Lumines is NP-complete

Or at least if your gamepad is broken

ANDRÉ NYSTRÖM & AXEL RIESE

Stockholm April 11, 2015

Degree Project in Computer Science, DD143X
Examinator: Örjan Ekeberg

Contents

1 Introduction

In this report a simplified version of the game Lumines is shown to be NP-complete. First the rules of the game are formally defined. Using these definitions the NP-complete subset sum problem is reduced to a Lumines gameboard with a defined sequence of blocks. A proof is presented showing that the only way to clear a specific number of blocks in the given Lumines gameboard is if the subset sum instance is a "yes"-instance.

1.1 The game

The video game Lumines was released in December 2004 in Japan for the PSP platform. The game was a major success for the developer Q Entertainment, and has since its original release been followed with a couple of sequels [11].

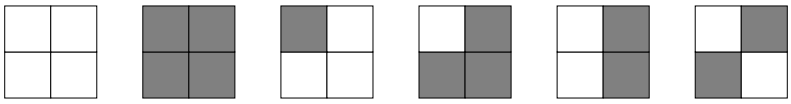


Figure 1: The six pieces of lumines: M{W,B}, L{W,B}, H and X

Lumines is a two-dimensional grid game, in which a sequence of 2×2 dichromatic *blocks* falls, and the user rotates and shifts the block before its bottom cells reaches other cells, *terrain*, in the gameboard. When the block reach existing terrain, it is no longer possible to shift or rotate the block. If both bottom cells of the block is supported by existing terrain, the block stays put on that terrain. Otherwise, the half of the block that is not supported by terrain will fall down until it is. The player is then presented with the next block.

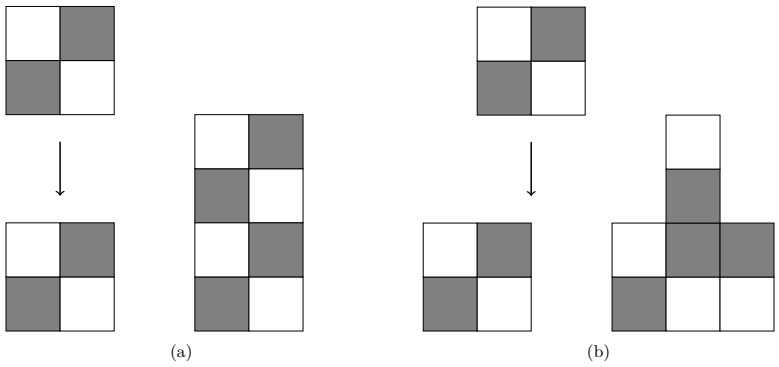


Figure 2: Block behavior with and without supporting terrain

If at any point the player forms a monochromatic 2×2 square, all cells belonging to that square is *marked*. Note that these squares may overlap, for example all cells within a monochromatic 2×3 rectangle is marked, since it consists of overlapping 2×2 monochromatic squares. At a regular pace a vertical *sweep-line* scans the gameboard and removes all marked cells.

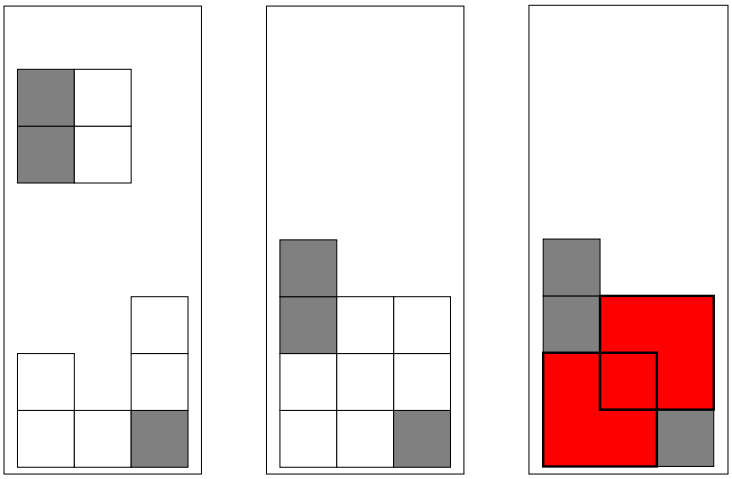


Figure 3: Marking of 2×2 monochromatic squares (from [2])

The version of Lumines found in the actual video games is played on a 10×16-gameboard. The goal of the standard game mode in Lumines is to survive, to keep placing blocks inside the gameboard, for as long as possible and clear as much terrain as possible in the process. The blocks given to the player is generated randomly, and 3-block lookahead is provided. The game starts on partially filled gameboard, which varies in size depending on the difficulty level [2].

1.2 Problem statement

This report considers the generalized version of the game, which is played on an $n \times m$-gameboard. The authors seek to formalize and simplify Lumines, and explore this new system's computational complexity characteristics. Certain goals of the game are considered by translating them into decision problems in this system.

While the sweep-line is a core characteristic of Lumines, this report considers all marked terrain as instantly cleared. Furthermore the whole sequence of blocks is considered to be known beforehand. This will be referred to as the *offline* version of the game, in contrast to the *online* version, where the blocks are generated probabilistically as in the actual video game. These are the two main simplifications, although a more thorough discussion is provided in Method. The impact of these simplifications on the validity of the results is considered in Discussion.

The main Lumines goals this paper examines are:

- Is (standard mode) Lumines NP-complete?
- Are all game modes of Lumines equally hard?

1.3 Motivation

Lumines is a game of interest because to the authors' knowledge it's not well researched from the perspective of computational complexity. Several similar games has been more thoroughly researched in recent years and comparing Lumines to these findings might yield interesting results. New findings on the computational complexity of Lumines may aid in the development of video game artificial intelligence. Our understanding of computational complexity in generic 2-dimensional grid games may also benefit from this research. Combining results from this report with results from previous similar research might make it possible to find a correlation between certain game characteristics and classes of computational complexity.

2 Background

2.1 Reduction

A reduction is an algorithm used to solve one problem with another. A reduction of a decision problem is valid if each yes-instance gives a yes-instance and each no instance gives a no instance. If the reduction is to be made in polynomial time the problem that is solved must not be harder than the problem we reduce to.[7]

2.2 NP and PSPACE

NP is the complexity class of all decision problems where a yes-instance can be verified in polynomial time. If it is possible to reduce all problems in NP to a certain problem, that problem is said to be NP-hard. A problem is referred to as a NP-complete problem if it is shown that it is both in NP and NP-hard. From this follows that if it is possible to reduce a NP-complete problem to some problem in NP using polynomial time, that problem is NP-complete as well.

PSPACE is the complexity class of all decision problems that can be solved by a Turing machine in a polynomial amount of memory space. Similar to NP-complete, a problem is PSPACE-complete if it is in PSPACE and all other problems in PSPACE can be reduced to the problem.

2.3 The Subset Sum problem

The *Subset Sum* Problem is defined as follows [10, p. 491]:

> Given natural numbers $w_1, \ldots, w_n$, and a target number W, is there a subset of $\{w_1, \ldots, w_n\}$ that adds up precisely to W?

The Subset Sum Problem plays a crucial role in this report; it is the chosen tproblem to reduce to the Lumines problem. It is known to be NP-complete [10, p. 492]. In fact it is a special case of the Knapsack Problem [10, p. 491], one of Karp's 21 NP-complete problems he discusses in his 1972 paper [8].

2.4 Previous studies

2.4.1 Classical Nintendo Games

In the paper on Classical Nintendo Games the computational complexity of games in the popular Nintendo series Legend of Zelda, Mario, Metroid, Donkey Kong and Pokémon are proven to be either NP- or PSPACE-hard [1]. The paper states that it is easy to understand that most games are members of PSPACE because their behaviour is a deterministic function of the player's controller input. The paper focus on the reachability problem in the aforementioned games, that is if it is possible to get from point A to point B on a generalized gameboard. Using the NP-complete problem 3-SAT the authors build a framework of "gadgets" to prove NP-hardness. For PSPACE-hardness a similar framework is used with the True Quantified Boolean Formula. The framework gives the authors a simple way to show NP- and PSPACE-hardness by building the gadgets as gameboards in the respective games. The authors also use previous studies on the Push-1 [3] and PushPush-1 [4] to show NP-hardness, respectively PSPACE-hardness in some of the games implementing these games as puzzles.

2.4.2 Lemmings

Lemmings is a 2D puzzle-platformer game where the objective is to guide a couple of characters called lemmings through obstacles to reach a designated exit by giving the lemmings abilities e.g.

digging or building. Much like the proof of the classical Nintendo games, the proof that Lemmings is NP-complete uses gadgets. The authors use the 3-SAT to prove NP-hardness.

2.4.3 Minesweeper

Minesweeper is a popular puzzle game shipped with the Windows operating system. The game is played on a $n \times m$ gameboard where random cells are designated to contain mines. The goal of the game is to clear all cells except the mines. When a player clears a cell and the cell is not a mine the player is presented with a digit. The digit tells the player how many adjecent cells contain mines. If there are no adjecent mines the cell is blank. The author use SAT to prove NP-hardness by building Minesweeper configurations of logic circuits e.g. NOT-, AND- and XOR-gates [9].

2.4.4 Match-three games

Candy Crush and Bejeweled are both popular puzzle games with the same concept. The game consists of grids where each cell consists one "gem". The player is allowed to swap a gems position in vertical and horizontal directions if she is able to match three of a kind. When three of a kind is matched, the gems are "popped" and the gems above the now empty cells take their place. At the same time, the empty cells at the top are filled with new gems. The authors consider the offline version off Bejeweled for their studies and show NP-hardness using 1in3PSAT. [6]

2.4.5 Tetris

Tetris is a popular puzzle game where the player is given a sequence of tetrominoes to pack into a grid-based rectangular gameboard. If a row is completely filled it is cleared and all pieces above the cleared row is dropped by one row. Using the NP-complete 3-Partition problem the offline version of Tetris has been shown to be NP-complete considering specific goals, for example maximizing the number of cleared rows. The reduction is done by building a Tetris gameboard from instances of the 3-Partition problem. A specific sequence of pieces are defined for each number in the 3-Partition instance. The authors show that for every yes-instance of 3-Partition there exists a trajectory given the sequences which clear the entire gameboard without triggering a loss. An important note is that the reduction does not use all of the tetrominoes existing in the game and therefore is slightly simplified. In the same paper, it is also shown that Tetris is highly inapproximable.

2.5 Lumines and its similarities to Tetris

The existing research on Lumines largely focus on effective strategies. One paper states that there exists strategies that can never lose, no matter what sequence of pieces are dropped [2]. This is different to Tetris where losing can be inevitable if the computer is allowed to change the sequence according to how the player organizes the pieces [5, p. 4]. Another relevant difference the brought up in prior research is that in Lumines it is possible to create terrain that can not be cleared, while in Tetris a gameboard can always be cleared if the player is given an appropriate sequence of pieces.

The standard Lumines game is played on a 10×16-gameboard, while standard Tetris is played on a 20×10-gameboard. In both games, when any block is fixed outside of the gameboard the player loses the game. While in Tetris a completed row is removed instantly, the marked cells in Lumines is not removed until the sweep-line has scanned the area.

The fact that falling blocks may seperate after being fixed in Lumines means that there aren't as many ways to build unique terrain as in Tetris. Nevertheless, some significant similarities exist between the two games making the existing research on Tetris a valuable asset when considering the complexity of Lumines.

3 Formal definitions

3.1 Lumines rules

This formalization of Lumines follows the methodology of Demaine, Hohenberger, and Liben-Nowell in their paper on Tetris [5]. Please note that the formalization corresponds to the simplified model of Lumines considered in this report, rather than the actual video game. With this in mind, in this section "Lumines" should be read as "Lumines with the constraints and modifications this report assumes".

3.1.1 Game objects

The gameboard The *gameboard* is a grid of m rows and n columns, indexed from bottom-to-top and left-to-right. The $\langle i, j \rangle$th *cell* is either *unfilled*, *black* or *white*. When a cell is either black or white, we call the cell *filled*. In a legal Lumines gameboard, no cell is unfilled if some cell above it is filled.

Game blocks In Lumines the player can only manipulate 2×2 squares of cells called *blocks*. The attributes of a block in play is defined as a *block state* P.It is 3-tuple, consisting of:

1. *block colors*, a 4-tuple $\in \{B, W\}^4$ corresponding to the corner colors of the block, counted clockwise from the lower-left corner. The six blocks core blocks pictured in figure 1 correspond to the following piece colors:

   MW (W, W, W, W)

   MB (B, B, B, B)

   LW (W, B, W, W)

   LB (B, W, B, B)

   H (W, W, B, B)

   X (W, B, W, B)

2. a *position* of the block's lower-left corner on the gameboard, chosen from $\{1, \ldots, m-1\} \times \{1, \ldots, n-1\}$.

3. the value *fixed* or *unfixed*.

In the *initial block state*, the block is in its base orientation unless noted, is always unfixed and its initial position $\langle m-1, \lfloor n/2 \rfloor \rangle$.

Rotating blocks Rotation of blocks is defined to be a transformation of the block's colors, according to the permutation $\sigma : \{B, W\}^4 \mapsto \{B, W\}^4$ corresponding to some rotation. The four possible permutations are:

1. *id*: $\sigma_{id}(\mathbf{c}) := (c_1)(c_2)(c_3)(c_4)$
2. $\frac{\pi}{2}$: $\sigma_C(\mathbf{c}) := (c_1\ c_2\ c_3\ c_4)$
3. π: $\sigma_H(\mathbf{c}) := (c_1\ c_3)(c_2\ c_4)$
4. $\frac{3\pi}{2} = -\frac{\pi}{2}$: $\sigma_{CC}(\mathbf{c}) := (c_1\ c_4\ c_3\ c_2)$.

A shorthand is defined for aliasing the blocks in some rotation. For any block color $b \in \{B, W\}^4$ we define $b_r = \sigma_r(b)$. For example the monochromatic black block rotated one quarter-turn counterclockwise can be referred to as $\mathbf{MB}_{CC}$. Please note that the purpose of this definition is to aid in the proof. In this system rotation is not an allowed block manipulation during gameplay.

3.1.2 Game operations

No operations are legal for a piece $P = (\mathbf{c}, \langle i, j \rangle, fixed)$. The following operations are legal for a piece $P = (\mathbf{c}, \langle i, j \rangle, unfixed)$, with current gameboard B:

1. A *slide to the right.* if the cells $\langle i, j+1 \rangle$ and $\langle i+1, j+1 \rangle$ are unfilled and in the bounds of the gamboard the move is legal. The new block state is $(\mathbf{c}, \langle i, j+1 \rangle, unfixed)$.
2. A *slide to the left.* if the cells $\langle i, j-1 \rangle$ and $\langle i+1, j-1 \rangle$ are unfilled and in the bounds of the gameboard, the move is legal. The new block state is $(\mathbf{c}, \langle i, j-1 \rangle, unfixed)$
3. A *drop* by one row, if the cells $\langle i-1, j \rangle$ and $\langle i-1, j+1 \rangle$ are unfilled and in the bounds of the gameboard, the move is legal. The new block state is $(\mathbf{c}, \langle i-1, j \rangle, unfixed)$.
4. A *fix.* If the cells $\langle i-1, j \rangle$ or $\langle i-1, j+1 \rangle$ are filled or $i = 1$ the move is legal. The new block state is $(\mathbf{c}, \langle i, j \rangle, fixed)$.

3.1.3 Gameplay

A *trajectory* τ of a block P is a finite sequence of (legal) game moves starting from the initial block state and ending with a fix operation. The application of a trajectory to a block P in a gameboard B renders a new gameboard B' according to the following rules:

1. B' is initially B with the cells of block P filled.
2. If the block is fixed, any column of the block which is not directly above a filled cell will continue to drop until it reaches either a filled cell or the bottom row. B' is the result of this fix operation.
3. If the block is fixed and the block position is $\langle m-1, j \rangle$, any $j \in \{1, \ldots, n-1\}$, a *game over* is triggered.
4. If any filled cell $\langle i, j \rangle$ in B' and its surrounding cells $\langle i+1, j+1 \rangle$, $\langle i, j+1 \rangle$ and $\langle i+1, j \rangle$ are the same color, these cells are marked and cleared instantly. Any filled cells in B' not directly above other filled cells or the bottom row now drops until they are. This stage is repeated until a further step would not result in any changes on the gameboard. B' is now the results of these steps.

A *game* $(B_0, P_1, \ldots, P_p)$ is defined as an initial gameboard and a sequence of blocks to be placed by the player. A *trajectory sequence* ϕ is a sequence $(B_0, \tau_1, B_1, \ldots, \tau_p, B_p)$ which for each i the trajectory τ_i applied to the game block P_i on the gameboard B_{i-1} results in gameboard B_i. If ϕ contains any trajectory τ_q which triggers a game over, ϕ naturally terminates at B_q instead of B_p.

3.2 Checkable and acyclic goals

The notion of checkable and acyclic goals is presented in [5]. These definitions will aid the proof presented later in this report. It is used in a similar manner in the original paper.

Definition 3.1. We say that an objective function Φ is *checkable* when, given a a game G and a trajectory sequence ϕ, we can compute the truth value of $\Phi(G, \phi)$ in time poly$(|G|, |\phi|)$.

Definition 3.2. We say that an objective function Φ is *acyclic* when, for all games G, if there is a trajectory sequence ϕ so that $\Phi(G, \phi)$ holds, then there is a trajectory sequence ϕ' so that $\Phi(G, \phi')$ holds and there are no repeated piece states in ϕ'.

3.3 The k-cleared-cells problem

The main Lumines goal this paper is concerned with is the problem of maximizing the amount of cleared cells given a particular game. Naturally the domain of the problem is the simplified version of Lumines previously presented in 3.1. Furthermore we add the constraint no block in play can visit a single position more than once. This effectively turns the k-cleared-cells-problem into an acyclic objective function. Since in the video game blocks drop at a fast speed, this constraint brings the problem domain closer to the actual game.

The problem is formally defined as follows:

> **The (offline, no-rotation, acyclic) k-cleared-cells problem**: Given a Lumines game $G = (B_0, P_1, \ldots, P_p)$ and a goal $k \in \mathbb{N}$, does there exist an acyclic trajectory sequence $\phi = (B_0, \tau_1, B_1, \ldots, \tau_p, B_p)$, such that when ϕ is applied to G at least k cells are cleared in total?

4 Method

The methodology used is standard procedure in the domain of computational complexity. A brief review of this terminology is provided in section 2. The process is as follows:

1. It is proven that given a solution of a k-cleared-cells instance, the solution is verifiable in polynomial time. By definition this will prove that k-cleared-cells is in NP.
2. A previously known NP-complete problem is reduced to k-cleared-cells. Since a reduced problem can be no harder (in a computational sense) than the problem it is reduced to, it follows that k-cleared-cells is NP-hard.
3. If k-cleared-cells is in NP and is NP-hard, it follows from definition that the k-cleared-cells is NP-complete.

4.1 Finding a reduction

As mentioned in subsection 2.5 there exists many similarities between Lumines and the more thoroughly researched video game Tetris. This fact suggested that is was a good idea to draw inspiration from papers concerning the computational complexity of Tetris. In one such paper by Demaine, Hohenberger, and Liben-Nowell, the authors present a reduction from *3-Partition* to the problem of how many rows can be cleared using some sequence of Tetris pieces [5]. Although a proof of similar completeness is considered to be out of the scope of this report, the notion of forcing game pieces to be placed in certain positions by creating constraints has showed valuable for the proof presented in this report.

NP-hard problems priorly encountered in literature and courses were first examined. Problems were considered on how good they could they could relate to spatial characteristics of Lumines. Finally the *Subset sum* problem was chosen as a potential problem to reduce from, on the grounds that it was well understood by the authors and that mappings could be made intuitively from the problem to some characteristics of Lumines. For example it seemed plausible that a set of elements in a *Subset sum* instance could be mapped to a set of Lumines blocks of the same kind in a sequence. The characteristic of summing elements in a *Subset sum* instance also seemed like a good match for the stacking characteristic of Lumines gameplay.

5 Result

5.1 On the size of the problem input

The input to a particular instance of the k-cleared-cells problem is the game $G = (B_0, P_1, \ldots, P_p)$. In order to analyze the verifiability of a given solution in polynomial time, we must begin by examining the size $|G|$. The game contains on gameboard B_0 of size $|B| = mn$, and p block states which are all of constant size. Hence of the size of all the blocks state are $O(p)$. Thus $|G| \in O(p \cdot |B|)$.

5.2 On the size of the problem solution

The solution to a particular instance of the k-cleared-cells problem is the trajectory sequence $\phi = (B_0, \tau_1, B_1, \ldots, \tau_p, B_p)$. Since definition 3.1 depends both on the size of $|G|$ and $|\phi|$, examining the size of $|\phi|$ is of importance. ϕ consists of p gameboards of size $|B| = mn$, and p trajectories of unknown but finite size. We therefore define C to be the upper bound of the amounts of operations in any trajectory in ϕ. Thus $|\phi| \in O(p \cdot |B| \cdot C)$.

5.3 Is k-cleared-cells in NP?

We begin by proving the following useful lemma:

Lemma 5.1. *The legality of a trajectory sequence $\phi = (B_0, \tau_1, B_1, \ldots, \tau_p, B_p)$ in any game $G = (B_0, P_1, \ldots, P_p)$ is a checkable objective function.*

Proof. For every trajectory τ in ϕ, deciding whether any operation in τ is legal is a check in constant time according to the rules in 3.1.2.

After a fix operation has been made, the gameboard enters a loop and changes according to the clear semantics defined in 3.1.3. This loop can run at most $|B|/4$ times, since at least 4 cells has to be cleared each iteration, and no change can occur if there aren't any cells left to clear. Each iteration requires checking at most $4|B|$ cells for monochromatic 2×2-squares to mark. Thus the time to compute the new gameboard for each τ in ϕ is $\in O(|B|^2)$.

Checking that applying τ_i to P_i in B_{i-1} yields B_i requires $|B|$ comparisons.

In total p new gameboards has to be generated and compared. Let the maximum amount of operations in any trajectory τ in ϕ be bounded by $C \in \mathbb{N}$. Thus we have that checking the validity of ϕ in G can be done in $O(p \cdot C \cdot |B|^3) = \text{poly}(|G|, |\phi|)$. □

We proceed by proving this modified theorem from [5]:

Theorem 5.2. *For any checkable acyclic objective Φ in Lumines, $\Phi \in NP$.*

Proof. We are given a Lumines game $G = (B_0, P_1, \ldots, P_p)$ and a acyclic trajectory sequence $\phi = (B_0, \tau_1, B_1, \ldots, \tau_p, B_p)$. Let B_i be $m \times n$ gameboard.

Since ϕ is acyclic, each of its p trajectories can only contain at most $(m-1)(n-1)+1 \in O(|B|)$ states, unfixed once in each possible position and fixed in one final position. Thus $|\phi| \in O(p \cdot |B|) \subset O(|G|)$. This implies $\text{poly}(|G|, |\phi|) = \text{poly}(|G|)$.

Checking the validity of the ϕ can be done in $\text{poly}(|G|, |\phi|) = \text{poly}(|G|)$ according to lemma 5.1.

Since Φ is checkable, we can then in time $\text{poly}(|G|, |\phi|) = \text{poly}(|G|)$ verify if $\Phi(G, \phi)$ holds. Thus $\Phi \in \text{NP}$. □

Now we just have to prove the following lemma, and the original question will be answered as a direct result:

Lemma 5.3. *The objective k-cleared-cells is checkable and acyclic.*

Proof. The objective is acyclic because it only depends on fixed block state at the end of each trajectory, since the fix operation is the only way to trigger clearing of cells and thereby changes to the gameboard. The path taken in the trajectory is therefore irrelevant, as long as its legal.

One can count the total amount of cleared cells simply by scanning the gameboard after each fix loop iteration, and then summing the results of each iteration. As presented in the proof for lemma 5.1, this loop runs $O(|B|^2)$ times, and the counting of cleared cells can be done in $O(|B|)$. In total the summing can therefore be done in $O(|B|^3 \cdot |\phi|) = \text{poly}(|G|, |\phi|)$ time. Thus the objective is checkable. □

This naturally leads us to the following corollary:

Corollary 5.4. *The k-cleared-cells problem is in NP.*

5.4 Is k-cleared-cells NP-hard?

We prove that k-cleared-cells is NP-hard by constructing a reduction from the *Subset sum* problem. The reduction consists of two parts: one part which considers $K > 1$ in the given *Subset sum* instance, and one part which considers the special case $K = 1$. The first part is more complex and consists of five phases, whereas the second part is relatively straightforward.

The goal of each part is to prove the following:

> Given a *Subset sum* instance $Q = \{q_1, \ldots, q_a\}, K \in \{1\}, \{2, 3, \ldots\}$, there exists a solution $S = \{s_1, \ldots, s_b\} \subseteq Q, \sum Q = K$ if and only if there exists a trajectory sequence ϕ which clears k cells in game G in some instance of *k-cleared-cells*.

5.4.1 The initial gameboard

The initial gameboard B_0 in the constructed *k-cleared-cells* instance is pictured in Figure 4. It consists of two identical structures which will from here on be referred to as "wells". From the figure we can deduce that B is a $2\left(\sum P + K + 1\right) \times 10$-sized gameboard.

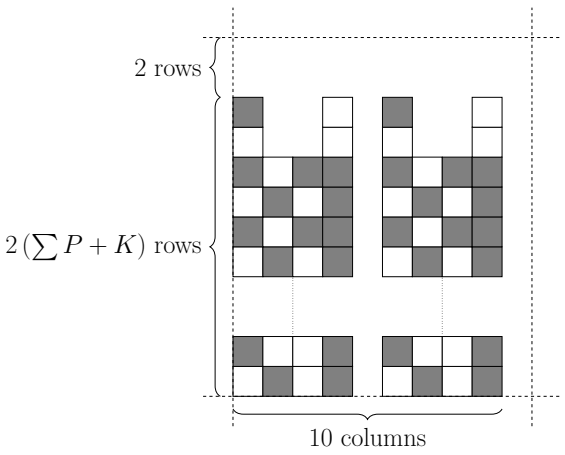


Figure 4: The initial gameboard

A schematic of the well structure is pictured in Figure 7. Column 5 of the well is empty, the content in the other columns depends on which section the rows are in. A well is divided into the following three sections:

- The bottom section consists of rows in the interval $[1, 2\,(K+1)]$. In this section, column 1 and 2 is checkered, column 3 consists of white cells, and column 4 consists of black cells.
- The middle section consists of rows in the interval $[2\,(K+1)+1, 2\,(K-1+\sum P)]$. In this section, column 1 to 3 is checkered, and column 4 consists of black cells.
- The top section consists of the rows in the interval $[2\,(K-1+\sum P)+1, 2\,(K+\sum P)]$. In this section columns 2 and 3 are empty. Column 1 consists of white cell in the bottom row, and one white cell in the top row. Column 4 consists of two white cells.

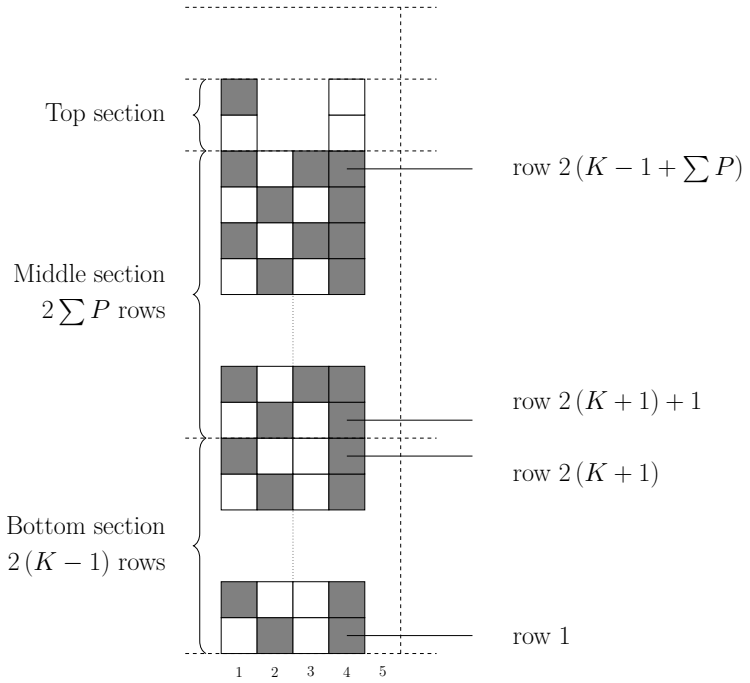
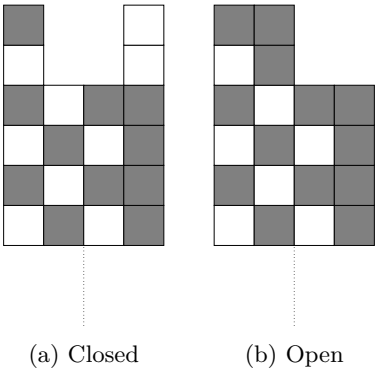


Figure 5: The well structure

The purpose of the well is to force the placement of blocks in play. One feature of the well structure is that if we were to drop a block with 2 black cells on the right half in column 4-5, the black right half would drop down and instantly clear itself and the two bottommost cells in column 4. The terrain in column 4 would then collapse, making place for a new block to be placed in the same manner.

This feature cannot be utilized in the default well state though, since fixing a block on the top of column 5 would result in a game over. Would the two right white cells in the top section disappear, this feature becomes available. This characteristic of the well is heavily used in the reduction. We therefore define two states of the well. If the well is *closed*, placing a block in column 4-5 yields a game over, if the well is *open* this placement does not yield a game over. The appearance of the top section corresponding to the two well states is pictured in Figure 6. The reason for this will later become apparent.



(a) Closed (b) Open

Figure 6: Open and closed wells

5.4.2 Phase 1

In this phase we transform every $q_i \in Q$ to sequences of game blocks. These sequences will be part of the block states $P_1, \ldots P_n$ in the constructed *k-cleared-cells* instance. The transformation is done by creating a sequence of 1 $\mathbf{H}_H$, q_i $\mathbf{H}$ and 1 $\mathbf{H}_H$ blocks (in their initial state) for each give q_i.

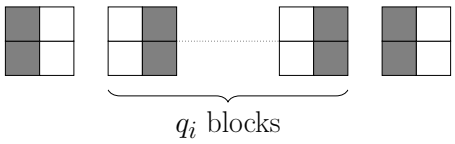


Figure 7: Transformation of *Subset sum* element q_i

5.5 Conclusion

6 Discussion

6.1 Rotations

The two main ways the player can control the game is to rotate and move the blocks

6.2 Sweep-line

In Lumines the sweep-line moves in a constant speed from left to right. The marked squares are cleared when the sweep-line reach the right side of the gameboard. The simplified Lumines instance that is presented in this paper uses an instant sweep-line. This means the blocks are cleared when a new block is fixed which is similar to how rows are cleared in Tetris but not the most true representation of how the sweep-line works in Lumines. There is probably a lot of different ways to make better models of the real sweep-line. For example, we could clear on every nth block that is fixed to imitate a moving sweep-line where n is an appropriate integer. The instant model was chosen because it was easier to find and prove a reduction in the time given for the project. In future work we would like to inquire into what impact the sweep-line has on the computational complexity and if there would be any significant results using a better model.

6.3 Online Lumines

In the reduction and proof of Lumines complexity presented in this paper the offline version of Lumines has been considered. This is similar to previous studies on Match-three games and Tetris. In these reports they argue that if a property can be shown in the offline version of the games intuitively it should not be easier in the online version. According to Demaine, Hoffman and Holzer, "...it is only easier to play Tetris with complete knowledge of the future, so the difficulty of playing the offline version suggests the difficulty of playing the online version" [5, p. 2]. This should also be the case for Lumines although we do not have a proof that this is the case. Therefore the online version of Lumines should be examined in the future to get a better understanding of the real game's complexity.

References

[1] Greg Aloupis et al. "Classic Nintendo games are (NP-) hard". In: *arXiv preprint arXiv:1203.1895* (2012).

[2] Greg Aloupis et al. "LUMINES strategies". In: *Computers and Games* (2007), pp. 190–199.

[3] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. "PushPush and Push-1 are NP-hard in 2D". In: *Proceedings of the 12th Annual Canadian Conference on Computational Geometry*. 2000, pp. 211–219.

[4] Erik D. Demaine, Michael Hoffman, and Markus Holzer. "PushPush-k is PSPACE-Complete". In: *Proceedings of the 3rd International Conference on Fun with Algorithms*. May 2004, pp. 159–170.

[5] Erik D Demaine, Susan Hohenberger, and David Liben-Nowell. "Tetris is hard, even to approximate". In: *Computing and Combinatorics*. Springer, 2003, pp. 351–363.

[6] Luciano Gualà, Stefano Leucci, and Emanuele Natale. "Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard". In: *CoRR* abs/1403.5830 (2014). URL: `http://arxiv.org/abs/1403.5830`.

[7] Viggo Kann. *Formella definitioner*. 2015. URL: `http://www.csc.kth.se/utbildning/kth/kurser/DD1352/adk13/schema/F22.pdf`.

[8] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[9] Richard Kaye. "Minesweeper is NP-complete". In: *The Mathematical Intelligencer* 22.2 (2000), pp. 9–15.

[10] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley, 2005.

[11] Wikipedia. *Lumines — Wikipedia, The Free Encyclopedia*. 2015. URL: `http://en.wikipedia.org/w/index.php?title=Lumines&oldid=641202548` (visited on 03/04/2015).