Gamescrafters: Santorini Notes

Matthew F. Yu

I worked with Chris Xiong on Santorini the semester.

Notes on Positions

It is possible to traverse and cover the whole board up to 2 full layers (essentially playing Santorini with a person winning by stepping on a level 1 block).

End States

The end states of the game are either of two results. The first is if either player's worker lands on a level three block or a player is unable to perform an action (an action consists of moving one of your workers and build). There are no ties in the game. This can be proved by contradiction:

Assume that there is a tie condition. Then this means both players cannot move. However, since there are turns in this game, this implies that one player is unable to move first. Although both cannot move, the losing player is the first player that couldn't move.

This leads to the next section.

Nature of the Game

This game is considered a nonloopy game. This is because of the design of the game where you have to build when you make a move. Because of the rule structure and game set up, the board is bounded by $n \times n \times 4$ dimensions. Then, the game has to terminate at some point. By above, since the game cannot have any ties, then there is a definitive, binary win/loss for the $left$ and $right$ players.

3 by 3 Board

Knowing that the 5 by 5 board is too large to solve, I switched my attention to what Chris Xiong references in his writeup, the 3 by 3 board.

Tier N for 3 by 3 Board

After running code in the file CopySantorini2.py

Tier 1 : 26

Tier 2 : 476

Tier 3 : 8396

Tier 4 : 144889

Tier 5 : 2251545

Tier 6 : 32603441

Solving the 3×3 Board

I attempted to solve the board for 3×3 dimensions, and in a sense, I semi-solved it. My solver can solve games that are Tier 13-28 within a second. For some in Tier 8-12, it would take around 10 seconds to 2 minutes to solve. Most board games states within tier 1-12 are not possible to solve within a reasonable time. Thus, I built a bot in the file TicTacToeBot.py that plays against itself (or "another" bot) or a human. After testing it, we see that its not comprehensive on

Future Work

There are two ways to solve this in the future.

Method 1

From the above section of counting Tier N for a 3-by-3 board, we have that counting the tiers of each level can be mathematically approximated. This is because of mathematical sequence $\left(\frac{|\text{Tier}(n+1)|}{|\text{Tier}(n)|}\right)_{n=0}^{28}$ is a unimodal sequence that is bounded (see Chris' stats on the tier for 5×5 game). In other words, $\exists a, b \in \mathbb{R}^+$ such that

$$\forall i, a < \frac{|\text{Tier}(i+1)|}{|\text{Tier}(i)|} < b$$

and that for some i such that $0 \leq i \leq 28$, we have $\forall k$ s.t. $0 \leq k < i$ or $i < k \leq 28$

$$\frac{|\text{Tier}(i+1)|}{|\text{Tier}(i)|} > \frac{|\text{Tier}(k+1)|}{|\text{Tier}(k)|}$$

I still have yet to develop a hashing function. Researching, I came across a method known as the Zorbrist Hash. The point is to hash each board state to a number $2^{64} - 1$, and that already computed board states don't need to be computed again. Instead, they can be called from the hash. I hope to try an implement this in the future.

Method 2

Although still not exactly the best way, we can continue with our primitive solver that we made in the beginning in the semester. I tried to improve the runtime by using the concept of $\alpha\beta$ pruning. This means as it is running through the moves, if it yields a winning move at some point of the list, it doesn't need to compute the remaining moves in the list. Although it helped a little, the runtime is still too long to compute.

However, this will lose information on the analysis, i.e. we wouldn't be able to find all possible winning, tieing, and losing positions.