ideal order so as to prune as much of the tree as possible. In this ideal case, the alpha-beta

pruning algorithm can search twice as deep as the minimax algorithm in the same amount of time [1]. In practice, this means that any efficient means of ordering the nodes so the that best nodes are checked first is extremely valuable. In a combinatorial game setting, even with alpha-beta pruning, it is still intractable for a program to search to the end of the game tree for games of sufficient difficulty (such as chess or *Santorini*). As such, this is where heuristics come in. A heuristic is a function used during the search process that evaluates nodes [1]. In a game tree where searching to the end of the game is feasible, the only necessary heuristic is “does this move cause a win/loss?” The minimax function works backwards from this point to determine the sequence of moves that leads to the win (or loss). In games where the search space is too large, a different heuristic is necessary so that nodes at the bottom of the built game tree can be evaluated. For games like chess and *Santorini*, this heuristic comes as an analysis of the board state. A heuristic in chess might be the difference between the sums of the values of each players’ remaining pieces. Heuristics can be combined and values adjusted by experimentation. Programs using different heuristics can be compared by playing them against one another over and over and observing the resulting win-rates [1].

# 8

# AI in *Santorini*

This project includes a software implementation of the game of *Santorini*. The software is configurable, enabling a user to play *Santorini* against another human player, against an AI, or to have two AI play each other. The software currently allows only for the play of partizan *Santorini*, but it does allow for a number of the other variants discussed in this paper. The maximum and winning heights are configurable, both two worker and four worker *Santorini* are possible, and the underlying graph structure can be changed to a number of different configurations.

## 8.1 Limitations of Look-ahead

The AI included in this software search the game tree of *Santorini* using the minimax algorithm in conjunction with alpha-beta pruning to improve search depth. Even with pruning, the search depth is not particularly high. This is not surprising, as *Santorini* has an absurdly high number of choices per ply for

a game that appears so simple. Assuming it is not being restricted by tower heights, a worker in the middle of the board has 64 move options, and a player has two workers. That means that it is possible for there to be up to 128 choices per ply. Chess averages 30 choices per ply [19]. When including the board edges and the fact that the number of choices typically decreases as the game goes on, *Santorini*'s average choices per ply comes down to about 60-80, but this is still fairly problematic as the number of choices per ply increases the size of the game tree exponentially. In addition, *Santorini*'s second win condition (checking to see if the opponent is unable to move) requires essentially an extra ply of search depth to evaluate the previous ply. Thus, the AI included in this software are able to operate quickly at three-ply look-ahead, but four and beyond cause difficulty for the average computer.

One-ply look-ahead is fairly obvious to understand: If there is a winning move, the AI takes it. Two-ply look-ahead means that the AI prevents its opponent from winning on their next turn, if possible. Three-ply look-ahead means that the AI tries to set itself up to win on its own next turn, if possible. If we limit ourselves to three-ply look-ahead, the AI is not particularly impressive. A simple AI with three-ply look-ahead that otherwise makes random moves has trouble getting itself into a position where it is two turns away from victory and the look-ahead feature is actually useful. This is where heuristics come in; heuristics evaluate the game board before the end of the game, providing us with a way to choose "better" moves without necessarily having the assurance that the move will lead to a victory (as we do when only evaluating end-game positions). These heuristics are chosen in the hopes that they lead the AI to positions where they are two turns away from victory and

the basic form of the look-ahead algorithm can take over.

## 8.2 Valuable Heuristics in *Santorini*

An obvious necessity is a heuristic that prioritizes height, as this encourages the AI to move its pieces upwards. Moving up is crucial to victory, as the regular win condition requires stepping onto a space of height 3. In addition, pieces that are higher up have more movement options, as pieces are allowed to move downwards along height differences of greater than one but cannot move upwards along them. Having more movement options helps a player avoid becoming trapped and losing because they are unable to move. This software first implements a simple heuristic that evaluates a game board by summing the heights of the active Player's 2 pieces. This is then extended into a twofold heuristic that subtracts out the sum of the heights of the opponent's pieces, thus simultaneously encouraging moves that force one's opponent to move downwards (and thus away from victory). An example calculation of this heuristic is demonstrated in Figure 8.1.

A second possible heuristic is dependent on the "centricity" of the active player's pieces. Under most circumstances, it is strategically advantageous to hold positions in the center of the board, as one can reach and interact with more of the board in a single move. This software implements a heuristic that evaluates the game board by summing the centricity values of each player's pieces and doing the appropriate subtraction. The middle space is given a value of 2, the spaces in the inner $3 \times 3$ ring have value 1, and the border spaces have value 0. An example calculation of this heuristic is demonstrated

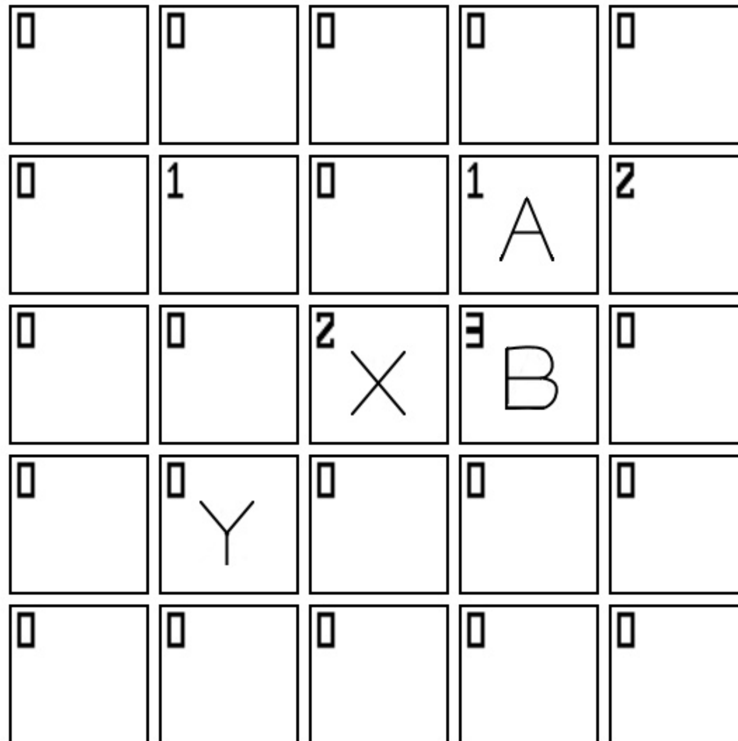


Figure 8.1: An example *Santorini* board where Player 1 (pieces represented by A and B) has a height sum of 4 while Player 2 (pieces represented by X and Y) has a height sum of 2. The evaluation of this board for Player 1 is thus $4 - 2 = 2$.

in Figure 8.2.

A third heuristic implemented in this software deals with distances between pieces. A piece that is sufficiently separated from its opponent's pieces can potentially set itself up to win and then subsequently win on its next turn without interference. If its opponents are close enough, after attempting to set itself up, its opponent will be able to move and then build on the tower of height 3 and thus prevent the piece from winning. As such, it is valuable for a player to have both of the opponent's pieces consistently within reach of his own pieces in order to have the potential to make blocking moves

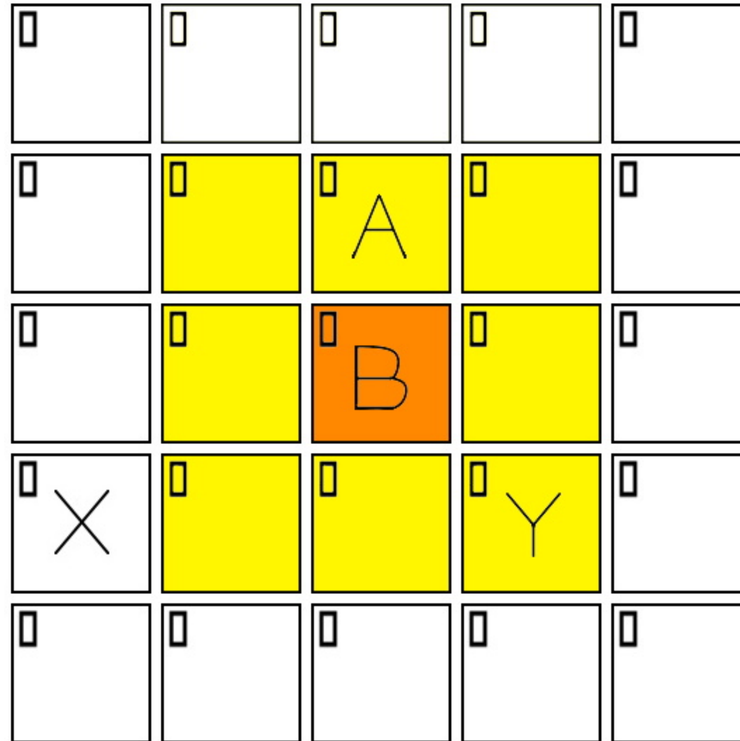


Figure 8.2: An example *Santorini* board where Player 1 (pieces represented by A and B) has a centricity sum of 3 while Player 2 (pieces represented by X and Y) has a centricity sum of 1. The evaluation of this board for Player 1 is thus $3 - 1 = 2$.

to prevent the opponent from winning. Consider the pieces A, B, X, and Y. A and B belong to Player 1; X and Y belong to Player 2. There are four relevant distances here: AX, BX, AY, and BY. Let the distance between two nodes be the shortest path between them. To obtain Player 1's distance from piece X, we take the minimum of AX and BX. Similarly, Player 1's distance from piece Y is the minimum of AY and BY. We are trying to minimize the sum of these distances, so lower numbers are actually beneficial. To correct the sign on these values, we simply subtract the sum of minima from 8 (the highest possible sum of the minimum distances) and use the resulting value for our

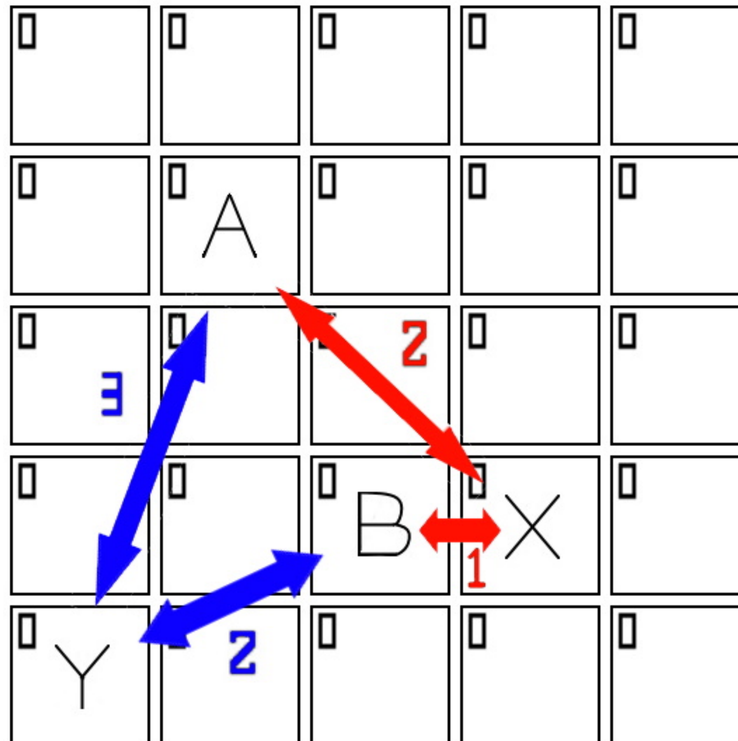


Figure 8.3: An example *Santorini* board showcasing the distances between pieces. We can evaluate Player 1's position by subtracting from 8 the sum of the minimum of the blue distances and the minimum of the red distances: $8 - (2 + 1) = 5$.

evaluation. As such, Player 1's position is evaluated as $8 - (min(AX, BX) + min(AY, BY))$. Player 2's position is analogously evaluated as $8 - (min(AX, AY) + min(BX, BY))$. An example calculation of this heuristic is demonstrated in Figure 8.3. Unlike the other two heuristics, Player 2's evaluation changes when Player 1 makes a move. It might seem initially that this heuristic is symmetric, but if one of Player 1's workers is close to both of Player 2's workers while the other is far away, then Player 1's sum of minima is smaller than Player 2's. This encourages the use of one piece to interfere with one's opponents and the other piece to set itself up for victory.

## 8.3 Improvements Using a Genetic Algorithm

The use of each of these three heuristics individually results in three AIs that are stronger than the basic three-move look-ahead AI that makes random moves. Each of these three heuristics uses some domain-specific knowledge derived from *Santorini* strategy. However, we still seek to improve the AI. A human player would likely make game decisions based on a number of strategic directives, considering multiple factors; an AI can emulate this by considering linear combinations of the heuristic values. To further improve this our AI, we would ask the AI to evaluate a game position using all three heuristic functions and combine them linearly with coefficients to obtain an overall value for the position. This evaluation function thus appears as: $e(x) = c_1 \times heightSum(x) + c_2 \times centricitySum(x) + c_3 \times minDistancSum(x)$, where $x$ is a position in *Santorini*. The values for these coefficients could be determined experimentally with the use of a genetic algorithm. In fact, the values for the heuristics in general (the point distributions for centricity, the values associated with each height level, and the values associated with varying distances) could all be determined genetically. A genetic algorithm is one inspired by Charles Darwin's theory of evolution. The fittest individuals of each generation are selected for breeding to create the strongest population. With AI, those with the strongest winrates against the others are selected for reproduction

# 9

# Conclusion

This project aimed to explore *Santorini* from three mathematical approaches. In this thesis, we have discussed graph theory and game theory concepts and investigated a number of variants of *Santorini* and brute-forced solutions to those variants played on small graphs. Further, we discussed Bachman-Landau notation, complexity classes, and attempted to prove that *Santorini* is PSPACE-complete.

In this endeavor we fell short. This thesis was successful in the first component of the proof: showing that finding a winning strategy in *Santorini* is in PSPACE. It was not successful in proving that *Santorini* is PSPACE-hard. Several proofs were made that took small steps from the proof that Geography is PSPACE-hard to show that certain restricted variants of *Santorini* are PSPACE-hard. Specifically, $1W_InH_{TW}D^*S$, $1W_I1H_{NTW}DS$, $2W_P1H_{NTW}D^*S$, and $2W_P1H_{NTW}DS$ are all PSPACE-hard.

Beyond theoretical aspects of *Santorini*, this thesis investigated AI development practices and implemented the minimax algorithm with

alpha-beta pruning in an AI that plays *Santorini* in a built software implementation of the game. This AI was capable of achieving three-ply look-ahead and was subsequently modified with three distinct heuristics born out of *Santorini* strategy.

The tri-pronged approach that this thesis took caused each area of analysis to be more cursory than is preferred. Each direction of analysis could be expanded significantly in the future. First, there is a considerable amount of data contained within the brute-forced small cases of *Santorini*. With more time to study the data and discern patterns, it is possible that larger generalizations about strategy among the different variants of *Santorini* on the many different graphs could be made. Second, one of this study's primary goals was to prove that finding a winning strategy in *Santorini* is PSPACE-complete. Having come up short of this accomplishment, a priority in future work would be to expand on the given proofs and combine all aspects of *Santorini* into a single reduction from QSAT to *Santorini* to show the missing aspect of the proof, that regular *Santorini* is PSPACE-hard. Third, in the realms of AI development, it would be of interest to explore other heuristics than those used in this thesis and to fully implement a genetic algorithm to determine what heuristics are most important. Future work could take a different approach to the AI development and use machine learning techniques and neural networks to train an AI to play *Santorini*. Additionally, future work could consider worker placement. This paper's analysis of *Santorini* is founded on finding a winning strategy for a given board state, but technically the first two turns in a standard game of *Santorini* are spent placing the workers. This adds an additional level of strategic depth that was not covered in this analysis.

# Bibliography

[1] Artificial intelligence: Alpha-beta pruning - javatpoint.

[2] Victor Adamchik. *Graph Theory*. Carnegie Mellon School of Computer Science, 2005.

[3] Eric Allender. *CRC Handbook on Algorithms and Theory of Computation*. University at Buffalo, 2009.

[4] Jose Luis Ambite. Complexity of domain-independent planning, Dec 2019.

[5] Akshay L. Aradhya. Minimax algorithm in game theory: Set 3 (tic-tac-toe ai - finding optimal move), Dec 2019.

[6] M.J. Atallah. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC-Press, 1998.

[7] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for Your Mathematical Plays*. Number v. 1. Taylor & Francis, 2001.

[8] John H. Conway. *On Numbers and Games*. Ak Peters Series. Taylor & Francis, 2000.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[10] Robert Crowter Jones. Santorini-strategy tips. *Elusive Meeple*.

[11] Aviezri S. Fraenkel and Elisheva Goldschmidt. PSPACE-hardness of some combinatorial games. *Journal of Combinatorial Theory, Series A*, 46(1):21–38, September 1987.

[12] Oded Goldreich. Invitation to complexity theory. `http://www.wisdom.weizmann.ac.il/~oded/PDF/inv-cc.pdf`, 2012. Accessed Sept 13, 2019.

[13] Gordon Hamilton. *Santorini Rulebook*, 2016.

[14] Gordon Hamilton. Santorini game rules. `https://www.ultraboardgames.com/santorini/game-rules.php`, 2017. Accessed Sept 13, 2019.

[15] Marjin J.H. Heule. Reasoning with quantified boolean formulas, 2015.

[16] A.J. Hildebrand. *Asymptotic Analysis*. University of Illinois, 2009.

[17] Robert Mullins. *Raspberry Pi Turing Machines*. University of Cambridge Department of Computer Science and Technology, 2012.

[18] Jesus Najera. Game theory - history and overview. https://towardsdatascience.com/game-theory-history-overview-5475e527cb82, 2019. Accessed Sept 13, 2019.

[19] Chris Piech. Deep blue. https://stanford.edu/~cpiech/cs221/apps/deepBlue.html, 2013. Accessed Sept 13, 2019.

[20] John Scott, Eren Jaeger, Anthony E., Gabby, Mariia, Joana, Meng Wong, Agatha, Romexsoft, Sonia Chang, and et al. The history of artificial intelligence, Apr 2019.

[21] Chris Wray. Santorini review. *Opinionated Gamers*.

[22] Brandon Yanofsky. Building a simple chess ai. https://byanofsky.com/2017/07/06/building-a-simple-chess-ai/, 2017. Accessed Sept 13, 2019.