Universiteit Leiden
The Netherlands

Opleiding Informatica

Solving and Generating

Ice Block Puzzles

Sem Kluiver

Supervisors:
Walter Kosters
Jeannette de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

22/05/2020

Abstract

This paper analyses Ice Block Puzzles from the Legend of Zelda series, which are a variation on PushPush. The puzzles consist of ice blocks, ice floors, floors, walls and targets. The movable ice blocks can move in four directions and only stop moving when they hit an obstacle. The objective is to get an ice block on every target. We discuss different algorithms to solve these puzzles and compare their results. We also discuss a way to generate these puzzles, in addition to exploring different properties of the puzzles, such as the solvability, reversibility, search space and difficulty.

Contents

1 Introduction

Nintendo's The Legend of Zelda series are well-known for their dungeons. Each dungeon has its own theme, and according to the different themes, different puzzles. Most games have reoccurring themes for dungeons - particularly dungeons based on elements, like fire, water, but also on things like forest and ice.

Especially the ice-themed dungeons feature a certain class of puzzles that is one of the series' staples: Ice Block Puzzles. The goal of these puzzles is to get ice blocks in a particular spot, which will open a door and allow Link (the player) to continue on his quest to slay Ganon and save the world from evil.

Figure 1 shows an example of an Ice Block Puzzle. The light blue squares represent ice blocks. The yellow marking in the middle represents a target, where one of the ice blocks must be put on. Even though it might seem hard, this puzzle is actually solvable! The method to solving this particular puzzle will be discussed in Section 5.

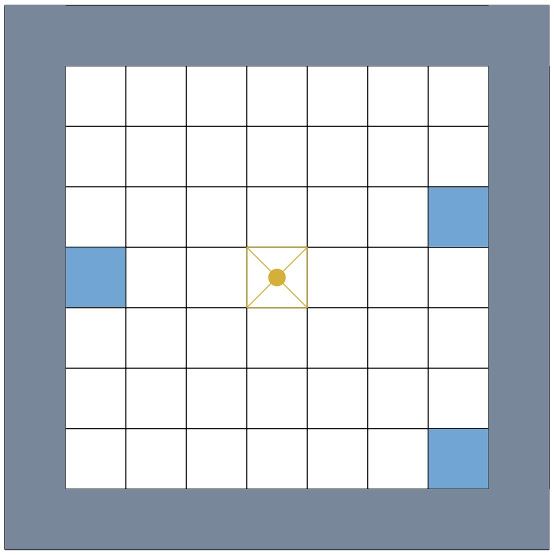


Figure 1: Example

In Section 2, we will explain the rules of the Ice Block Puzzles. Section 3 will mention previous and related work. Section 4 proposes the different algorithms we have used to solve the Ice Block Puzzles and an algorithm to generate new puzzles. In Section 5, we will explain the solvability and reversibility of Ice Block Puzzles. In Section 7, we try to give Ice Block Puzzles a difficulty measure. Section 8 will discuss the results of the different algorithms. Appendix A contains all Ice Block Puzzles that appeared in the Legend of Zelda games (so far).

This paper is a bachelor thesis for the Computer Science studies at Leiden University, supervised by Walter Kosters and Jeannette de Graaf.

2 Description

Ice Block Puzzles are played on a (usually) rectangular "board". The board consists of five different types of fields:

1. Ice blocks: These are movable blocks which slide over the ground.
2. Ice floors: The icy floors over which ice blocks can slide. The player is allowed to walk here.
3. Normal floors: Normal floor where the player can walk, but ice blocks are not allowed to be on these fields.
4. Walls: These are unmovable blocks or obstacles: ice blocks are not allowed to be on these fields and the player cannot push ice blocks from here.
5. Targets: these are the fields where an ice block(s) must be put on. The player is allowed to be on a target and can push ice blocks while standing on the target, unless there is an ice block on the target itself

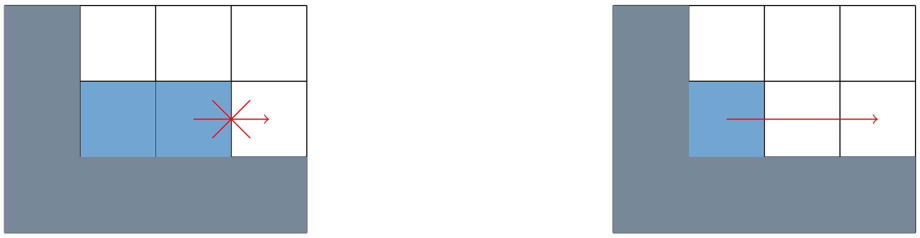
The number of targets and ice blocks may vary. There also do not have to be as many targets as there are ice blocks, but the number of ice blocks has to be equal to or larger than the number of targets for the puzzle to be solvable.

For the player to push an ice block, he must stand directly next to the ice block on the opposite side of the direction he wishes the ice block to be pushed in. Ice blocks can be pushed in any of four directions: north, east, south and west. The ice block will move as long as there is nothing which prevents it from moving in that direction; it will continue to slide until it can slide no more, either due to hitting a border of the board, a wall, a normal floor or another ice block. Ice blocks will slide over targets and will continue to do so, unless an obstacle next to the target prevents this. Ice blocks cannot be lifted and placed wherever one likes, and, since they continue sliding until they hit an obstacle, cannot be stopped by the player.

The goal of Ice Block Puzzles is to have an ice block placed on every single target. There are some restrictions on when a player is allowed to push an ice block. It is impossible for a player to push an ice block while standing on a wall or another ice block. It is also impossible to push multiple ice blocks at the same time, even if they are adjacent to each other. We have no restrictions on player movement. The player may walk anywhere he pleases. If we have puzzle where a part of the puzzle is surrounded by walls, the player is able to climb over the walls and get into the inner section of the puzzle and move ice blocks. The same holds for ice blocks. The player can therefore not be imprisoned by a sequence of moves.

In this paper, we will depict ice blocks as blue tiles, normal floors as gray tiles, ice floors as white tiles, walls as dark gray tiles and targets as white tiles with a yellow cross and dot, surrounded by lines. For simplicity, the player has been omitted, as their movement is not restricted: only the pushing of ice blocks may be restricted. We assume all puzzles are surrounded by normal floors. Therefore, we usually omit them in small examples.

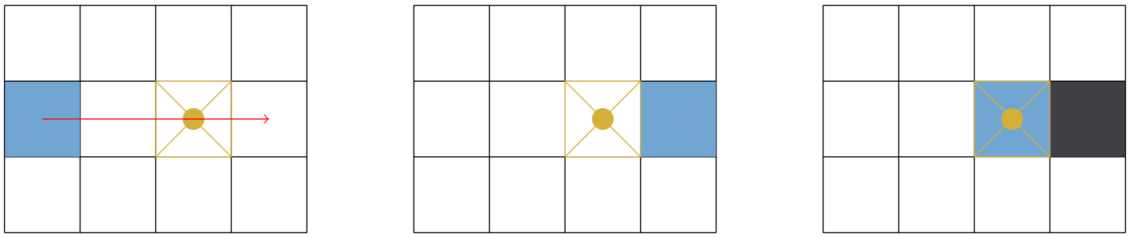
Since the player is not able to push every ice block at all times, some moves are impossible to do. For example, when two ice blocks are next to each other, the move shown in Figure 2a is impossible. This move would also be impossible if there was a wall to the left of the ice block we try to move. The move shown in Figure 2b, however, is possible, since nothing prevents the player from standing on the opposing side of the direction we try to slide the ice block.



(a) Impossible move due to adjacent ice block (b) Possible move

Figure 2: Impossible moves

The move seen in Figure 3a will not result in the ice block being on the target, but rather, it will slide past the target and result in the position seen in Figure 3b. For the ice block to reach the target with this move, there would have to be something to stop the ice block, such as the wall in Figure 3c. This could, however, also have been a normal floor or another ice block.



(a) Move (b) Result (c) Desired result

Figure 3: An ice block will slide past targets unless stopped by an obstacle

Of course, the key to harder puzzles is to place ice blocks in such a way that other ice blocks can reach a target. For example, consider the simple puzzle and its solution as seen in Figure 4. This puzzle would be impossible to solve if the puzzle only had the bottom left ice block. In puzzles like these, more than one ice block is required to solve the puzzle.

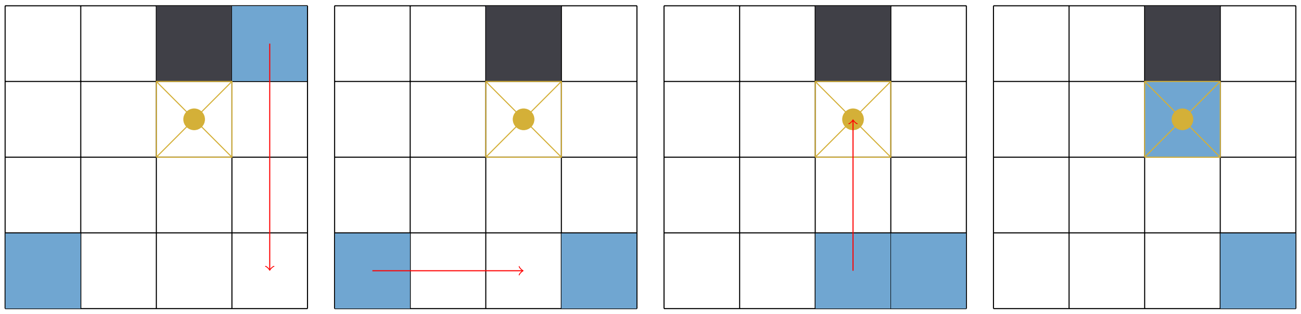


Figure 4: A simple example of a puzzle requiring two ice blocks to solve

3 Related Work

Not much about Ice Block Puzzles with their exact rule set has been published. There have, however, been multiple papers about similar puzzles.

A paper about Ice Sliding Games was accepted in 2015 [DDF+18]. In this paper, the authors describe a game where the player-controlled "robot" has to reach a target, rather than ice blocks pushed by the player. Just like with Ice Block Puzzles, this puzzle is played on a board. The authors researched the minimum number of blocks necessary such that the "robot" can visit all fields of the board. The robot did not have to actually stop on all fields, it could just slide past them. There is only one robot, as opposed to Ice Block Puzzles potentially having multiple ice blocks. The authors do suggest puzzles with multiple robots to "help" the main robot in its task to visit all fields.

Ice Block Puzzles are somewhat similar to PushPush, a 1994 Macintosh Game. In that game, there are two different types of movable blocks, Idgits and normal blocks. Idgits are creatures that need to be pushed onto the targets. The normal blocks are only there to help creating a path for the Idgits. In addition, Idgits disappear when they hit a target: they will not slide through or stay on the target, meaning that multiple Idgits can go to the same target. Not every target has to have an Idgit on it to win the game. In Ice Block Puzzles, the ice blocks slide over targets unless another ice block, a wall or normal floor prevents this. Once the ice blocks are on the target, they will stay there, unless they are pushed off by the player. This means only one ice block can be on a target at a time, and thus all ice blocks need to be on different targets. The NP-hardness of a 3D version of PushPush has been proven by Joseph O'Rourke and The Problem Solving Group [OCC+99]. Martin Demaine, Joseph O'Rourke and Erik Demaine have proven PushPush is also NP-hard in 2D [DDO00b]. They have also proven that Push-1, a similar puzzle, in which pushed blocks only move one square rather than until they can go no further, is NP-hard [DDO00a]. It has not been decided yet if PushPush is in NP, or if every puzzle has a polynomial-length solution path, or if it is PSPACE-hard [Dem].

Another similar puzzle is Sokoban. The main difference between Ice Block Puzzles and Sokoban is the fact that blocks that are pushed in Sokoban only move by one square at a time. Just like Ice Block Puzzles, there are unmovable walls, which are called barriers in Sokoban. Sokoban has unrecoverable configurations. Ice Block Puzzles also have these, which will be discussed in Section 5. Sokoban has been proven to be PSPACE-complete when the puzzle contains barriers [Cul98] and later, Hearn and Demaine proved that Sokoban is PSPACE-complete even without barriers [HD09].

Jarno Huibers investigated a somewhat similar type of puzzle with sliding particles. Contrary to the Ice Block Puzzles, however, the sliding particles puzzles have a so-called global operator, which makes all particles move in a certain direction in one move: it is impossible to move only one particle in one direction. The paper discusses Strongly Connected Component graphs and how states of the puzzles can be divided in Strongly Connected Components. Huibers states that no new Strongly Connected Component graphs can be found after a certain grid size, defined by the number of particles and obstacles on the grid [Hui17]. More has been researched on similar puzzles in [SMD+15] and [BDF+19].

4 Algorithms

In this section, we will explain the algorithms we have used to solve Ice Block Puzzles. The algorithms try to find a solution to the Ice Block Puzzles. We also discuss the generation algorithm.

4.1 Random Algorithm

The random algorithm first chooses a random ice block and then a random direction in which to move the ice block. All moves have the same probability of happening — even moves which are not possible or do not result in a different game state. The program asks for a a maximum number of moves the random algorithm is allowed to make. This is necessary, since the random algorithm might get stuck while searching for a solution. It might make a sequence of moves resulting in a game state where the solution cannot be reached anymore. Additionally, some solutions are too hard to find for the random algorithm, since these solutions require a certain pattern, such as the one discussed in Section 5, which is very unlikely to be solved by a random algorithm. This algorithm stops as soon as it finds a solution.

4.2 Brute Force Breadth-First Search

Brute Force Breadth-First Search (from now on referred to as BFS) tries every single possible move in every single game state. For every encountered game state, rather than playing it until the end, BFS tries every move on that state, before moving on to the next state, in which every move is tried as well. This is the opposite of Depth-First Search (DFS), where the algorithm "commits" to a move until the solution is found, then trying another move using backtracking. DFS is not suited for finding solutions for the Ice Block Puzzles because it would take a very long time to find the fastest solution. DFS has to continue until all possible "paths" have been found. On the other hand, BFS has found, in that case, one of the fastest possible solutions as soon as it finds one. BFS only considers new game states: after a move has been done, the algorithm checks if this state has not occurred before. This prevents BFS from getting stuck in an infinite loop by moving ice blocks back and forth. BFS stops looking for a solution as soon as it has found one.

4.3 Pruned Brute Force Breadth-First Search

Pruned Brute Force Breadth-First Search (from now on referred to as P-BFS works very similarly to BFS, trying every single move on every single game state. One key difference is, however, that P-BFS receives a penalty score for blockchanges, that is, a penalty for changing which ice block the algorithm uses in its current move as opposed to the ice block used in the last move. See: Section 7, where we explain blockchanges and what we consider as a puzzle with high difficulty. So, contrary to BFS, P-BFS considers blockchanges as difficult. Every possible state receives a difficulty measure based on the number of moves that has been done and the number of blockchanges. P-BFS tried to find the easiest possible solution, with the lowest difficulty. Therefore, P-BFS will continue looking for solutions, even if it has found one.

Since P-BFS assigns difficulty measures to states, this algorithm sometimes needs to reconsider states it encountered earlier, for example if this state can been reached in an easier way. States also need to be reconsidered if they were reached with a different last-used ice block. However, if P-BFS encounters a state it had found earlier and with another last-used ice block, but the difference in difficulty is greater than the blockchange factor, the state will not be reconsidered again. Still, P-BFS has to calculate a lot more than BFS, and is therefore much slower.

Depending on the blockchange factor, P-BFS may find that making more moves and less blockchanges is easier than making less moves and more blockchanges. P-BFS will therefore find different solutions depending on the blockchange factor.

4.4 Monte Carlo

Basic Monte Carlo, referred to as BasicMC, tries every possible move at a given game state and proceeds to play a certain number of random games with the new game state. By assigning scores to the states the board is in after the series of random moves, all moves that are possible from the current game state are ranked by a score. The score, in our case, is based on the number of times the puzzle has been solved after the series of random moves. First, we tried giving a positive score for every target where an ice block has been pushed on, but as can be read in Section 8, this resulted in BasicMC being "satisfied" after having only one ice block on one target, which prevented the algorithm from solving puzzles with multiple targets. We changed this so that BasicMC only gets a positive score when completely solving the puzzle. Based on this score, BasicMC tries to determine what might be a good move and chooses the best one. If all random runs fail to find a solution, no move will get a score. BasicMC will then proceed to make a random move. The program accepts user input for the number of random games to play and the number of moves the random algorithm must make (unless a solution has been reached in less moves).

It turned out that the original BasicMC algorithm would at some point move ice blocks back and forth between two states, since both of these states were so close to the solution. The random sequence of moves would in both of these states lead to a solution, therefore giving positive scores to the moves that resulted in the "dead-lock". We therefore decided to tweak BasicMC, as can be read in Section 8. The algorithm can do less and less moves in its random function for every move it makes. By reducing the number of moves BasicMC was allowed to make, the algorithm breaks free out of the "dead-lock" and solve more puzzles correctly.

4.5 Generation

We decided that the generation of solvable puzzles would be the easiest by making the puzzles in a backwards, retrograde manner. The algorithm accepts user input for the board size, the number of targets, the maximum number of ice blocks and the maximum number of moves. It then proceeds to generate a board using the following steps:

1. Place all targets randomly on the board, with an ice block on it.

2. Choose one of the ice blocks.

3. Choose a random direction for this ice block, which can be restricted depending on its previous direction.

4. Place something next to the ice block in the direction opposite the direction we just chose, if there is nothing there already. This may be another ice block or a normal floor, which is chosen randomly — if there are ice blocks left to use, otherwise the algorithm will always place normal floors. The edge of the board also works. We do not use walls, since these may restrict some moves, resulting in an unsolvable puzzle.

5. Choose a random distance between one and a certain maximum.

6. Place the ice block on the new location, determined by the distance from the previous step.

7. Repeat step 2 through 6 until a given number of moves has been done.

Because of the nature of Ice Block Puzzles, the directions in which an ice block can move are restricted. If the ice block moved horizontally in the last step, it can only go vertically in the next move. This also holds the other way around. Ice blocks that are used in step 4 to stop another ice block also have this same restriction: if a new ice block has been placed next to a current ice block and the direction chosen in step 3 was horizontal, the new ice block may only move vertically in its first step. Otherwise, the random direction of the new ice block may be chosen to be horizontal as well, which will prevent the original ice block from reaching the field it was previously on. The algorithm notices if the ice block cannot move in the chosen direction at step 3 and will try another direction along the same axis. If this fails, the algorithm will choose another ice block, and if all else fails, the algorithm will stop prematurely.

While generating, the algorithm keeps track of over which fields the ice blocks need to slide, called "paths". This prevents our generation algorithm from putting normal floors over paths an ice block has moved over in a previous generation step. The coordinates ice blocks have visited while generating are also explicitly kept track of.

Generating puzzles in this manner will theoretically always result in a solvable puzzle. However, puzzles generated by our algorithm usually have a solution that required less moves than the one that was generated. This is because the generation algorithm chooses random directions. The algorithm does not check if there are easier ways to reach targets. We have tried to reduce this last issue by having our Breadth-First Search algorithm solve the puzzle after the generation. When BFS is done, the program analyses the solution and checks if the ice blocks have moved over the aforementioned paths. If not, the generation algorithm will randomly (with some constraints to keep the puzzle solvable) place a normal floor on ice floors which have not been labeled as "paths", but where an ice block has moved over. This prevents unintended moves from being possible. This results in the fastest solutions (that is, requiring the least number of moves) of generated puzzles being more likely to actually be how they were intended to be by the generation algorithm.

A short and simple example of our final generation algorithm can be seen in Figure 5. If there would be multiple targets at play, the step seen in Figure 5a would have all targets with an ice block placed on it. If there were more ice blocks on the board, in the step shown in Figure 5b a different ice block could be chosen per movement. The normal floor which has been placed in Figure 5b could also have been another, new ice block. The normal floor placed in Figure 5e could also have been placed one field lower.

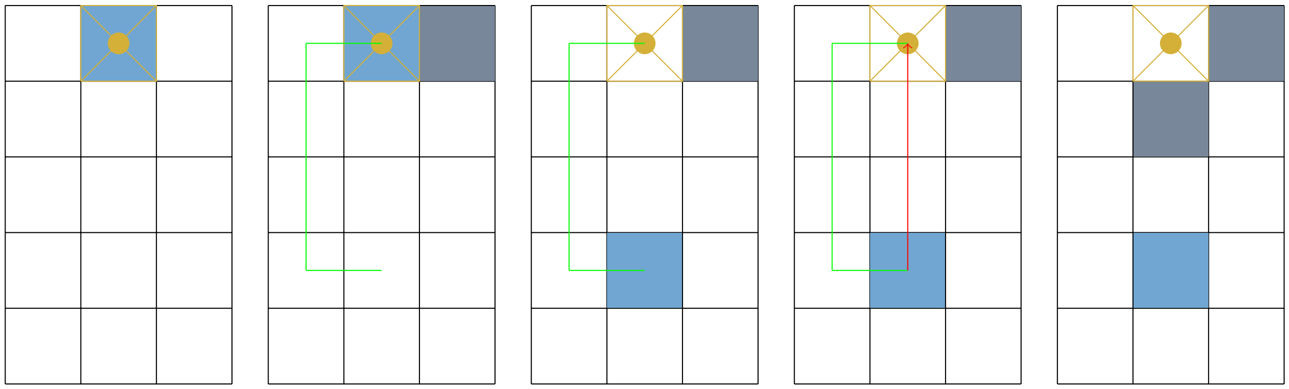

(a) Placing targets (b) Choosing random directions and distances, placing blocks (in this case a normal floor) where necessary (c) Generated puzzle with path (d) Solution found by BFS (e) Resulting puzzle with randomly placed normal floor

Figure 5: A simple generation example

Of course, if the user inputs a ridiculous number of moves on a board of limited size (e.g.: 50 moves on a 5 by 5 board), this random normal floor placing safety measure will not work. By the time all of the moves have been done while generating, the ice blocks are very likely to have visited every single space of the board, filling it with paths. This prevents the generation algorithm from knowing over which fields the ice blocks were supposed to slide and over which one they should not slide, because the ice blocks have moved over every single field. The solution BFS finds is therefore accepted as correct, because due to all fields being paths, the generation algorithm cannot know if BFS took a shortcut: besides, the generated solution would be impossible to preserve, because placing a normal floor on a path would make the generated solution impossible.

Figure 6 shows some of the more "impressive" puzzles this algorithm has generated.

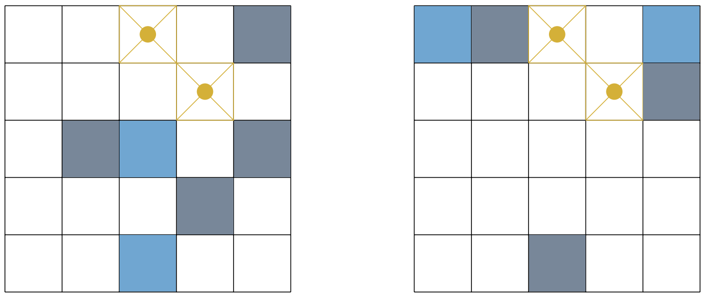


Figure 6: Two simple 5 × 5 puzzles

5 Solvability and Reversibility

In this section, we discuss further properties of the Ice Block Puzzles: the solvability and the reversibility.

5.1 Solvability

One interesting aspect of Ice Block Puzzles is that a certain type of puzzle is always solvable. As mentioned in Appendix A, the first and third puzzle from the Twilight Princess Ice Block Cavern are interesting, and the reason is that these are examples of a kind of puzzle that is always solvable. Indeed, every puzzle with a rectangular board that contains three ice blocks and one target and no obstacles is solvable, no matter where the target and the ice blocks are in the initial state. Furthermore, every single field on the board is reachable when the player has three ice blocks at their disposal, meaning that regardless of where this single target is placed, the puzzle can be solved. See the example in Figure 7, where we have chosen an initial state similar to the first and third puzzle of the Ice Block Cavern. If this is not the case, all ice blocks need to be pushed into corners.

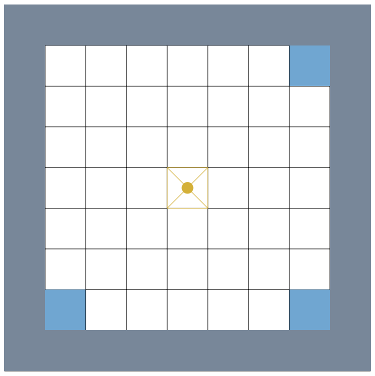


Figure 7: Initial state of puzzle

The first step to solve these kinds of puzzles is making sure that one of the ice blocks can reach the same row or column as the target in the following step. This can be achieved if we do the moves shown in Figure 8. If the board is even larger than the one shown here, the upper of the three ice blocks shown in the rightmost image of Figure 8 may be moved west, south, east and north. This can be repeated until the row (or column) the target is on can be reached in the following step.

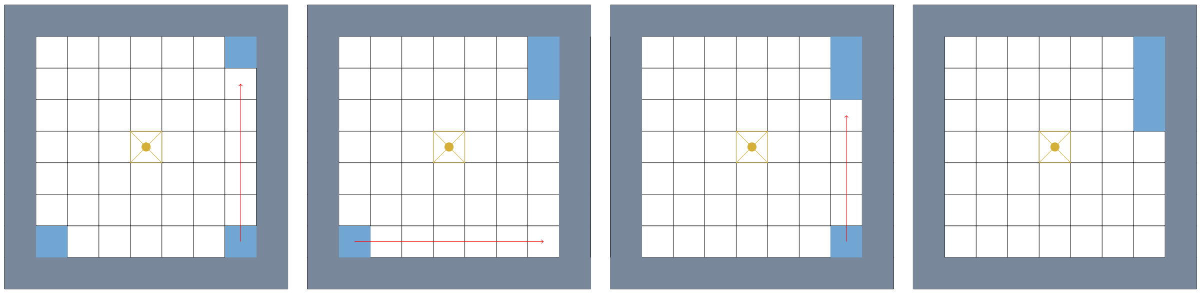


Figure 8: The first loop

Next, an ice block should be placed on the same row or column as the target. The ice block the farthest from the target row or column must also be placed on the opposite side of the board. In our example, it is now possible for us to get an ice block in the same row as the target by doing the moves seen in Figure 9. We also bring the current upper ice block in the lower half of the field.

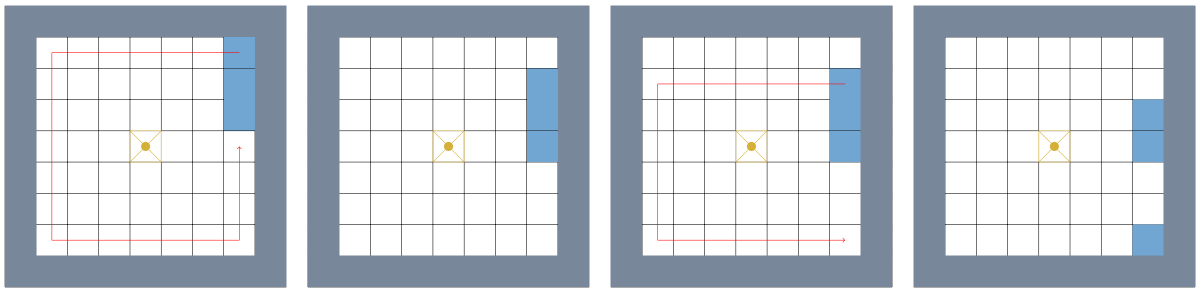


Figure 9: The second loop

Now we can repeat a loop of moves to keep stacking the ice blocks onto each other. We essentially create stopping block which moves further into the field each time, which allows one of the ice blocks to finally reach the middle target, as seen in Figure 10.

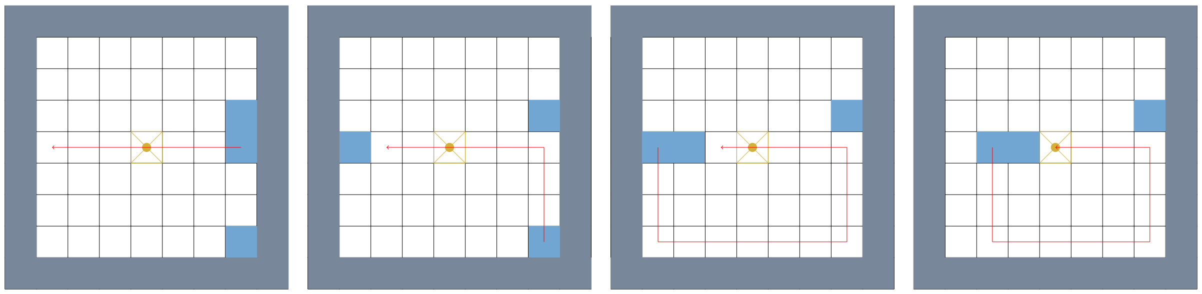


Figure 10: The loops

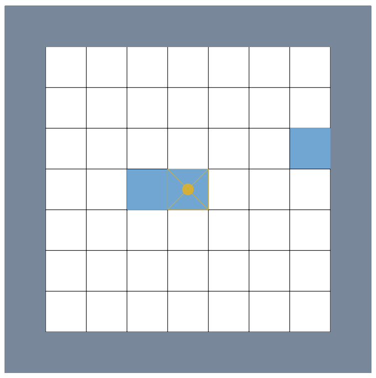


Figure 11: Solution

Since the solution to this puzzle essentially consists of two looping patterns of moves (shown in Figure 8 and Figure 10), which results in getting an ice block one square further into the field with every loop, this puzzle can also be solved with boards larger than the above example. We refer to this looping of moves as "trains". Now, imagine there is another target and another ice block, then we can then do the same kind of moves for this second target after having placed an ice block on the first target. In general, we suspect that every puzzle with a rectangular board without walls and normal floors and with m targets can generally be solved with $m + 2$ ice blocks, since we can repeat the example above after having put one ice block on one of the targets.

When there are way more ice blocks on the board, the movement of the trains will get constrained. We cannot say with how many ice blocks on the field this strategy still works, see Section 9.

5.2 Reversibility

One interesting aspect of the Ice Block puzzles is that moves are almost always reversible. Most moves can usually be undone, either in just a single move or by doing a sequence of moves. Usually, undoing a move is as simple as sliding an ice block back in the opposite direction, as seen in Figure 12. We call a state and move reversible if the state can be reached again with a sequence of moves.

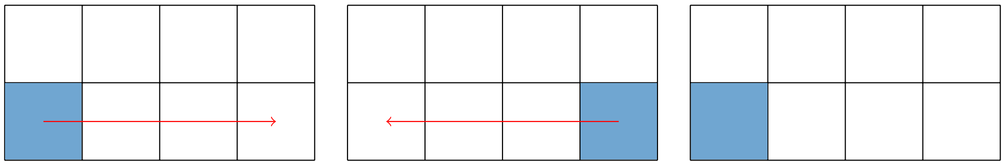


Figure 12: Reversible in a single move

This will not work when the ice block was initially not against a square that stops it when moving it back, since this would result in the ice block moving past its previous position, as seen in Figure 13. We call states and moves as seen in Figure 13 irreversible, since no sequence of moves can result in the initial state.

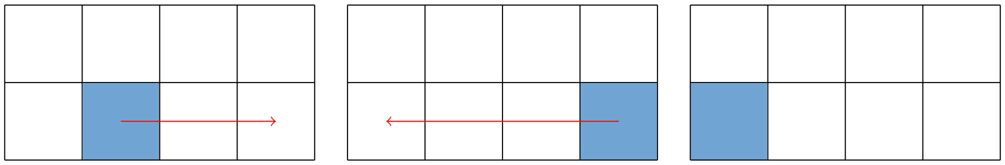


Figure 13: Move is not reversible for there is nothing to stop the ice block

The initial position might, however, still be reached by doing a sequence of moves (this is not possible in the example above). A state and move are not directly reversible if more than one move is required to bring the puzzle back in the initial state. For example, when the ice block is moved against another ice block or a wall, this prevents the player from moving the ice block back all together, as seen in Figure 14. This example is not reversible in one move. However, if we were to push the ice block north, west and south, we would still regain the initial state.

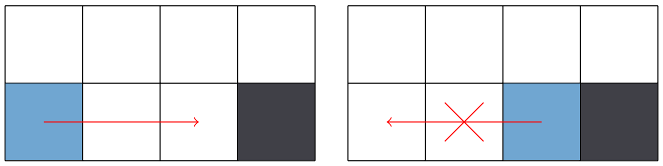


Figure 14: Move is reversible, but not in one move

There are configurations of puzzles that prevent the game board from ever reaching a certain set of states again after a certain move. Rather than sliding past the field the ice block was initially placed on, as in Figure 13, other blocks or even the structure of the board may hamper reaching the original state. The smallest configuration is the "puzzle" seen in Figure 15, which has a wall that prevents the ice block from sliding back.

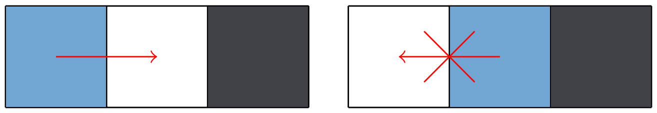


Figure 15: Move is not reversible at all due to wall

Figure 16 shows a sequence of moves for a puzzle with ice floors in a T-shape. This is an example where the shape of the board may obstruct the reversal of moves.

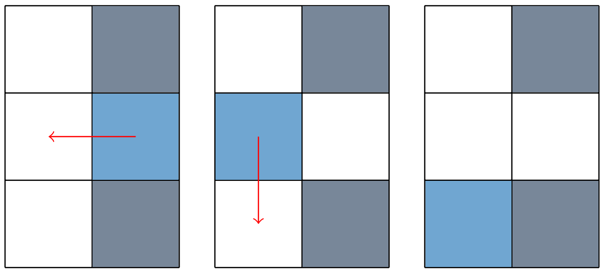


Figure 16: An irreversible sequence of moves on T-shaped ice floors

Moving the ice block up in the rightmost state would result in sliding it past the intersection. Reaching the middle right square (and the middle left square as well) is impossible after this sequence of moves. Analogously, moving the ice block up in the second move gives the same result.

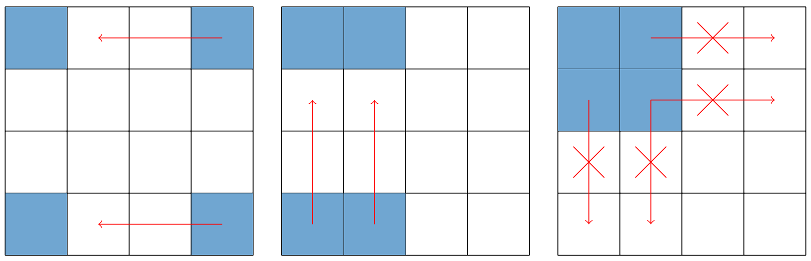


Figure 17: Building a stack of ice blocks in a corner leads to no moves being possible at all

Another example of an irreversible state and sequence of moves is one where ice blocks would be stacked in a corner. If there are four ice blocks on the board and these are moved in a corner to form a two by two block, no moves are possible anymore at all. This can be seen in Figure 17.

States that are reachable from each other can be placed in a subset of states. Since the states in this subset can all be reached from another state within that same subset, it does not matter in which state of this subset the puzzle currently is. We call puzzles where every state can be reached from every other state entirely reversible. These puzzles have a single set of states, since every state can be reached from any other state. This, however, does not mean that all states are as close to the solution of the puzzle as the other states in the same subset. We call puzzles where some subset of states cannot be reached from another subset of states not reversible or irreversible.

Some of the puzzles that are not reversible cannot be solved anymore after a certain sequence of moves. In this case, the puzzle is stuck in a subset of states that cannot lead to a solution.

6 Search Space

This section discusses the search space of the Ice Block Puzzles. The size of the search space is defined by the number of unique states a puzzle can be in. Here, we consider all ice blocks to be equal, so if two ice blocks are swapped, the resulting state is the same as the original one and will therefore not be counted as a new one. Thus two states are only different, if and only if at least one of the ice blocks is on another position. We consider rotations of the board as unique.

Table 1 shows the number of unique states as found by our BFS algorithm. The horizontal axis shows the board size, the vertical axis the number of ice blocks on the board. Keep in mind that, for example, a 3×4 board gives the same results as a 4×3 board, so not all combinations are shown because this would be redundant. All boards are empty, except for the ice blocks (and targets) — there are no walls or floors. The ice blocks initially start along the edges of the board, for if we would place them randomly, the initial state might not be reachable. Since we are calculating the number of possible states, the position of targets is not relevant. Certain results only contain a dash: these turned out to be too computationally expensive.

	3×3	3×4	3×5	4×4	4×5	4×6	5×5	5×6	6×6	7×7
2 ice blocks	22	34	46	46	58	70	70	82	94	118
3 ice blocks	74	180	334	428	744	1140	1242	1844	2676	4874
4 ice blocks	121	448	1153	1656	4072	8270	9541	18600	34996	95685
5 ice blocks	122	744	2681	4148	14212	37752	45913	116516	284684	-
6 ice blocks	81	881	4577	7662	36552	125750	162546	-	-	-
7 ice blocks	32	746	5941	11020	73714	-	-	-	-	-
8 ice blocks	9	461	5962	12400	120165	-	-	-	-	-

Table 1: The sizes of search spaces of different empty square boards

When studying Table 1, two things stand out: First, we see that for boards with only two ice blocks, the search space increases in size by 12 for each increase of dimension (by one). The other thing one can notice is that the number of possible states does not increase indefinitely. This can be seen on the 3×3 board: the number of unique states increases until five ice blocks are on the board, after which this number decreases. This is obviously because the board becomes too full at some point when adding ice blocks: for example, when eight ice blocks are on a 3×3 board (which has nine fields in total), there is only one empty space. This empty space can be one of all nine fields, therefore there are only nine possible unique states. We briefly mentioned in Section 5.1 how adding more ice blocks indefinitely would, in the end, hinder movement. We will provide a calculation for the size of the search space of empty puzzles on square boards with two and three ice blocks in, respectively, Section 6.1 and Section 6.2.

Table 2 shows the number of unique states as found by our BFS algorithm, this time for the puzzles that appeared in the Legend of Zelda series. Note, here, that the leftmost vertical column refers to a particular puzzle from Appendix A. The abbreviations are made by first taking the name of the game, then the name of the location it appeared, then the number (if there were more puzzles at that location). So, for example, the puzzle from Ocarina of Time in the Ice Cavern is called OOTIC, the second puzzle from the Ice Block Cavern from Twilight Princess is called TPIBC2, etc.

Note that the search space may differ compared to the search space of the original puzzle. The MCTOD puzzle has a search space so large that it was not calculated. Its search space is much larger than it originally was, because ice blocks do not fall in water and disappear like in the the game, but can be put against the walls at the edge of the board.

	Search Space
OOTIC	22
OOTIR	15
OOSSAS	5477
MCTOD	-
TPSPR1	173
TPSPR2	1801
TPSPR3	1801
TPIBC1	334
TPIBC2	411
TPIBC3	1242
STST1	14
STST2	21
STST3	16
STST4	20

Table 2: The sizes of search spaces of the puzzles from Appendix A

TPIBC1 and TPIBC3 puzzle are equal to, respectively, the 3×5 and 5×5 puzzles with three ice blocks, so these have the same search space. The TPSPR puzzles are more oddly-shaped. These puzzles have 4×5 empty fields, below a row of 4 empty fields and below that one empty field. In total, this puzzle has 25 empty fields. The TPSPR1 puzzle, with two ice blocks, has a search space of 173, which is much larger than the 5×5 puzzle with two ice blocks, which has only 70 unique states. The TPSPR2 and TPSPR3 puzzles have three ice blocks and are both played on the same board, hence they have the same search space of 1801. Which is quite a bit larger than the search space of a 5×5 board. We can conclude that oddly-shaped boards greatly increase the number of unique reachable states. As we have shown in Section 5, three ice blocks are able to reach every single space on the board. While fixed normal floors will not be as versatile, they still greatly increase the number of states that can be reached, because the ice blocks can be pushed against the odd shaped of the board in order to reach new states, compared to an empty, square board. We do not compare the other puzzles because their shapes are too different from the square boards.

6.1 The search space with two ice blocks

We define $A(2, m, n)$ as the number of unique possible states a puzzle with two ice blocks on an m by n board can have. The computation of this quantity is split in several parts.

First, we consider all possibilities where both ice blocks have to be in a different corner. For each boards, regardless of its size, there are $\binom{4}{2} = 6$ possible ways the ice blocks can be put in corners. The first part of the equation is therefore 6.

Next, we consider all configurations with two adjacent ice blocks on the edge of the board. Since we have two ice blocks, we can move the ice blocks like a “train” alongside the edge of the board, as described in Section 5.1. We have two sides with length m and two sides with length n. Since our trains have a length of two, there can be $m-1$ trains on both sides with length m and $n-1$ trains on both sides with length n. This results in the following part of the equation: $2 \cdot (m-1) + 2 \cdot (n-1)$.

At any point, we can remove either one of the two ice blocks from the train and park it in a corner while leaving the other behind (at the edge of the board). We can leave the edge ice block at $m-2$ and $n-2$ positions for both edges with length m and n respectively. The minus two is there, for there are two corners per edge where we cannot leave this ice block: this would result in a double counting of the corner configurations, since the other ice block is parked in the corner. This gives us $2 \cdot (m-2) + 2 \cdot (n-2)$ positions for the edge ice block. The other ice block, which we do actually want in a corner, can be in either of the four corners. Therefore, we need to multiply the previous part by four: $4 \cdot (2 \cdot (m-2) + 2 \cdot (n-2))$. There is one other important subtraction we need to make here: if an ice block is originally adjacent to a corner, the corner ice block can be placed here. This will result in a double counting of the “trains” we already counted earlier. This can occur in eight instances (two per edge), meaning we need to subtract eight from this equation: $4 \cdot (2 \cdot (m-2) + 2 \cdot (n-2)) - 8$.

The last part is similar. We split up the train, but do not place one of the ice blocks in a corner. We leave it at the opposite side. Considering we can make $2 \cdot (m-1) + 2 \cdot (n-1)$ distinct trains, we need to subtract the cases in which sliding one of the ice blocks to the opposite side would result in that ice block being in a corner. This happens on two occasions per side, when the train is in one of the corners. The last part of the equation is therefore: $2 \cdot (m-1) + 2 \cdot (n-1) - 4 \cdot 2$.

No new states can be reached if we continue sliding the ice blocks. It is not possible for ice blocks to be removed from the borders.

If we add all these parts together, we get:

$$\begin{aligned}
A(2,m,n) &= 6+2\cdot(m-1)+2\cdot(n-1)+4\cdot(2\cdot(m-2)+2\cdot(n-2))-8+2\cdot(m-3)+2\cdot(n-3) \\
&= 6+2m-2+2n-2+4\cdot(2m-4+2n-4)-8+2m-6+2n-6 \\
&= 6+2m-2+2n-2+8m-16+8n-16-8+2m-6+2n-6 \\
&= 2m+2n+8m+8n+2m+2n-6+6-2-2-16-16-8-6 \\
&= 12\cdot(m+n)-50
\end{aligned}$$

We see that the results obtained by the BFS algorithm are in line with the formula derived above for $m+n>4$. E.g. if $m=n=3$, then $12\cdot(3+3)-50 = 72-50 = 22$, which is correct. Likewise, if $m=n=4$, then $12\cdot(4+4)-50 = 96-50 = 46$, and if $m=n=5$, then $12\cdot(5+5)-50 = 120-50 = 70$, etc. One exception is $A(2,2,2)$ for which the equation gives -2 as a result, which is obviously impossible. The actual number of possible states is six: a two by two field only has fields that are corners. Therefore, the first part of the equation is equal the total number of possible states, which is six.

6.2 The search space with three ice blocks

We define $A(3,m,n)$ as the number of unique possible states a puzzle with three ice blocks on an m by n board can have. The computation of this quantity is also split into several parts. All mentioned states can indeed be reached. We sketch a proof that these are the only states that can be reached.

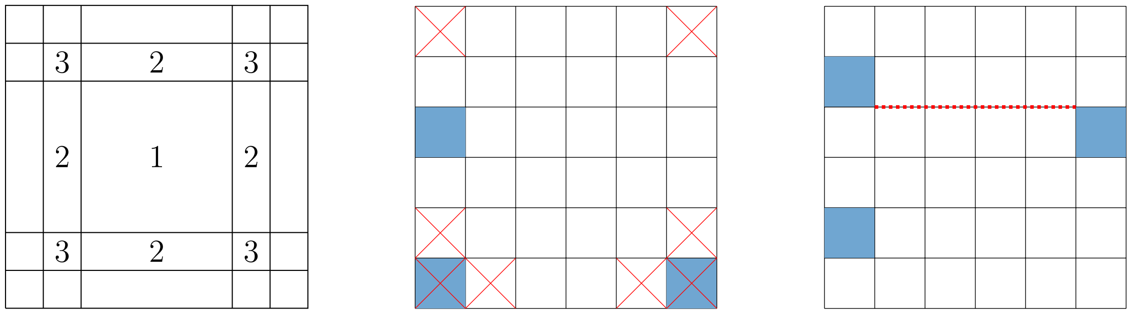
As with the computation of $A(2,m,n)$, we need to consider trains in the puzzle. Since we now have three ice blocks, we can both make trains with two ice blocks, called 2-trains, and trains with three ice blocks, called 3-trains.

First, we consider the number of 3-trains we can build along the border of the board. The border of a board always consists of $B = 2\cdot(m+n)-4$ fields, so there are B 3-trains at the border of the board.

Next, we consider the number of 2-trains we can build along the border of the board. The third ice block can be on almost any field (see Section 5). There are just as many 2-trains possible as there are 3-trains, which is B. This needs to be multiplied by the number of possible positions for the third ice block, which is $m \cdot n$, from now on referred to as S. However, there are a few restrictions: The third ice block may not lay in or be adjacent to the 2-train, which means that six needs to be subtracted from S. Then, however, we omit too many possibilities: when the 2-train lays in the corner (there are eight of these situations), there are only five fields occupied by or adjacent to the 2-train, so we need to add 8 again, giving a total of $B \cdot (S - 6) + 8$ possibilities.

Then, there is the number of reachable positions for the 2-train where the train is not at a border (see Section 5). The 2-train can be in any of $m - 1$ positions in every column except the first and last two. The remaining, third ice block can be in any of 8 positions, for a total of $8 \cdot (n - 4)(m - 1)$ possibilities. In the second and one-to-last column, the third ice block can be at 6 instead of 8 positions, for a total of $6 \cdot 2 \cdot (m - 1)$ possibilities. Adding these and simplifying gives $4 \cdot (m-1)(2n-5)$ states. A similar number holds for the rows, so $4 \cdot (n-1)(2m-5)$ states.

Next, we consider situations without trains. We can first put one ice block in the middle part of the field, which has a size of $(m - 4)(n - 4)$ fields. This is the subfield labeled as '1' in Figure 18a. We do this by building a 2-train towards the desired position, and thereafter sliding one of the two ice blocks to the side of the field. This gives 24 possibilities: 16 if we move one of two ice blocks of the train sideways and 8 if we move the head of the train, so $24 \cdot (m - 4)(n - 4)$ in total. However, after doing this, we can create a 2-train with the two ice blocks at the edges and bring these to some other position, as long as we finish by splitting the train in two (and having no ice blocks in corners). This gives us an extra of $2 \cdot (m - 4)(n - 4)(n - 5)$ possibilities for the rows, and $2 \cdot (m - 4)(n - 4)(m - 5)$ for the columns, the $m - 5$ and $n - 5$ being there for there are that many trains that are possible in each row or column, excluding the first and last two, totalling to $24 \cdot (m - 4)(n - 4) + 2 \cdot (m - 4)(n - 4)(n - 5) + 2 \cdot (m - 4)(n - 4)(m - 5)$.



(a) The three subfields (b) Exactly two in corners (c) All at a border, no corners

Figure 18: Some illustrations

The second and one to last row and column, excluding its corners, labeled as subfields '2' in Figure 18a consist of $2 \cdot (m + n - 8)$ fields. The other two ice blocks can be in 12 positions, for a total of $24 \cdot (m + n - 8)$ states. Again, we can then create a 2-train and create more possibilities. Since no ice block may be in the corners, we can add $4 \cdot (m + n - 9)(m + n - 8)$ for a total of $4 \cdot (m + n + 8)(m + n - 3)$ possibilities.

In the previous part, we omitted the corners of the second and one to last row and column. There are, of course, 4 corners. For each of these corners, the other ice blocks can be in six positions. Again, we can create a 2-train and create more possibilities, for a total of $8 \cdot (m+n-5)$ states.

Now, we will consider all cases without trains where there is at least one ice block in a corner. First, when all ice blocks are in a corner, there are four possibilities, regardless of the board size. Two ice blocks in corners have $\binom{4}{2} = 6$ possibilities, the third ice block having $S - 8$ possibilities as seen in Figure 18b, so $6 \cdot (S - 8)$. With only one ice block in a corner, the other two can be placed at all places on the boundary, which is $\binom{B-6}{2}$. We do accidentally count some trains, so the total with one ice block in a corner is $4 \cdot (\binom{B-6}{2} - (B - 10))$. Lastly, there are the cases where there is one ice block in a corner, one on the boundary and one in the middle of the field. Trains are still not allowed! This gives a total of $4 \cdot (B - 6)(S - B - 1)$ possibilities.

Next, we consider all three ice blocks at the border, but none are allowed in the corners, and neither are trains. We can create 3-trains and slide the middle ice block away for a total of $2 \cdot (m + n - 8)$ possibilities.

Last, we consider cases where all three ice blocks lie alone at the border, with no ice blocks on corners or any trains. Note that at least two of the ice blocks need to be opposite to each other, with one row or column difference, as seen in Figure 18c. It is also possible this is the case for both, in which case there are $2 \cdot (m + n - 8)$ possibilities. When there is only one pair of ice blocks opposite to each other with one row difference, there are $4 \cdot (B - 10) + 2 \cdot (m - 5)(B - 12)$ possible states. The same holds for the columns, so $4 \cdot (B - 10) + 2 \cdot (n - 5)(B - 12)$ states.

When adding all these parts and simplifying them greatly, we conclude:

$$A(3, m, n) = 492 - 100x + 12xy - 18y - 8x^2$$

$A(3, m, n) = 492 - 100(m + n) + 12(m + n)(m \cdot n) - 18(m \cdot n) - 8(m + n)^2$
Here $x = m + n$ and $y = m \cdot n$.

7 Difficulty

As can be seen in Appendix A, there is a very wide variety of Ice Block Puzzles, resulting in puzzles of a wide range of difficulty. There are puzzles with only one ice block, puzzles with multiple ice blocks, puzzles with multiple ice blocks of which not necessarily all need to be moved, etc. Sometimes the player needs to change which ice block he pushes next very often, sometimes not. Assigning a difficulty to a certain puzzle is therefore quite hard. We asked ourselves the question: “What makes a puzzle hard?” It seems reasonable that frequently having to push another ice block than the ice block we pushed in our previous move (from now on referred to as blockchange) makes a puzzle hard. Imagine what would be harder to solve: a puzzle where one ice block needs to be pushed ten times, or a puzzle where ten blocks need to be pushed once. A puzzle that only requires one ice block to be pushed has a solution that can usually be traced back by looking at the board. There needs to be a predetermined path that the ice block has to follow, since there are no other ice blocks which need to be pushed to make this path possible. There are some exceptions, such as the puzzle from Minish Cap (shown in Appendix A), where only one ice block needs to be pushed but where more ice blocks can be pushed, which makes finding the solution harder. However, we argue that a puzzle that has ten ice blocks that each need to be pushed only once requires great insight of the player: nine ice blocks need to be pushed in such a way that the state of the board changes and that final ice block to be able to reach the target.

This makes it seem as if the number of blockchanges is a good difficulty measure — and to a certain extent, it is. However, only considering the number of blockchanges disregards the difference in difficulty between puzzles that only require one ice block to be moved. It is also wrong to assume that a puzzle with zero blockchanges that can be solved in just one move is as hard as a puzzle with one ice block that needs to be pushed a million times.

Now, it seems tempting to just multiply the total number of moves with the number of blockchanges. This is, however, also not a right difficulty measure: imagine a puzzle with ten blockchanges and ten moves, which would get a difficulty of $10 \cdot 10 = 100$. A puzzle with one blockchange and one hundred moves would also get a difficulty of $100 \cdot 1 = 100$. We feel that the first puzzle would be harder, because, as described earlier, it seems like this would require more insight from the player.

In the end, we decided on the following formula for the difficulty of a solution: difficulty = total number of moves + number of blockchanges · user-defined factor. This makes sure that puzzles with only one ice block do have varying difficulties (which will here be based on the number of moves necessary to solve the puzzle), but also ensures that puzzles with a high number of blockchanges may be considered more difficult, depending to the user-defined factor.

So far, we have only talked about the difficulty of a puzzle in terms of the easiest solution as found by P-BFS (see Section 8). However, now that we have got a difficulty measure, we can assign this measure to every single reachable state of a puzzle.

It is, however, also interesting to calculate the difficulty of each and every reachable state and to see how often each difficulty occurs, depending on the blockchange factor. We have calculated this for a selection of puzzles, some as seen in Appendix A. The results of these puzzles can be found in Section 8.

8 Results

In this section, we discuss if and how fast our four algorithms were able to solve the puzzles and compare the results. Furthermore, we show the influence of the blockchange factor on the difficulty of the reachable states, as discussed in Section 7. We have tested all puzzles shown in Appendix A, but we mostly discuss the puzzles with more than one ice block, since these are the most interesting to solve.

8.1 The algorithms and their ability to solve puzzles

We will first discuss how well our four algorithms were able to solve the puzzles, and, if applicable, the tweaking of these algorithms, and how these affected their ability to solve the puzzles.

Random Algorithm

We have run the random algorithm a few times for all puzzles, since not all puzzles were solved in the first try. As expected, the random algorithm was neither very good nor fast at solving puzzles. It was usually able to solve relatively simple puzzles, such as puzzles with only one ice block. Irreversible puzzles could usually not be solved, since this algorithm has no ability to check if a move might result in the puzzle not being able to be solved anymore. For example, the random algorithm could not solve the puzzle from Oracle of Seasons or the puzzle from Minish Cap in our runs. Both of these puzzles are surrounded by walls. If an ice block were to be pushed against these walls, the puzzle becomes unsolvable, so it is very easy for these puzzles to get in a state from which no solution can be reached. The third puzzle from Spirit Tracks was also not always solved. Even though this puzzle is very simple (it has only one ice block), it has a wall: if the ice block is pushed against this wall (which the random algorithm would sometimes do), the puzzle cannot be solved anymore. The random algorithm was able to solve the first puzzle from the Snow Peak Ruins from Twilight Princess, which has two ice blocks. This puzzle is not particularly hard, but it does require both ice blocks to be moved to solve it. The second (and also the optional variation) was not solved. The random algorithm found a solution to the first puzzle from the Ice Block Cavern, also from Twilight Princess, which features three ice blocks. This was probably only due to the small dimensions of the board. It was never able to solve the third puzzle, which is quite similar in nature, but features a larger board. Surprisingly, it has solved the second puzzle from the Ice Block Cavern as well, on very rare occasions. This puzzle is actually quite hard and, although it took very long and many moves, we have seen random solve this puzzle once or twice. Solving puzzles like this one, the third puzzle from the Ice Block Cavern, the second from the Snow Peak Ruins, the one from Minish Cap and the one from Oracle of Seasons require either quite some insight by the player or a specific strategy, like repeating pattern of moves. These are types of puzzles the random algorithm seems unable to solve.

Of course, the random algorithm can theoretically solve all puzzles if it gets very lucky; it is just not very likely to happen.

Brute Force Breadth-First Search

Brute Force Breadth-First Search (BFS) is not very fast in finding a solution, but is able to find a solution to any solvable puzzle. BFS is not necessarily slower than BasicMC, especially when BasicMC is given a lot of random games to play. BFS always finds one of the fastest solutions, that is, it finds a solution that requires the minimum number of moves. Interestingly, for the puzzle from Oracle of Seasons and the Ice Block Cavern puzzles featured in Twilight Princess, BFS found a solution that required less moves than the solutions that online solutions in walkthroughs for the game suggested. These puzzles mentioned above are also the hardest puzzles from all that have featured in the Legend of Zelda, since these are the few that contain three ice blocks all of which need to be pushed. These were therefore also the most likely to have multiple solutions, contrary to all puzzles with only one ice block. The fact that BFS finds a faster solution than these walkthroughs suggest, might be because walkthroughs might prefer solutions with a lower difficulty, that is less blockchanges, as is discussed in Section 7, since these solutions are somewhat more easy to follow.

Brute Force Pruned Breadth-First Search

Brute Force Pruned Breadth-First Search (P-BFS) is much slower than BFS. In contrast to BFS, it does not stop looking for a solution as soon as it finds one. It continues and tries to find the easiest one, based on the difficulty measure, that gives penalties for switching ice blocks and depends the blockchange factor. In addition, P-BFS might need to recalculate a lot of states if it finds a previously encountered state. If this state has been found in an easier manner than before, all states that follow need to be recalculated. Even if the same state is found with the same or a higher difficulty, but with a different last used ice block, states need to be recalculated: the last used ice block affects the difficulties of the following states due to the blockchange factor. If a certain ice block needs to be used to reach a state, the difficulty of this state will be lower if the last used ice block is the same as the ice block used to reach the new state. For example, imagine we encounter a state with difficulty 18 and last used ice block 1 (situation 1), but we already encountered it with difficulty 16 and last used ice block 2 (situation 2), it is possible that reaching a new state may require ice block 1 to be moved. If the blockchange factor is 5, this will result in a difficulty of 19 for situation 1 (the last used ice block was 1, so no blockchange factor), while this will result in a difficulty of 22 for situation 2 (one move extra and a penalty of five for the blockchange). We therefore still need to calculate the difficulties for both situation and the states that may follow from those states. We only do not need to do this if the difference in difficulty between both situations exceeds the blockchange factor.

Interestingly, depending on the blockchange factor, it does find different solutions. Obviously, P-BFS with a blockchange factor of 0 finds the same solution as BFS. Of course, BFS or P-BFS with a blockchange penalty of 0 both find a solution which requires the smallest number of moves. However, with a blockchange penalty greater than 0, P-BFS will find other solutions with more moves but less blockchanges. Due to the very frequent recalculations P-BFS has to execute, some puzzles took a very long time to solve. Because P-BFS also does not stop calculating after finding one solution, it has to consider every single state. Therefore, the larger the search space of a puzzle, the longer it takes for P-BFS to solve it. The number of ice blocks also has a huge impact on the duration of P-BFS, because, as mentioned before, if a state was reached with a different last used ice block, P-BFS still has to calculate the following states. The more ice blocks there are, the more states P-BFS needs to calculate, even when encountered earlier with a different last used ice block. The puzzle from Minish Cap have a (relatively) much larger search space than all other puzzles from Appendix A and was not solved by P-BFS simply because it took too long (more than several days).

Monte Carlo

BasicMC gave much better results than the random algorithm. Puzzles with only one ice block were solved with hardly any trouble. However, it would still get stuck on some puzzles. This is because BasicMC uses the random algorithm to give a score to moves. If the random algorithm is not able to find a solution, BasicMC cannot assign scores to the moves and will make a random move as well. BasicMC therefore also had a harder time with the puzzles the random algorithm could not solve (in reasonable time). However, because BasicMC was given 250,000 random games to play for every move and the random algorithm might get a correct solution once or twice, this would enable BasicMC to find a correct solution.

In addition, when getting closer to the solution in a puzzle with multiple ice blocks, two very similar game states may result in similar scores. BasicMC allows the random algorithm to make a user-defined number of moves to find the solution. When getting close to the solution, two similar states can both be very close to a solution. For example, if both of these states can be reached from another, both are close to the solution. This resulted in the random algorithm being able to solve the puzzle in both of these states, which caused the moves leading to those states getting high scores. BasicMC would then make and unnecessary or wrong moves, such as moving a single ice block back and forth forever. Therefore, we decided to let the number of moves that the random algorithm can make decrease. With every move BasicMC makes, the number of moves the random algorithm is allowed to make (and also the number of moves BasicMC has left) decreases. This fixed the problem of BasicMC moving ice blocks back and forth endlessly.

BasicMC initially also struggled with some of our puzzles with more than one target. At first, states would get points for every target that had an ice block placed on it. This resulted in states getting high scores with only one ice block on one target, regardless of what moves the algorithm would make. BasicMC would and slide ice blocks back and forth until it ran out of moves, and no ice block would be placed on the other target(s). We therefore only gave states points when all targets have an ice block on them. This resulted in the TPIBC2 puzzle being solved as well.

We have tried different numbers of random games for BasicMC. We tried 100,000 random games, but we found that at least the TPIBC3 puzzle was not solved. We decided to increase the number of random games to 250,000, which would give pretty good results: all puzzles (except the Minish Cap puzzle) were solved (which further proves that the random function can, in theory, solve these puzzles; otherwise, BasicMC would have been left in the dark). Oddly enough, when BasicMC used an even larger number of random games (2,500,000), it gave different solutions. For example, we observed that the TPIBC3 puzzle was solved with one move less, but with a higher difficulty – remember that BasicMC does not take blockchanges into account. This illustrates that the number of random games have a huge impact on the solution and overall success of BasicMC. We decided that 250,000 was a good number of random games, since BasicMC was able to solve all puzzles (except the Minish Cap puzzle) with a correct solution, while also being relatively fast.

In our experiments, we set the maximum number of moves BasicMC is allowed to do at 15. This number is quite close to the number of moves necessary to solve most of the harder puzzles. If BasicMC was given an excess number of moves over the number needed to solve the puzzles, it would result in BasicMC moving ice blocks back and forth, or reaching similar states to one that was already visited (by, for example, mirroring this state), until the number of moves BasicMC has left has decreased so much that the puzzle has to be solved (otherwise BasicMC will run out of moves before solving it. All puzzles from Appendix A were solved by giving BasicMC 15 moves. Giving the exact minimum number of moves required to solve a puzzle was often too hard for BasicMC, especially on puzzles with more than one ice block.

Overall, BasicMC has pretty good results. It does not necessarily find the easiest or fastest solution, but it does find a solution on most puzzles, especially when given the aforementioned large number of random games.

8.2 Solutions of the algorithms compared

We thought it would be interesting to investigate how good the solutions found by BasicMC, BFS and P-BFS are compared to the optimal solution available, both with respect to their length and difficulty value. We chose an arbitrary blockchange factor of 5. Keep in mind that BasicMC and BFS do not take this penalty or its number of moves into account while finding a solution: it only calculates the difficulty of the solution found by the algorithm, which we compare against the scores of the solution that P-BFS found. Furthermore, since BasicMC and BFS do not consider blockchanges while choosing their moves, there may have been moves that resulted in a less difficult solution. For example, one can see that the solution for the TPSPR1 puzzle found by BFS and BasicMC had five moves and two blockchanges, while P-BFS also solved the puzzle in five moves with just one blockchange. We have given BasicMC a maximum number of moves of 15, since all of our puzzles can be solved in 12 moves or less. We want to mention that BasicMC sometimes (or on average) gave better results with a maximum number of moves closer to the number of moves that BFS took to solve the puzzle, but we show the results with a maximum of 15 moves for every puzzle for consistency. BasicMC was always given $250,000$ random games to play per state, because in that case BasicMC was still relatively fast and also capable of solving most puzzles on most occasions. We ran BasicMC multiple times and chose the best results BasicMC has calculated.

Table 3 shows solutions as found by the algorithms, compared. Fields with a dash were either too computationally expensive (P-BFS) or not solved (BasicMC).

	BFS			P-BFS			BasicMC		
	#moves	#blockchanges	difficulty	#moves	#blockchanges	difficulty	#moves	#blockchanges	difficulty
OOTIC	7	0	7	7	0	7	7	0	7
OOTIR	3	0	3	3	0	3	3	0	3
OOSSAS	9	4	29	9	3	24	9	5	34
MCTOD	5	0	5	-	-	-	-	-	-
TPSPR1	5	2	15	5	1	10	5	2	15
TPSPR2	8	3	23	8	2	18	9	2	19
TPSPR3	12	4	32	12	3	27	-	-	-
TPIBC1	7	4	27	8	2	18	7	4	27
TPIBC2	10	6	40	14	3	29	11	6	41
TPIBC3	11	5	36	13	3	28	11	5	36
STST1	5	0	5	5	0	5	5	0	5
STST2	7	0	7	7	0	7	7	0	7
STST3	6	0	6	6	0	6	6	0	6
STST4	6	0	6	6	0	6	6	0	6

Table 3: A comparison between the solutions found by the algorithms

Remember that BFS calculates a solution that has the least number of moves, that is, one of the fastest solutions, while P-BFS calculates one of the least difficult solutions. As can be seen in Table 3, the results of BFS and P-BFS indeed differ in this manner.

What stands out is that BasicMC finds a solution that approaches or is the same as either the one found by BFS or by P-BFS. These are, however, cherry-picked, and we did see BasicMC sometimes requiring the full 15 moves to solve the puzzle, sometimes also reaching difficulties higher than 60. BasicMC was also not always able to solve the TPIBC3 puzzle with the given number of random games of 250,000. The other puzzles were always solved without problems, but still with varying results.

The algorithms all found the same solutions for the puzzles with only one ice block. Note that the MCTOD result for BasicMC was left blank; BasicMC was, however, able to solve this puzzle when given 9,999,999 random games to play (and with the usual 15 moves given). This puzzle is especially hard for BasicMC and the random algorithm because it has six ice blocks, of which only one needs to be moved five times. Doing any move other than required in the solution of this puzzle will result in the puzzle becoming unsolvable. Basically, the probability this puzzle is solved by the random algorithm is $(\frac{1}{6\times4})^5 \approx 1,256 \times 10^{-7}$. We tried to give BasicMC 9,999,999 random games for TPSPR3 as well, to no avail.

8.3 The influence of the blockchange factor on reaching states

We have run P-BFS on a few puzzles from Appendix A and a few empty puzzles with a rectangular board with different blockchange factors in order to plot how this factor influences the number of states that can be reached with a certain difficulty. We have run all of these puzzles with blockchange factors 0, 1, 2, 3, 4, 5 and 100. The TPIBC1 and TPIBC3 puzzles are both empty puzzles with a square board and were therefore also calculated. Furthermore, we ran these tests on the OOSSAS, TPIBC2 and TPSPR puzzles from Appendix A. We have chosen factors 0 through 5 to show little changes in the blockchange factor have effect on the cumulative number of reachable states with a certain difficulty value (or less). In addition, we chose the blockchange factor of 100 in order to illustrate how many states can be reached with the least possible number of ice block changes. A penalty of 100 is so high that the algorithm would only switch ice blocks if strictly necessary for reaching new states. We have normalized the calculated difficulties by dividing them by the highest found difficulty, because the difficulties would otherwise get much higher with every increase in blockchange factor. Not normalising the difficulty would result in confusing graphs. The following graphs show, per puzzle, the difficulty measure, normalized, plotted against the cumulative number of reachable states. In other words: the number of states on the vertical axis can be reached with the normalized difficulty (or lower) on the horizontal axis. For all puzzles, we took the starting position where all ice blocks are in a (different) corner of the board. Keep in mind that the size of the search space of the puzzles can be found in Section 6, Table 1 and 2. An example to read these graphs: in Figure 19, for blockchange factor 100 (the green curve), there are round 120 states that can be reached with (normalized) difficulty 0.5 or lower.

As can be seen in Figure 19, there is only a slight difference between the different blockchange factors. A factor of 0 gives a very smooth curve. Factors 1 through 5 seem to wave around each other. Obviously, factor 100 gives a very jagged, “stair-case”-like line, due to the incredibly high spikes in difficulty per blockchange.

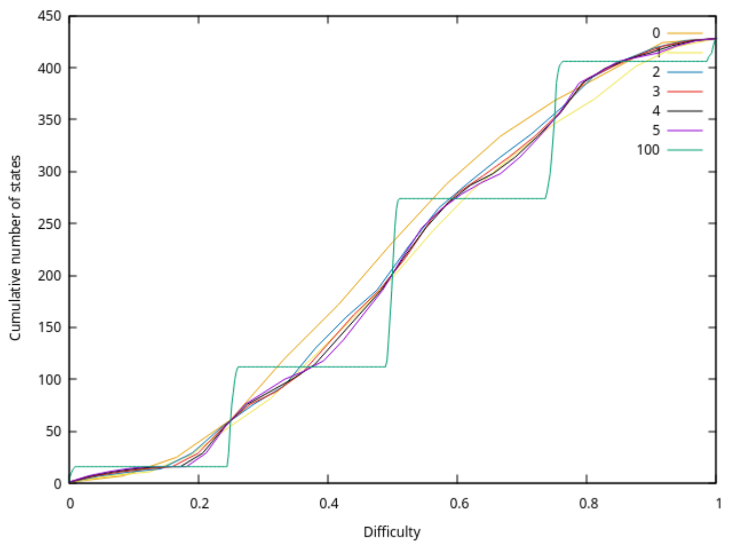


Figure 19: A 4×4 board, with 3 ice blocks

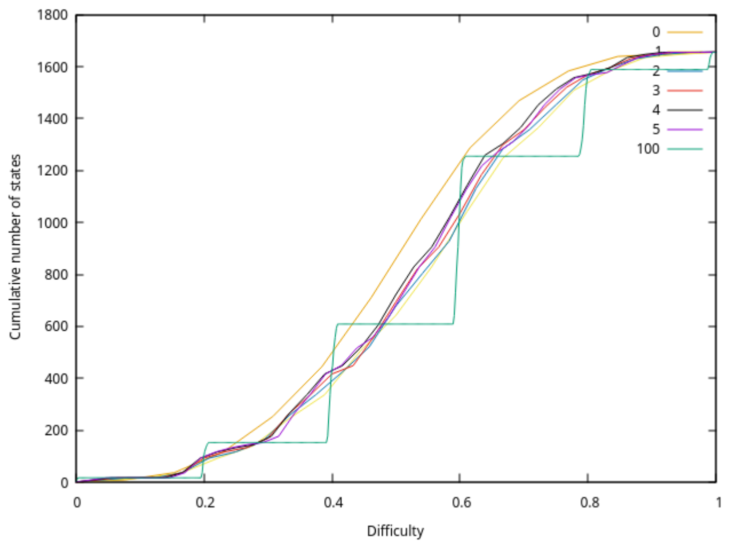


Figure 20: A 4×4 board, with 4 ice blocks

Since calculating these graphs is very computationally expensive, we were only able to make a graph of a puzzle with four ice blocks for a 4×4 board, as seen in Figure 20. Compared to Figure 19, the lines from difficulties 2 through 5 are somewhat more jagged, but overall, the graphs are quite similar.

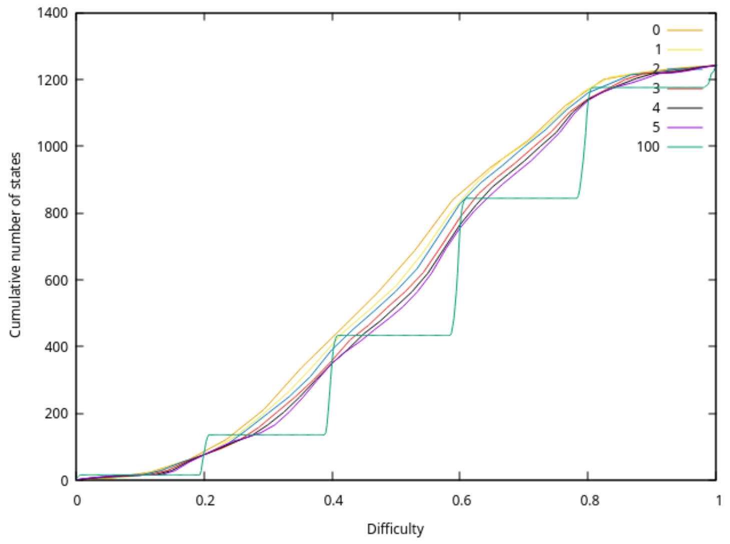


Figure 21: A 5 × 5 board, with 3 ice blocks (TPIBC3)

Figure 21 shows more parallel curves between the blockchange factors. The shape of the curves for the different blockchange factors (except 100) is very similar, but they differ slightly in height: they get lower with each increase in blockchange factor, which means that less states can be reached for that difficulty.

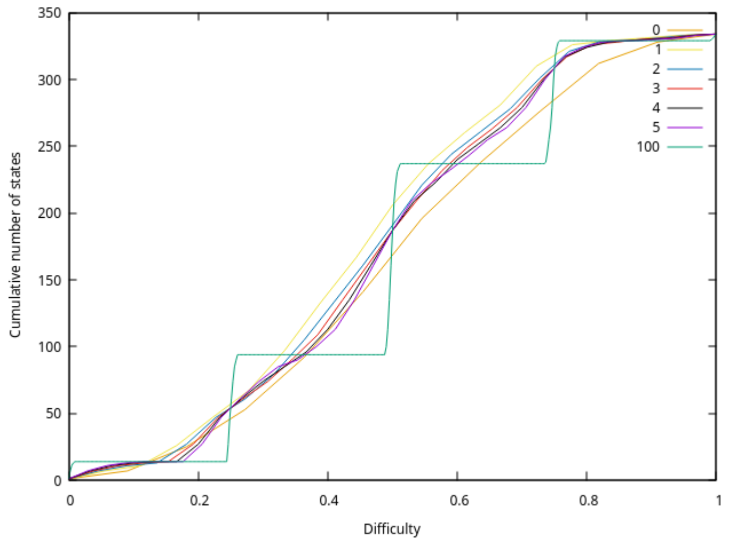


Figure 22: A 3 × 5 board, with 3 blocks (TPIBC1)

Figure 22 has somewhat larger discrepancies between the different blockchange factors.

Just as a fun exmaple, we also have graphs for the OOSSAS, TPSPR1 and TPSPR2 (which is the same as the TPSPR3 puzzle) and TPIBC2 puzzles, which can be seen in Figures 23, 24, 25 and 26.

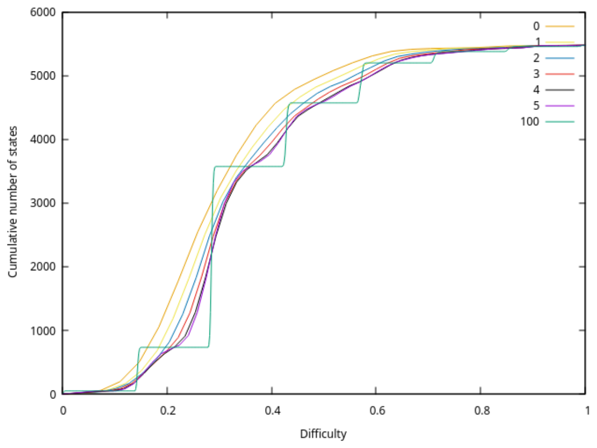


Figure 23: The OOSSAS puzzle

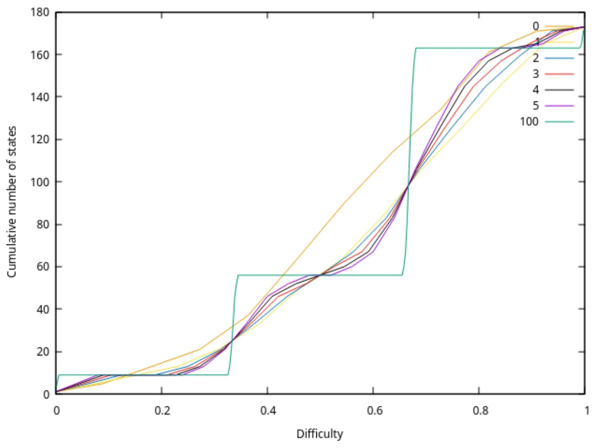


Figure 24: The TPSPR1 puzzle

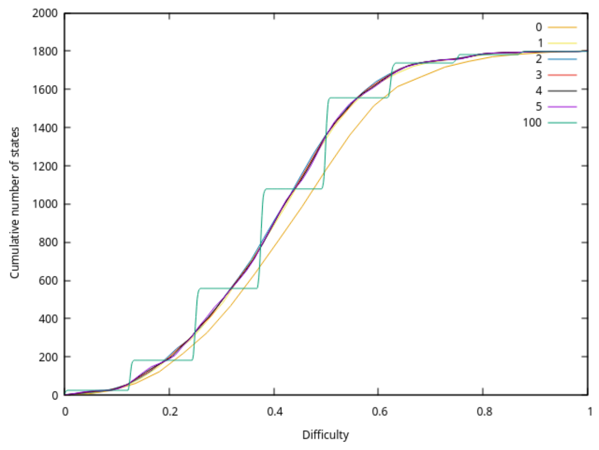


Figure 25: The TPSPR2 and TPSPR3 puzzle

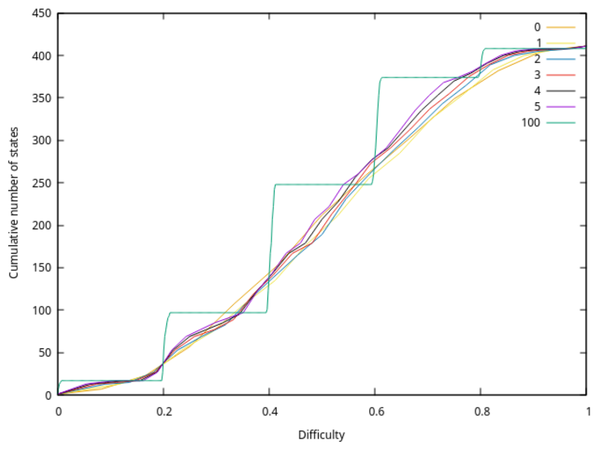


Figure 26: A 4×4 board with a floor in one corner and two ice blocks (TPIBC2)

The OOSSAS puzzle has a relatively interesting graph that is more curved and leans more on the left. This puzzle has an odd shape and consists of a lot of walls: the border of the board consists of walls, and some walls are scattered in the middle of the puzzle. There are a lot of irreversible states and moves, even a lot of states where no more moves can be done at all. This explains the fact that most states can be reached with a low difficulty: the higher the difficulty, the more chance that the puzzle is in a state where no more moves are possible.

When looking at all graphs, one can conclude that the more possible states there are, the smoother the graph is and the more the graphs converge. The TPSPR1 puzzle has relatively few possible states (only 173), compared to larger puzzles, and one can see how the graph smooths out and converges when there are more possible states. The largest puzzle we were able to calculate fully was a puzzle of 5×6 with 3 ice blocks, of which the graph can be seen in Figure 27, which shows a really smooth, converged line for all blockchange factors (except factor 100, of course).

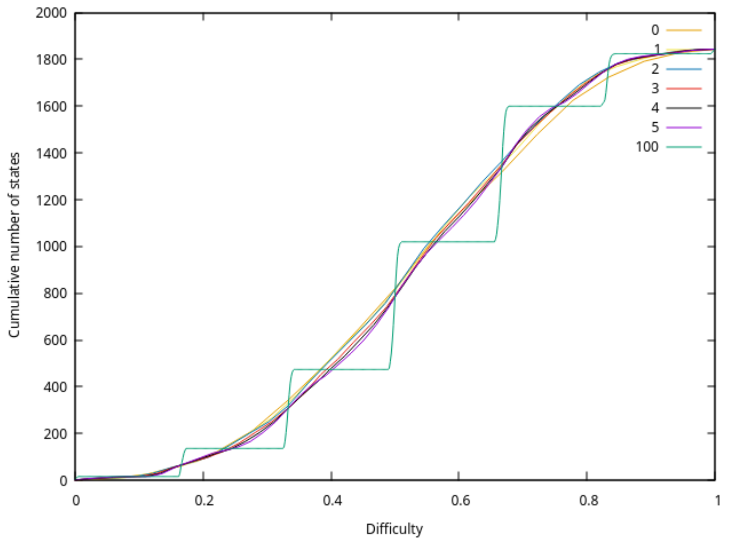


Figure 27: A 5 × 6 board, with 3 ice blocks

9 Conclusion

In this paper, we have taken an in-depth look at the ICE BLOCK PUZZLES often featured in the LEGEND OF ZELDA series. While similar, they are somewhat different from PUSH-PUSH. We have discussed a random algorithm, a BRUTE FORCE BFS algorithm, a slightly more sophisticated BRUTE FORCE BFS algorithm (called P-BFS) and a MONTE CARLO type algorithm (called BASICMC) to solve the puzzles. We also discussed an algorithm which produces new puzzles by generating them backwards: starting with a state in which the puzzle is solved, the algorithm builds a puzzle in reverse while putting objects on the board where necessary.

Next, we looked at some properties of the ICE BLOCK PUZZLES, such as the reversibility of states and moves. We showed some configurations of the board together with (series of) moves being irreversible: some states can not be reached (directly) anymore after a certain sequence of moves.

We described a strategy for finding a solution to puzzles on a square, empty board (that is, no obstacles), three ice blocks and one target.

We discussed the search space of ICE BLOCK PUZZLES and derived a formula, without proof, for square, empty boards (again without obstacles), that calculates the size of the search space for these boards with two and three ice blocks.

Next, we discussed what makes a puzzle difficult. We proposed a difficulty measure depending on the number of moves and a penalty for each time the algorithm had to change which ice block it would use in its next move, which we called the blockchange factor. Using our difficulty measure, we compared the results of our solving algorithms and their ability to solve some puzzles, including the ones as found in the LEGEND OF ZELDA series, which can be found in Appendix A. We found that BFS was always able to find one of the fastest solutions, that is, a solution requiring the least number of moves, while P-BFS finds the least difficult solution. BasicMC was, after some tweaks, also quite capable of solving some puzzles.

Finally, we plotted the normalized difficulty against the cumulative number of reachable states of some puzzles, which shows how the blockchange factor influences the number of states that can be reached with a certain difficulty. However, its impact is quite limited, especially on larger puzzles.

Future Work

As speculated in Section 5, there is a certain strategy involving building trains which allows ice blocks to reach every single field on the board. We conjectured that this strategy works as long for m targets and no obstacles and there are $m + 2$ ice blocks on the board. While this strategy looks promising, it gets harder to execute when the board becomes fuller, that is, when there are more and more ice blocks on the board. At a certain point, the movement of ice blocks is constrained due to the large number of ice blocks on the board. It might be interesting to find the actual upper bound for which this strategy still works.

We have also thought about a strategy where the puzzle can be solved row by row, beginning in the bottom row. Our idea involves a "supplier" of ice blocks at the top of the board (or any side of the board, for that matter), which would be able to provide ice blocks at the necessary field by pushing them down. The top of the board would in this case be filled with ice blocks (but in such a way that they can still be moved). This supplier would be able to move back and forth with the train construction discussed in Section 5 and push down blocks where necessary, working row by row. However, the board will fill up from the bottom, in the end constraining the movements of the supplier at the top of the board. We have not yet been able to decide whether this is a viable strategy, and if so, what its constraints are.

If any of these above strategies (or another that we did not think of) might be invented or elaborated further upon, a more efficient algorithm might be created, rather than BasicMC or (P-)BFS. As described in Section 5, most, if not all states can be grouped together in a larger set of states, for this state can be reached by other states in this set, and we do not know how close to the solution we are. There may be a way to figure out a measure how close we are to a solution in a certain state, which could make it possible to create a self-learning algorithm.

One could also research the impact of the number of random games played by BasicMC on the solution it finds or does not find. The impact of the blockchange factor on the reaching states could also be researched further.

In Section 6, we talked about the search space for Ice Block Puzzles. We took a more detailed look for the search space of puzzles with two and three ice blocks and derived a formula for the search space. However, we have not actually proven the correctness of this equation. There may even be a more general equation that works for any rectangular board, given its dimensions m and n and the number of ice blocks.

References

[BDF$^+$19] Aaron T. Becker, Erik D. Demaine, Sándor P. Fekete, Jarrett Lonsford, and Rose Morris-Wright. Particle computation: complexity, algorithms, and logic. *Nat. Comput.*, 18(1):181–201, 2019.

[Cul98] J. C. Culberson. Sokoban is PSPACE-complete. In *Proceedings International Conference on Fun with Algorithms (FUN98)*. Carleton Scientific, 1998.

[DDF$^+$18] Paul Dorbec, Éric Duchêne, André Fabbri, Julien Moncel, Aline Parreau, and Éric Sopena. Ice sliding games. *Int. J. Game Theory*, 47(2):487–508, 2018.

[DDO00a] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. *CoRR*, cs.CG/0007021, 2000.

[DDO00b] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush is NP-hard in 2D. *CoRR*, cs.CG/0001019, 2000.

[Dem] Erik D. Demaine. PushPush website. http://erikdemaine.org/pushpush/. Retrieved April 1, 2020.

[HD09] R.A. Hearn and E.D. Demaine. *Games, Puzzles, and Computation.* CRC Press, 2009.

[Hui17] Jarno Huibers. The behaviour of sliding particles using a global operator. Master's thesis, Leiden University, December 2017.

[OCC$^+$99] Joseph O'Rourke, Beenish Chaudry, Sorina Chircu, Elizabeth F. Churchill, Sasha Fedorova, Judy A. Franklin, Biliana Kaneva, Halley Miller, Anton Okmianski, Irena Pashchenko, Ileana Streinu, Geetika Tewari, Dominique Thiébaut, and Elif Tosun. PushPush is NP-hard in 3D. *CoRR*, cs.CG/9911013, 1999.

[SMD$^+$15] Hamed Mohtasham Shad, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete, and Aaron T. Becker. Particle computation: Device fan-out and binary memory. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 5384–5389. IEEE, 2015.

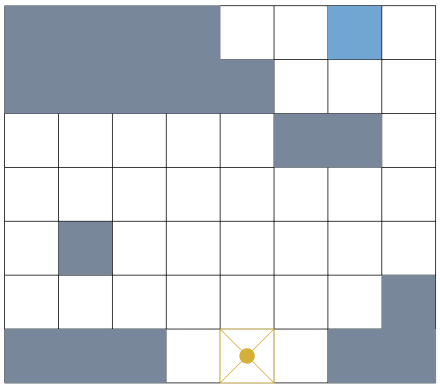
A Original Puzzles

In this appendix, we will show all puzzles found in the original The Legend of Zelda-series, ordered by release date and in the order which the player would encounter them. Some of the puzzles have been altered slightly, since our program does not support all types of blocks in the original game. However, these blocks do not change the outcome of the solution: the sequences of moves required to solve the puzzles are preserved. Recall that the given puzzle boards are surrounded by normal floors.

Some of the puzzles required a few changes. For example, some of the puzzles were originally surrounded by crevasses. If the player made a move that caused the ice block to fall into the crevasse, it would reset to its original position. This does not influence the sequence of moves required to solve the puzzles, which is why they are not included in the puzzles.

Ocarina of Time

The two puzzles shown in Figure 28 were originally surrounded by crevasses which have been omitted. This does not affect the solution to the puzzle.


(a) Ice Cavern

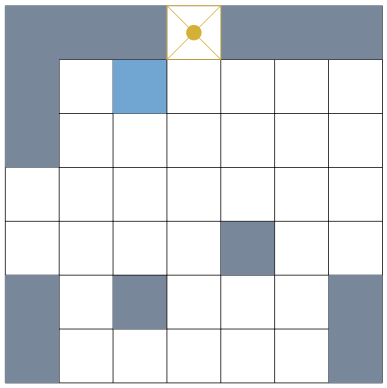

(b) Ice Room (Ganon's Castle)

Figure 28: Two Ice Block Puzzles from Ocarina of Time

Oracle of Seasons

The puzzle seen in Figure 29 is one of the largest puzzles in size and also one of the harder puzzles. This one required no modifications; it is an exact copy of the one in Sword and Shield maze from the 2001 game. This puzzle is surrounded by walls, so one wrong move can result in it being unsolvable.

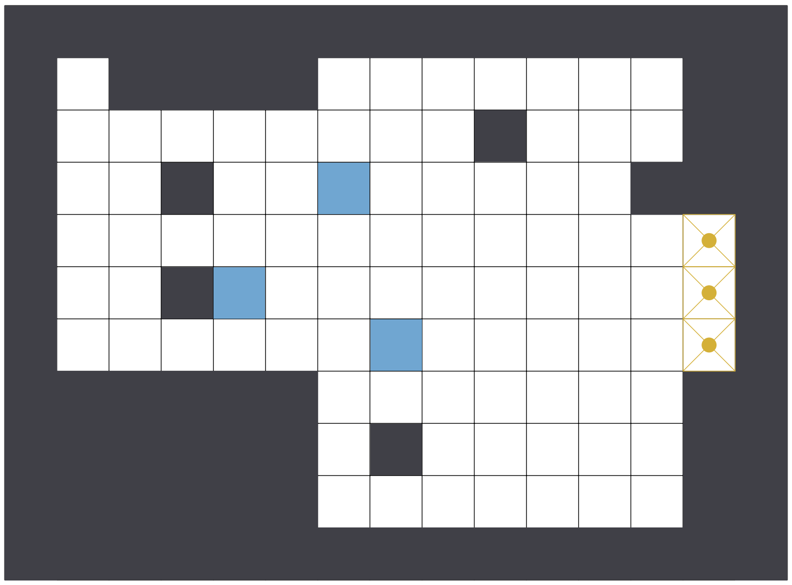


Figure 29: Ice Block Puzzle in Oracle of Seasons

Minish Cap

The puzzle as seen in Figure 30 is also one of the larger puzzles. Originally, this puzzle was surrounded by water. Sliding an ice block in the wrong direction would make the puzzle impossible to solve. The water has been represented as a wall, which has basically the same effect. This is the only puzzle that does not require all available ice blocks to move to complete the puzzle. In fact, the puzzle must be solved by moving just one of the ice blocks. It is therefore not very hard, even if the solution might not be obvious at first glance.

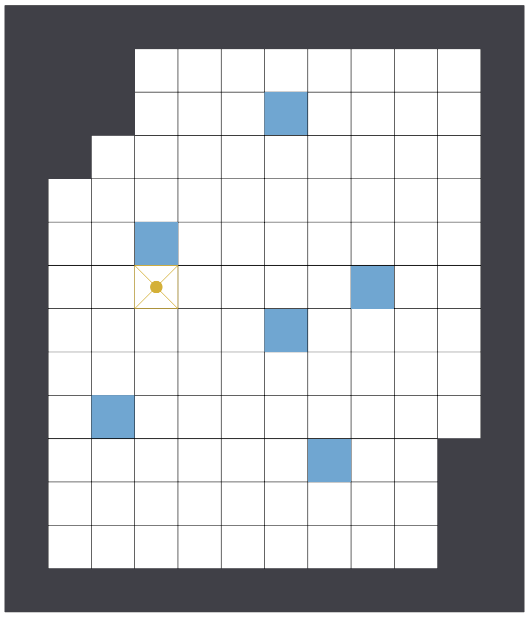


Figure 30: Ice Block Puzzle in the Minish Cap, from the Temple of Droplets

Twilight Princess

This 2006 game probably has the hardest ice block puzzles that ever occurred in the whole series. In addition, it has the most puzzles of any game in the franchise.

Snow Peak Ruins

The Snow Peak Ruins puzzle consists of two parts with slightly different lay-outs. The middle wall from part one, seen in Figure 31a, is actually a target frozen in ice, while the top right is another ice block which has been frozen shut. The second part, seen in Figure 31b has to be solved at a later point, when the target in the middle is reachable and the top right ice block can be moved.

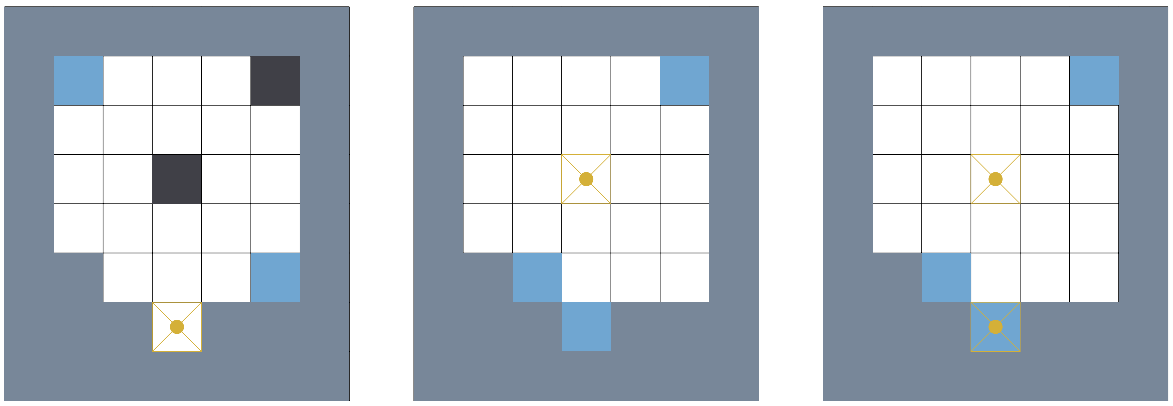

(a) Part One (b) Part Two (c) Optional variation

Figure 31: Snow Peak Ruins puzzle

While it is not strictly necessary to have an ice block on both targets, we thought it would be fun to consider this puzzle in the case we would have to place an ice block on both targets. We decided that the initial state of this variation would be the same as the starting state of part two. See Figure 31c.

Ice Block Cavern

The Ice Block Cavern consists of three different puzzles in three different rooms. Their lay-out is represented in Figure 32. The puzzle in the first room, seen in Figure 32a, is quite small in terms of dimensions, but needs a certain strategy and is harder than one would guess at first glance. Ice blocks have to be placed in such ways that another ice block can eventually reach the target. The second room contains the puzzle seen in Figure 32b. Surprisingly, considering how hard all Ice Block Puzzles are in Twilight Princess, this is the only one that actually requires two targets to have an ice block on them in the whole game. The last puzzle, shown in Figure 32c, is in terms of solution quite similar to the first puzzle from the Ice Block Cavern. The first puzzle and this third one are actually very interesting and are discussed further in Section 5.

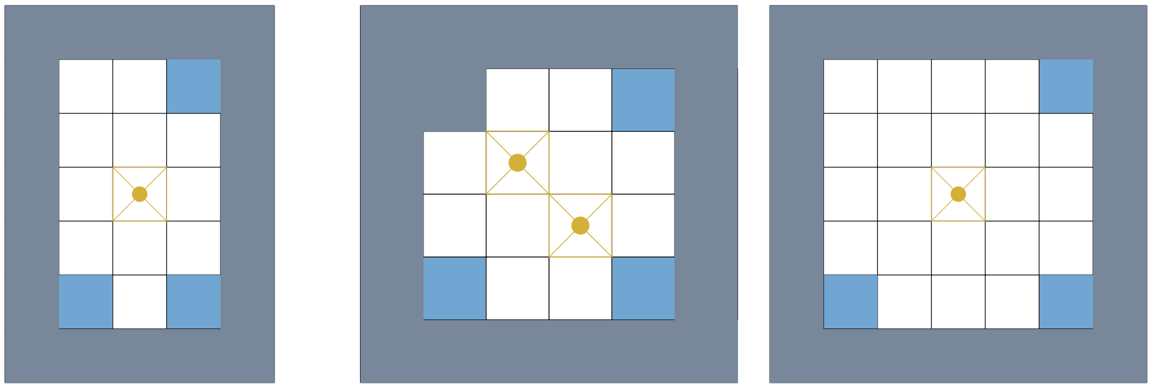
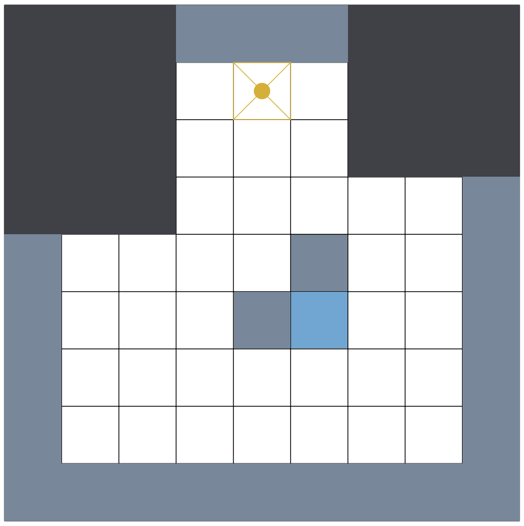
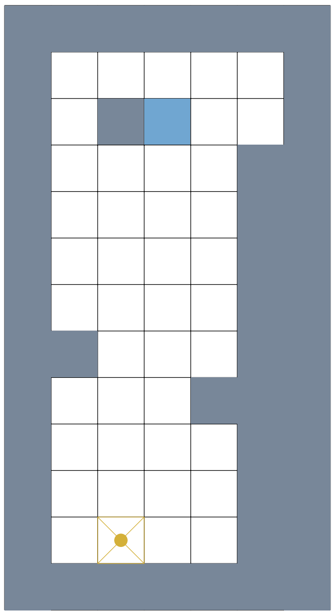

(a) Room One (b) Room Two (c) Room Three

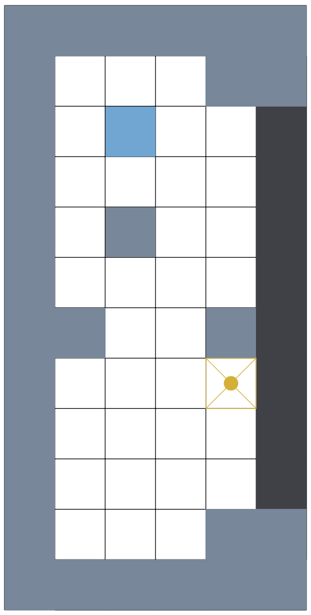
Figure 32: The three puzzles in the Ice Block Cavern

Spirit Tracks

The last game so far to feature Ice Block Puzzles is this 2009 game. All puzzles are located in the Snow Temple and are relatively simple, always featuring just one ice block and one target — a nice way to end on after the harder puzzles of Twilight Princess. The puzzles have been altered very slightly: we had to change the coordinates of some of the targets just a little, because in some of the puzzles, the targets are not adjacent to a wall, normal floor or border, and the ice blocks would fall into the target, rather than slide over them. This change does not have effect on the set of moves necessary to complete the puzzle, as the solution is the same as in the original games.


(a) Puzzle One


(b) Puzzle Two


(c) Puzzle Three

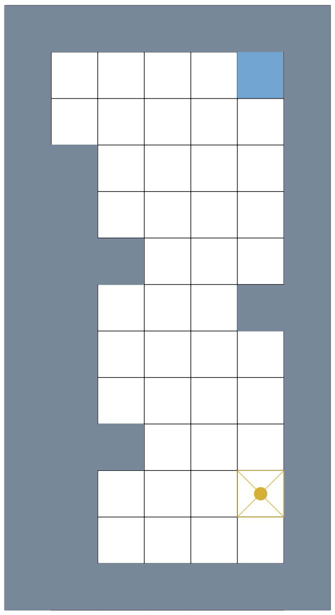

(d) Puzzle Four

Figure 33: Snow Temple puzzles