Universiteit Leiden
The Netherlands

Opleiding Informatica

Bomb-r-cover

Expanding the difficulties of Bomberman

Rogier van den Burgh

Supervisors:
Rudy van Vliet & Hendrik Jan Hoogeboom

BACHELOR THESIS

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

31/08/2021

Abstract

Bomberman can be considered a classic in the gaming community. The main character, Bomberman, armed with bombs is tasked to eliminate the enemy, be that an AI or another player. This thesis looks into a smaller problem where we are only interested in the crates used in the game. Given a Bomberman board and a number of bombs with limited range, are you able to destroy all the crates on the board?
In the work of R. van de Vlasakker, Bomb-cover [Vla19], the problem is shown to be NP-complete when the range of the bombs is infinite. This thesis shows that the problem is also NP-complete for any finite range bigger than 0. This is achieved by creating a reduction from 3-SAT to Bomb-r-cover. The Bomberman board is constructed using several components that connect to each other, forming wires that lead to special clause components. In particular this thesis introduces a component that allows two wires to intersect without risking that the boolean value of the wires switches.

Contents

1 Introduction

Bomberman is a game series which was first released for the NES in the 1985. In the game you play a Bomberman able to place bombs where it stands. There are two main game modes, the single player variant where you are tasked to destroy a number of enemies and a multiplayer variant where the goal is to destroy the other player(s). The bombs with a fixed range are able to destroy crates, players and enemies. In many versions of the game Bomberman also has access to power-ups. These upgrades vary from increasing the range of bombs to making Bomberman temporary invincible. This thesis builds upon the work of R. van de Vlasakker, Bomb-cover [Vla19]. Here we have a single player game of Bomberman where the goal is to destroy all the crates on the field. Bomberman has a given number of bombs with a limited range. The question is, whether or not this number of bombs is sufficient to destroy all the crates on the board. This thesis will show that this problem is NP-complete for any $r \geq 1$ using a reduction from 3-SAT.

The thesis will start by setting the terminology used throughout the thesis in Section 2. Next in Section 3 we discuss some related work. With the ground work set, we continue by expanding on the problem in Section 4. Here the idea of the reduction is explained as well as the crossover gadget from [Lic82]. This reduction will use several different components. In Section 5 the most important rules that the components must adhere are explained. Subsequently, Section 6 will provide rules to ensure that you are able to connect different components, such that the behaviour is as expected and that the board still satisfies the rules from Section 5. Section 7 will describe several components that will be used for the board. In Section 8 these components are used to create the final component used in the reduction, an intersection component. With all the components and rules described Section 9 will describe the construction of the Bomberman board and finish the proof that our problem is NP-complete. Finally we draw the conclusions from our work, and discuss several unsolved problems in Section 10.

2 Definitions

In this thesis we consider a simplified version of Bomberman. This version of Bomberman consists of a board with **empty squares**, **walls**, **crates** and a **player** as can be seen in Figure 1. The **player** has access to a limited number of bombs. A bomb creates an explosion that stretches in horizontal and vertical direction similar to a plus sign. This explosion has a range of r going from 1 to infinity. During an explosion the player is removed from the board and he returns to the board when the explosion is finished. This is to ensure that the player is not affected by the explosion unlike in other versions of Bomberman.

The player can walk on any empty squares as shown in Figure 1a. For this reason the player begins on an empty square. Explosions are unaffected by this and can thus expand over these squares.

A wall as can be seen in Figure 1b is a solid space. As a consequence walking on these squares is not possible for the player. In fact walls can be used to restrict the movement of the player. This also counts for an explosion. It is not able to pass through these squares. When an arm of an explosion

(a) **empty squares** (b) **wall**

(c) **crate** (d) **player**

Figure 1: Different squares on a Bomberman board.

hits a wall, this arm stops stretching from this point on. This will be important if you want to ensure that the explosion has a limited effective range even if the range of the bombs themselves is infinite.

Finally the board has crates as depicted in 1c. Like walls they are solid so the player cannot pass them and neither can an explosion. The difference between crates and walls is what happens after an explosion. If a crate is hit by an explosion it will be destroyed when the explosion subdues and the square where the crate was becomes an empty square.

Crates can be **directly reachable**, **reachable** or **unreachable**. If a crate is directly reachable then there exists a path from the player to the crate where the path only goes over empty squares. When a crate is reachable then there exists a path from the player to the crate but the path goes over empty squares as well as over crates. These crates will have to be destroyed to create the path only consisting of empty squares. For crates that are unreachable such a path cannot be formed, in other words walls are blocking the crate for the player.

Our goal is to destroy all reachable crates with a given number of bombs. This is more difficult than it seems because of the many choices the player can make. The example where the bombs have infinite range was already proven to be NP-complete in [Vla19]. That thesis considered two versions of the problem, called **Bomb-cover** and **Bomb-r-cover**. For Bomb-cover the bombs have infinite range and in Bomb-r-cover the bombs have a limited range r. In our thesis we do not care about the range of the bombs, for this reason we will call the combined problem Bomb-r-cover. In [Vla19] the problem is also approached by the greedy technique as it seems a very natural solution to the problem. In this approach the player tries to destroy as many crates as possible with each explosion. Note that range matters here as choices made may be different for different ranges. In the thesis an example was shown where the greedy approach does not give the minimum number of bombs, thus an instance with this number would yield `false`. This example can be seen in Figure 2.

To find out if our given number of bombs is enough, one could try every possibility where the bomb destroys at least one crate. This brute force approach has a worst case complexity of $O(c!)$ where c is the number of crates. To see this, consider a case where you can only destroy one crate at a time, so you have to destroy any crate followed by any of the remainders until all the crates are destroyed and repeat this for every crate. For this reason we speculated that Bomb-r-cover may be NP-complete, even for limited range. According to Garey & Johnson [GJ79] there are several steps in proving whether a problem is NP-complete. This boils down to two main aspects:

1. The problem is in NP, that is, the problem can be solved by a nondeterministic polynomial

Figure 2: A counterexample for a greedy algorithm as used in [Vla19].

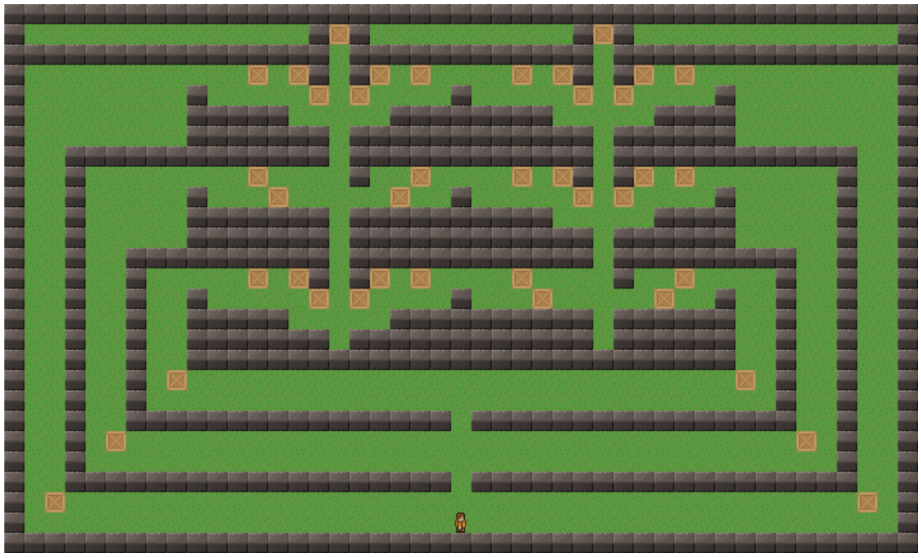


Figure 3: The board for the 3-SAT formula $(x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z)$ as used in [Vla19]

time algorithm.

2. The problem is NP-hard, that is, there exists a reduction from an NP-complete problem to our problem in polynomial time.

In [Vla19] it is already proven that Bomb-r-cover is in NP. All that is left to be done now is to prove that Bomb-r-cover is NP-hard which is the main topic of this thesis.

3 Related Work

Our thesis, as mentioned before, is a continuation of the work by Van de Vlasakker [Vla19]. That thesis deals with the Bomb-cover problem, which we mentioned before, in detail. In this problem the bombs have infinite range. Bomb-cover was proven to be NP-complete using a reduction from 3-SAT. This was accomplished by using wires and clause crates as shown in Figure 3. These wires are the loops that you can see on the board. They represent the variables of the 3-SAT formula. You are meant to "follow" the wires by always destroying 2 crates at the time. There are two ways to divide the crates in a wire into pairs that can be destroyed together. The way that is chosen, is called the parity of the wire. It corresponds to the truth value (true of false) of the variable. At the

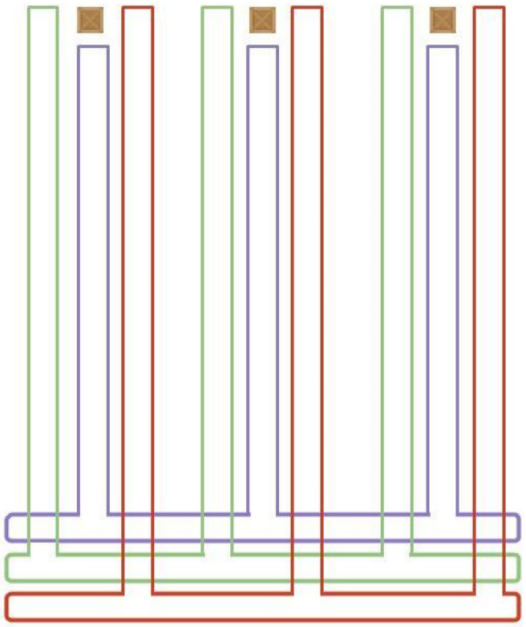
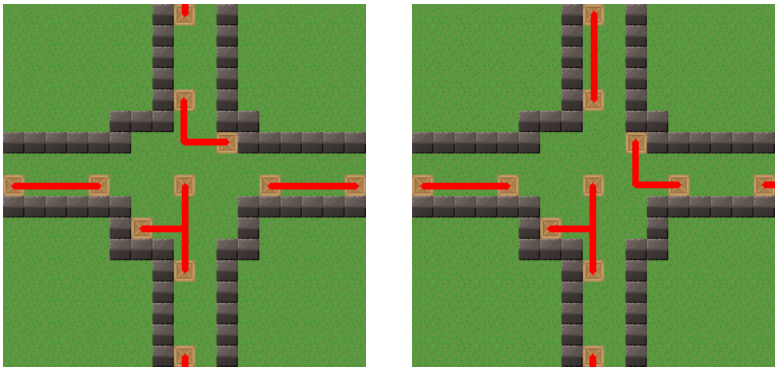


Figure 4: Schematic of a suggestion for a reduction from 3-SAT to Bomb-r-cover, the crates represent clauses and the wires represent the variables

top of the board you can find crates which represent the clauses of the formula. Every variable is connected to the clause crates via fragments which are different for variables x that simply occur as x in the clause, variables x that occur as $\neg x$ in the clause and variables that are not present.

If the parity of the wire is equal to the occurrence of x in this clause, the clause crate can be destroyed without using an additional bomb. You are able to destroy three crates instead of two so the crate is effectively free. When the parity does not match with the occurrence of x then you do not get the clause crate for free. If the entire clause is false then the clause crate on top of the board is still present and has to be destroyed with an additional bomb to clear the board. However there are enough bombs to destroy the crates in the wires, two at the time, but there are no extra bombs for the clause crates that are left. For this reason the answer to the decision problem is true if and only if the 3-SAT problem that is represented can be evaluated true. Thus this is a successful (polynomial) reduction from 3-SAT to bomb-cover. This reduction and the described nondeterministic polynomial algorithm imply that Bomb-cover is NP-complete.

Van de Vlasakker [Vla19] attempted to modify this reduction to prove that the decision problem Bomb-r-cover (finite range) is NP-complete, but he encountered a problem. In the reduction to Bomb-cover the clause crate is on top of the wires and the explosion should go from the wire (variable) that satisfies the clause all the way to the clause crate. This single shaft works with infinite range, but not with limited range. For this reason the wires should be led towards the clause crates bringing the explosion in range with the clause crate. This idea would work except that it might produce many intersections between wires, see Figure 4. As depicted in Figure 5a and Figure 5b the proposed intersection has multiple solutions. You might be able to switch the parity of the wire yielding the possibility to make false wires true and vice versa. Using this you



(a) **Possibility 1** (b) **Possibility 2**

Figure 5: Intersections that may not work as used in [Vla19]

can destroy clause crates that should not be destroyed according to the 3-SAT formula. As a result, a no instance of 3-SAT might be transformed into a yes-instance of Bomb-r-cover. For this reason this thesis will solve this problem by implementing the intersections with a construction inspired by a so-called crossover gadget, which was used in [Lic82] to prove that Planar 3-SAT is NP-complete.

4 Graph representation

In this thesis we will create a reduction from 3-SAT to Bomb-r-cover. Our Bomberman board uses wires to represent the variables of the 3-SAT formula and clause crates that represent the clauses similar to [Vla19]. When a variable is used by the clause, the wire corresponding to the variable is extended towards the respective clause crate, again as suggested in [Vla19]. But in this approach we still encounter intersections.
Our approach is to first find a graph representation of our 3-SAT formula. If we can ensure that this graph is planar, it can be represented by a Bomberman board which does not have intersections. In this graph the nodes will represent the variables and the clauses of the 3-SAT formula. Then for every variable that is used in a clause there is an edge connecting them see Figure 6. As said, our goal is to create a planar graph from this graph, which represents an equivalent 3-SAT formula. However, finding an equivalent planar graph for our general graph is often complex. If this planar graph becomes too complex, the space required to fit all the wires might create unexpected problem when creating the Bomberman board.

Planar Formulae

Lichtenstein found a solution to our problem in his work "Planar Formulae and their uses" [Lic82]. Here Lichtenstein describes a subgraph to replace an intersection as shown in Figure 7. This subgraph can be seen in Figure 8. Here the variables are represented by labeled nodes and the clauses by unlabeled nodes. Note however that there is one exception to this, ξ. This is a clause but unlike any other clause it combines four variables instead of three or less. In our Bomberman board we can still use this as if it were a four variable clause, but in other cases one might want to use a splitting method to create two clauses of three variables. The clauses occurring in 8 are

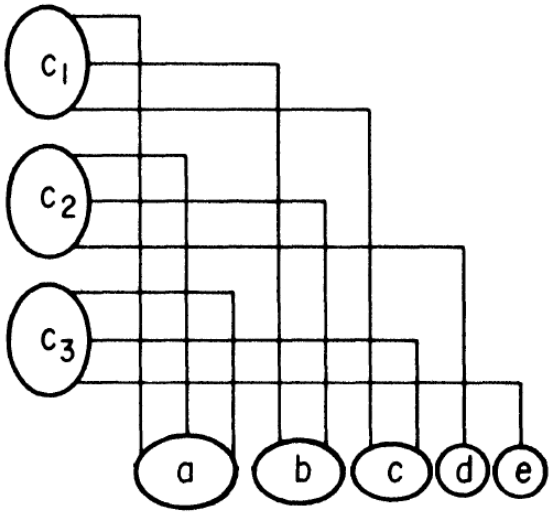


Figure 6: Example of a graph representing a 3-SAT formula based on [Lic82]

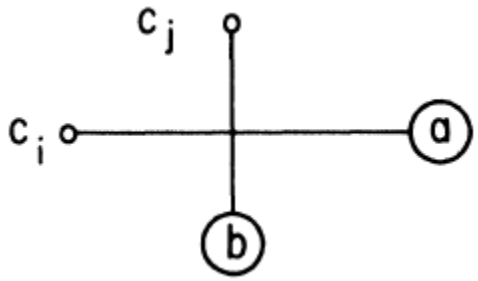


Figure 7: An example of a graph intersection from variables a an b as used in [Lic82]

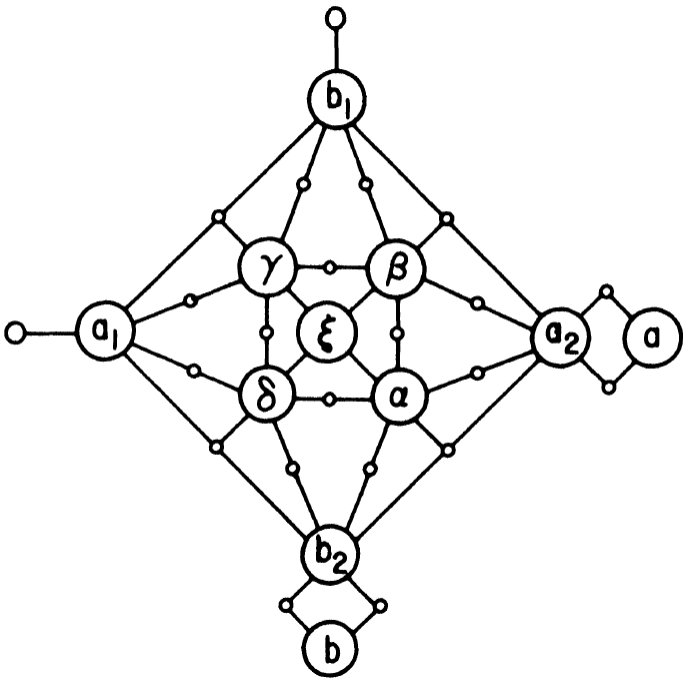


Figure 8: The subgraph used to solve intersections as used in [Lic82]

$(\neg a_2 \vee \neg b_2 \vee \alpha) \wedge (a_2 \vee \neg\alpha) \wedge (b_2 \vee \neg\alpha)$,	$i.e., a_2 \wedge b_2 \Leftrightarrow \alpha$;
$(\neg a_2 \vee b_1 \vee \beta) \wedge (a_2 \vee \neg\beta) \wedge (\neg b_1 \vee \neg\beta)$,	$i.e., a_2 \wedge \neg b_1 \Leftrightarrow \beta$;
$(a_1 \vee b_1 \vee \gamma) \wedge (\neg a_1 \vee \neg\gamma) \wedge (\neg b_1 \vee \neg\gamma)$,	$i.e., \neg a_1 \wedge \neg b_1 \Leftrightarrow \gamma$;
$(a_1 \vee \neg b_2 \vee \delta) \wedge (\neg a_1 \vee \neg\delta) \wedge (b_2 \vee \neg\delta)$,	$i.e., \neg a_1 \wedge b_2 \Leftrightarrow \delta$;
$(\alpha \vee \beta \vee \gamma \vee \delta)$;	
$(\neg\alpha \vee \neg\beta) \wedge (\neg\beta \vee \neg\gamma) \wedge (\neg\gamma \vee \neg\delta) \wedge (\neg\delta \vee \neg\alpha)$;	
$(a_2 \vee \neg a) \wedge (a \vee \neg a_2) \wedge (b_2 \vee \neg b) \wedge (b \vee \neg b_2)$,	$i.e., a \Leftrightarrow a_2, b \Leftrightarrow b_2$;

Table 1: The clauses corresponding to the unlabeled nodes in Figure 8, note that the ξ corresponds to the four variable clause

explained in [Lic82] and can be seen in Table 1.

This introduces additional variables to the graph and thus the 3-SAT formula, but they do not effect the overall satisfiability. If this subgraph is introduced for every intersection, with its own respective variables, the graph becomes planar. In this thesis the Bomberman representation of this subgraph will be used to solve the intersections we may encounter in our reduction.

5 Rules

To create a correct reduction from 3-SAT to Bomb-r-cover we want to create an equivalent Bomb-r-cover instance for any 3-SAT formula. We need to establish what our wires, clauses and intersections look like. To facilitate building a suitable Bomberman board we will establish some rules and create

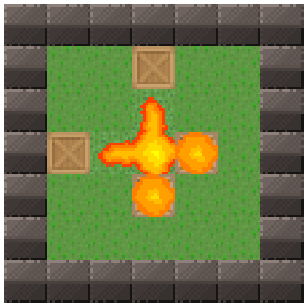


Figure 9: Explosion with range 1

components based on these rules. With these components any 3-SAT formula can be represented as a Bomb-r-cover instance. In this thesis we will create components that can be combined to achieve this reduction for any explosion range r. Creating a component for a fixed, finite range is less complex than creating one for any range, as this limitation allows crates to be out of range. Our reduction also works with infinite range. For this reason we will establish some rules that should make sure that our component aids toward our reduction for any range r. In this section every rule will be described followed by the motivation. This motivation will explain why the rule is important for a component and how a component breaks if it does not satisfy this rule.

1. A wire is a loop of crates in which you can destroy all n crates using at least $n/2$ bombs. This minimum of n/2 can only be achieved by destroying the crates in pairs of two. The clause crates are not considered part of the wire.
   Rule:
   For any pair of crates that should be destroyed together, the Manhattan distance between the two crates should be 2. In other words the target crates are separated by a single empty square.
   Motivation:
   The minimum range for our explosion is 1 as stated in section 2. This implies that the crates to be destroyed should directly surround the player. When a crate is not adjacent to the player it is not destroyed by an explosion of range 1 as shown in Figure 9. If two crates are adjacent to the player then the distance from one crate to another crate is 2, take a step from the crate to the player and then to the other crate. Here it is important that you need an empty square between the crate for the player to stand on.

2. **Rule:**
   For most crates on the board the coordinates keep the same remainder of division by 2, so even coordinates stay even and odd coordinates stay odd. All crates at the edges of the component should at least comply to this rule. Those inside may change, but that does not occur frequently.
   Motivation:
   From the first rule we know that by placing crates in a straight line they change either two squares in horizontal or vertical direction and the other coordinate stays the same as can be seen in Figure 10. Only when you make a tight corner, the crates are diagonally adjacent as can be seen in Figure 11, both coordinates change their remainder, so even becomes odd and vice versa. To ensure that different components can be combined the edges should have the

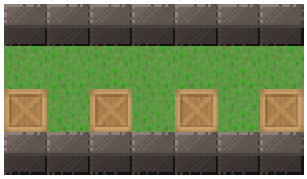


Figure 10: An example of an straight path

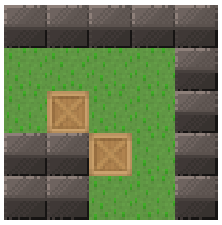


Figure 11: An example of a tight corner

same remainder, this will be explained in more detail in Section 6.

3. **Rule:**
   From any empty square you may only hit up to two crates or up to three including a clause crate even if you have infinite range with the explosion.
   Motivation:
   This rule is important to ensure that you can only break the crates in the intended way. If you could hit three or four crates you might be able to save a bomb till the end which allows you to destroy a clause crate that should not be broken. It is still possible to break zero or one crate with a bomb, but then you will not be able to remove all the crates with the provided number of bombs. If you include a clause crate you should destroy three crates. This implies that the clause crate is "free", which should be the case if and only if the clause is true.

4. **Rule:**
   Building on from rule 3, if you bomb multiple times on some reachable location you should destroy at most two crates in total or three if it includes a clause crate even if you have infinite range with the explosion.
   Motivation:
   If you are able to break more than two crates by repeated explosions on the same position then you have either of two situations. Either the crates are placed in a straight line as shown in Fig 12a or the crates form a corner as depicted in Fig 12b.

   (a) In Figure 13, we see a fragment of a wire (corresponding to a variable), consisting of a straight line of crates with a clause crate in the middle. In this example the variable does not make the clause true. If the wire is followed in the intended manner the clause crate is not destroyed. The clause crate can only be destroyed in the intended manner if this variable make the clause true.

   In Figure 13a the player places the bombs at the intended positions with each bomb destroying the next two crates. Note that in this case the clause crate is not destroyed

(a) An example where you find three crates in a straight line

(b) An example where you find three crates in a curve

Figure 12: Examples of three crates that can be destroyed from the same position

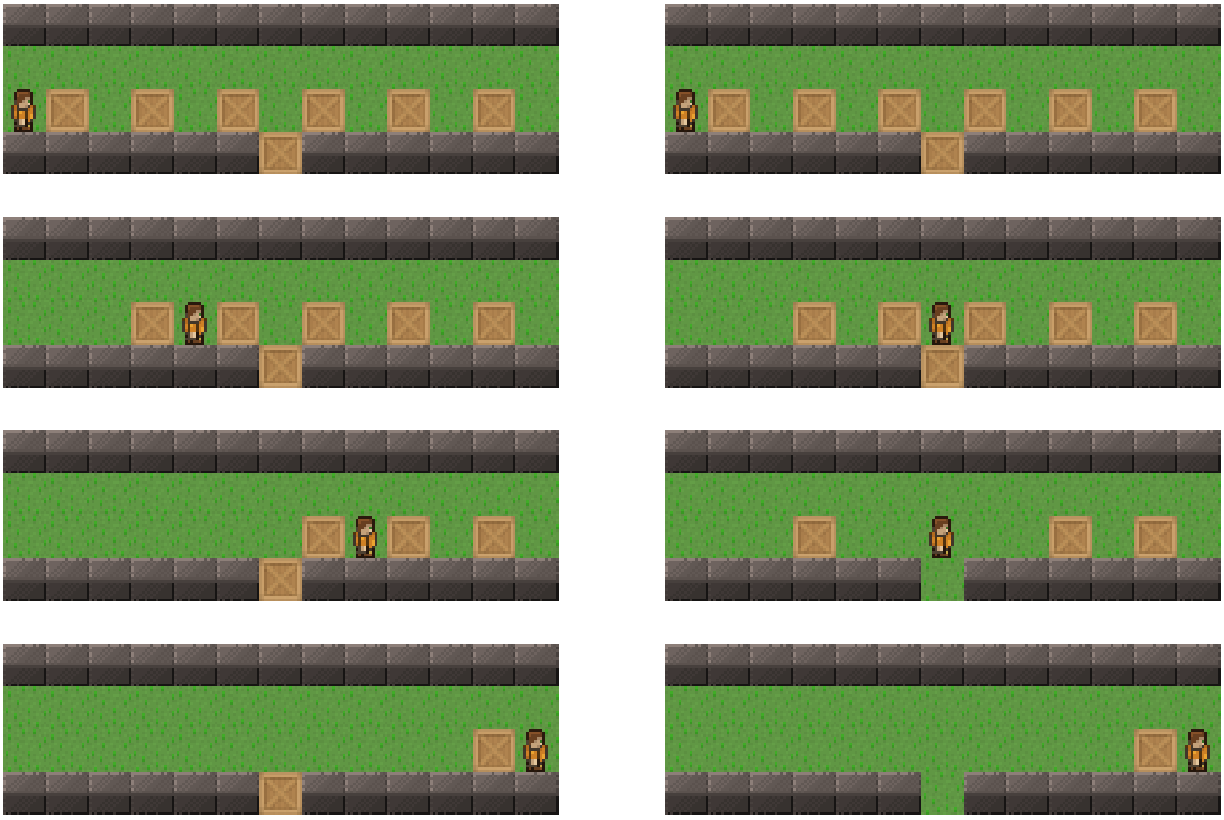
because as said before the variable does not make the clause true.
However if you look at Figure 13b you can see that the end result for the wire is the same, but here the clause crate has been destroyed. In fact we have temporarily switched the parity for the wire. This problem occurs only if the range of the bombs is 3 or higher. If the range was 1 or 2 we could not reach the outer crates from the position above the clause crate.

(b) The situation seen in Figure 14 is similar to the previous problem. Here we should not make the clause true but if the player has a bomb with a bigger range he is able to destroy the clause crate and all the crates with the same number of bombs. This implies that you are able to destroy the clause crate for free while this should not be possible.

The previous two rules may not be as intuitive as the first two. However they exists to ensure the following property:
Property:
Each crate in the wire can be destroyed in a pair in exactly two ways, with two different crates.
Motivation:
In order to enforce that you follow the wires you need to ensure that there are exactly two ways you can follow this wire. As said before, these two ways correspond to evaluating the wire (which corresponds to a variable) as true or false. We want to prevent escaping from the wire at random moments or otherwise finding a way to evaluate the wire in an unexpected way. We do not present this property as a separate rule, because it is more of an effect from all the other rules. It is, however, worth to be mentioned, as it summarizes some core properties.

6 Connecting components

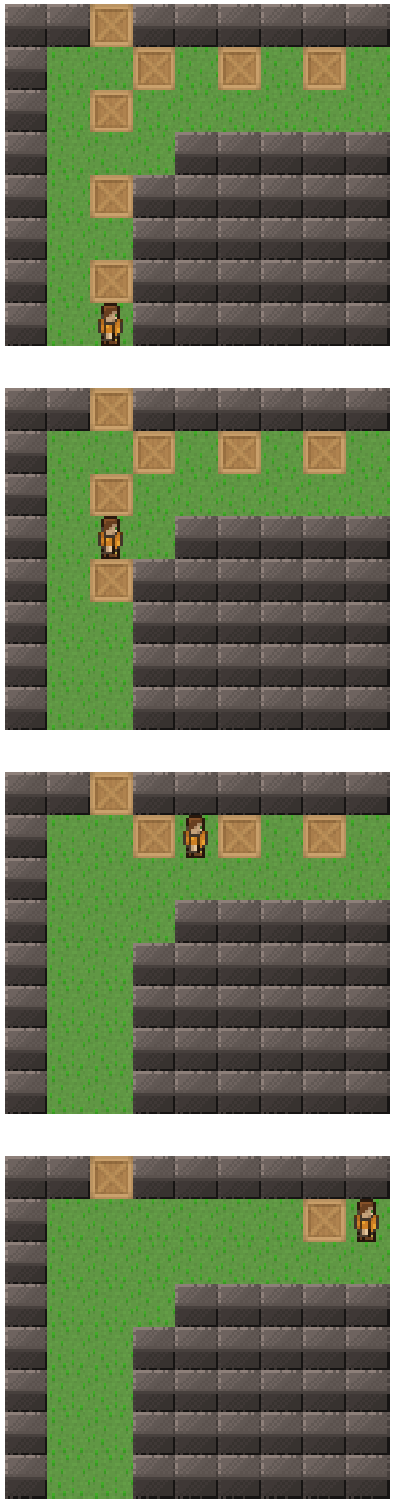
Once we have our components we can start the construction of the board. The only problem is that the combination of components may violate some rules. It is likely that randomly connected components do not satisfy the rules described in Section 5 even though the individual components do. In this section we will describe a standard for combining components that will ensure that the rules are still satisfied.
In this section we will often speak about connection points. These are the points where the wire of a component connects to the wire of the neighbouring component. We start out however, with a



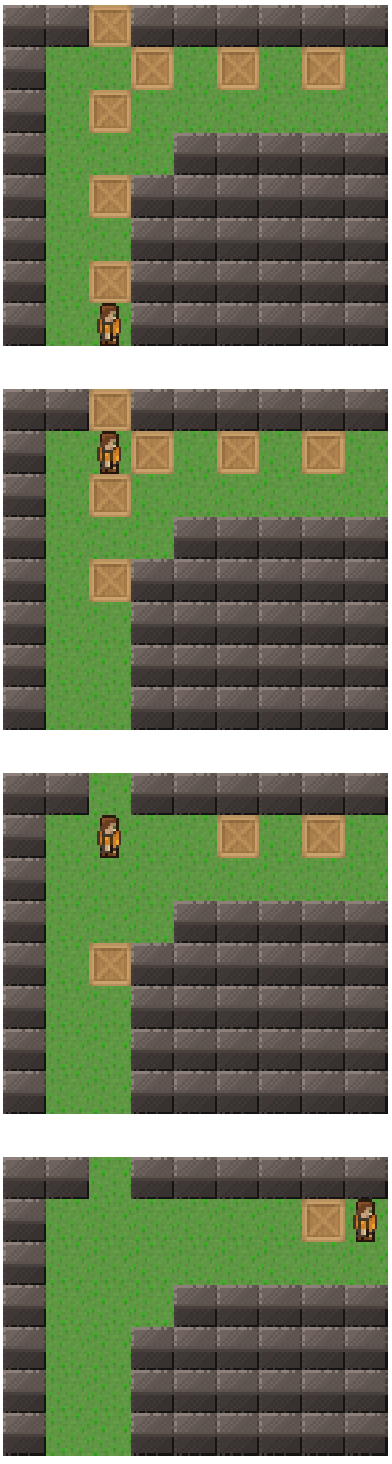
(a) A straight line of crates that is destroyed in the intended way

(b) A straight line of crates that is destroyed such that the clause crate is destroyed

Figure 13: An example where a straight line of crates causes a problem.



(a) A corner of crates that is destroyed in the intended way



(b) A corner line of crates that is destroyed such that the clause crate is destroyed

Figure 14: An example where a corner of crates causes a problem.

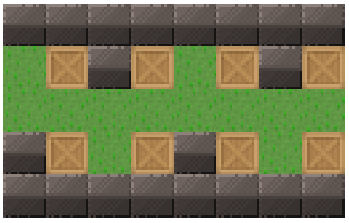
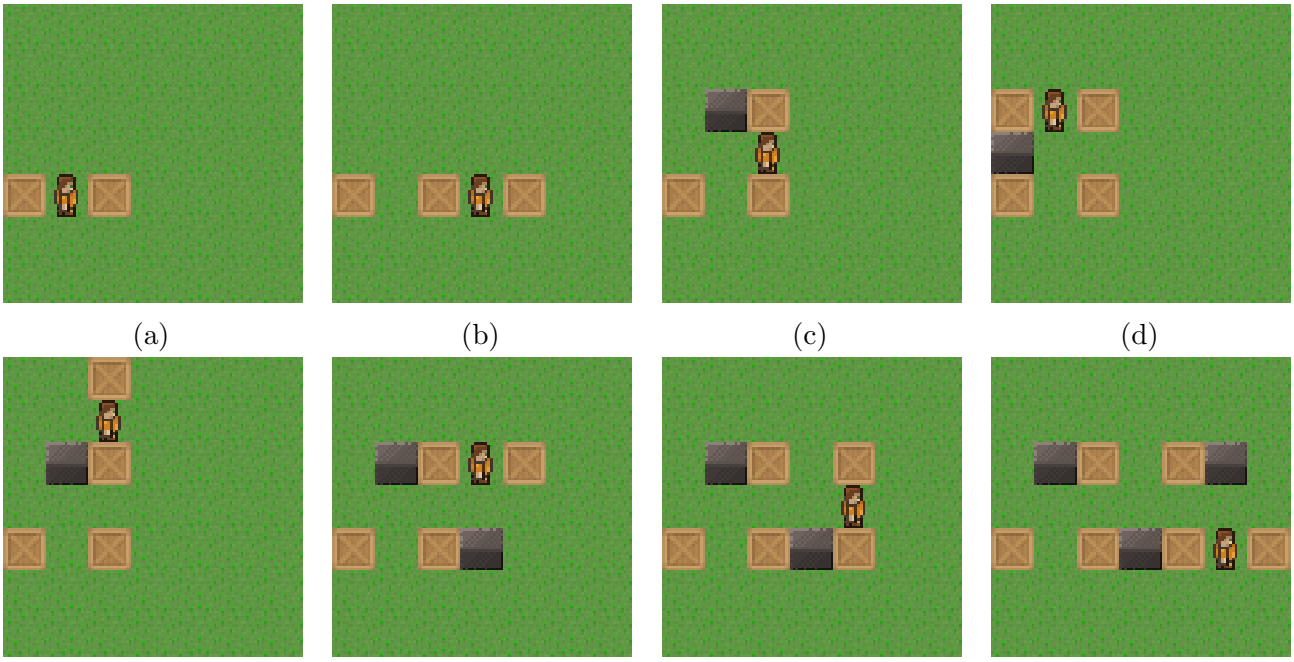


Figure 15: The new snake like structure for a straight path

pattern that will be omnipresent in our reduction: the snake pattern.

6.1 Snake

In this thesis we use a snake like pattern when we have a straight moving wire as can be seen in Figure 15. In this section we will show how this component came to be and why it is so special in our reduction. We construct a path step-by-step going from left to right. Our player will indicate where we currently are on the path and thus where we can continue from. This can be seen in Figure 16. At each step, we will verify if the current component is still valid according to the rules from Section 5.
We start in the position seen in Figure 16a. Here we have a crate to the left and right side of our player. As described in rule 1 we keep this distance of the crates so that the bombs with the minimal explosion range will still be able to destroy two crates at the time.
Our first try is to simply go right from our current position 1 as can be seen in Figure 16b. Here we already come to a halt as we created a situation where we have three crates in a row. This is forbidden by rule 4 so we have to backtrack one step.
Now we try to go up as shown in Figure 16c. Here we have the possibility for the upper and left crate to be jointly destroyed by a bomb with range at least 2. For this reason we place a wall to prevent this. In this case the wall does not obstruct the intended path. In our previous attempt we could not add a wall. If we tried, then we would have to place a crate between the leftmost and rightmost crate. The only possibility would have been to place it on their path thus making the intended path impossible.
Since our current situation in Figure 16c is still valid we try to continue from here. Our first attempt will be going left as can be seen in Figure 16d. Here we move the wall so that our previous problem is still solved, this wall also solves the problem where the left two crates could be destroyed together as well. Although this is still a valid component we are moving backwards. If we continue from this point we will be standing more to the left then our starting position so we are not making any progress. For this reason this option is invalid for our current goal.
We backtrack and try to go up as depicted in Figure 16e. Here we have the same situation as in Figure 16b where we have three crates in a row, which is not valid.
Hence, we try to go to the right as shown in Figure 16f. Here we have have to add a wall between the lower and upper right crates so that similar to Figure 16c we prevent the crates from being destroyed together. With this we have a valid component to continue from.
From this position we can go up, down and right. If we go right we again create a situation where we have three crates in a row. In case we go up we go further away from our original horizontal axis. While this is a valid action we essentially end up in a situation similar as Figure 16c. We can



(a) (b) (c) (d)

(e) (f) (g) (h)

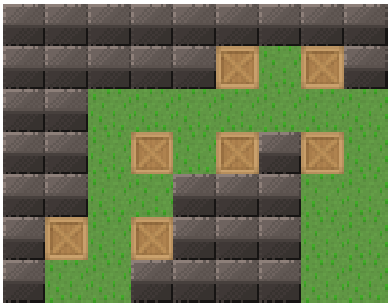
Figure 16: Snake explained

repeat this and go up and right alternatively. However, our goal is only to go right, for this reason this situation is also not used.
If we go down we stay close to our original horizontal axis and we are still in a valid state, which is shown in Figure 16g.
This situation is similar to the one in Figure 16c. We conclude that the only valid next step is to go right, as shown in Figure 16h. From this position we can keep on repeating this and by adding lines of walls along the upper and lower crates we have exactly the snake from Figure 15.

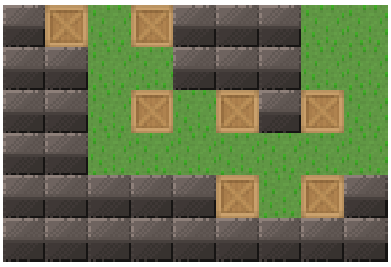
6.2 Connecting

As stated at the beginning of the section the combination of components may not be a valid component. That is why there will be a few more rules concerning the connection points.

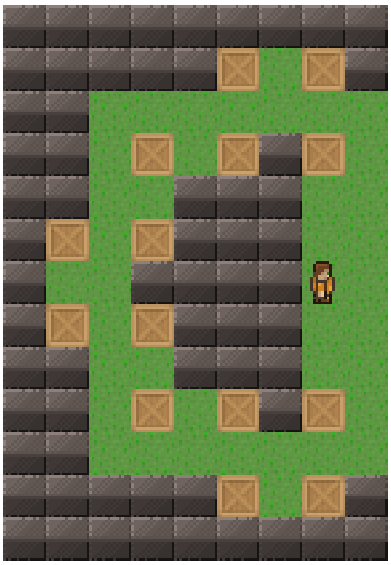
1. A component may comply to all the rules from Section 5 but explosions leaving the component may have undesired effects on other components. When we only consider this component we do not know what the explosion could hit when it is connected to other components, so we have to apply more rules where the other components are taken into consideration.
   Rule:
   An explosion can only escape the component from the connection points.
   Motivation:
   As stated before the separate components can comply to the rules, but the combination may create new possibilities for crate breaking. A position that would normally hit only one crate could result into more crates destroyed when another component is present.
   An example of this can be seen in Figure 17. Here the two separate components comply



(a) First part to be connected



(b) Second part to be connected



(c) Both parts connected

Figure 17: An example where connecting two components causes a problem.

to the rules described in Section 5. However, when you combine the two as can be seen in Figure 17c the player can stand in between the two crates from the different components. From this position he is able to destroy four crates with two bombs and therefore the combined component violates rule 4. Note that the connection points are where the wires go, the wire comes in from the right and leaves at the right. Here the components do not stop the explosion from going up or down from the right end of the component respectively.

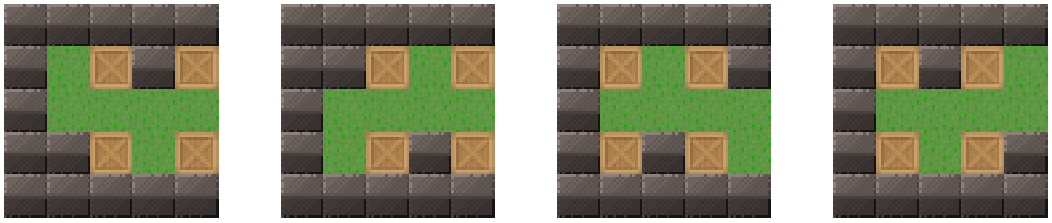
2. As discussed earlier we want to use a snake pattern to connect two components. The snake pattern can end in four possible ways, as depicted in Figure 18. Note that we only visualize the situation where the border is at the right or bottom side, however the left and upper side can be obtained by rotating the picture.

   cright This stands for crate right. As depicted in Figure 18a the crates are at the border where the other component should connect. Because there exist two situations where this can happen we also describe where the previous wall is located in order to get rid of this possibility. In this case the wall is on the right side when you look into the component from the connection point.

   cleft This stands for crate left and can be seen in Figure 18b. Similar to **cright** the crates are on the border but the wall is now on the left side behind the crates.

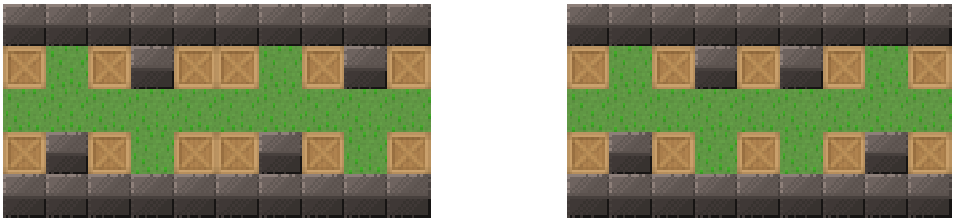
   wright This stands for wall right as can be seen in Figure 18c. Here, unlike with the crate variant, the wall is now on the border of the component and the crates are behind it. When you look inside from the connection point, the wall is on the right side.

   wleft This stands for wall left down as shown in Figure 18d. This is again similar as **wright** but now the wall can be seen at the left side.



(a) cright (b) cleft (c) wright (d) wleft

Figure 18: The different connection points on the right of the component, for demonstration the right orientation are visualised, the left is blocked off for visualisation purposes only



(a) crate on crate fault (b) left on right fault

Figure 19: Faults

Rule:
You can only combine **cright** with **wright** and **cleft** with **wleft**.
Motivation:
When the rule is followed the resulting component will have a full snake pattern where the two parts are connected thus maintaining the rules described before. However if you were to create other combinations the resulting component does not have the desired shape. An example for this is combining two components where both connecting points have crates on the border, the same goes for walls. An example of this can be seen in Figure 19a. Here the crates are touching each other violating rule 1 from Section 5. The distance between the crates is so small that there is only one possible way to destroy all crates with the minimum number of bombs with range 1, which is by placing bombs in the green 'corridor' in the middle. This implies that we can only evaluate it as either true or false, but we will never be able to try out the other evaluation. The same goes for combining a left end with a right end, this can be seen in Figure 19b. Here the walls allow only one way to combine the middle crate in the upper row with another crate, i.e., with the crate below it. Just like in the previous case, this yields only one possible evaluation of the wire.

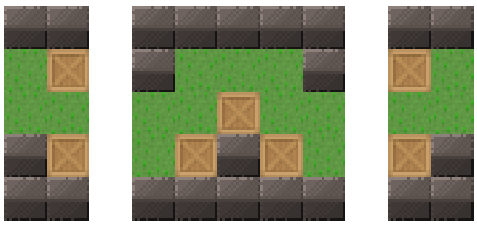
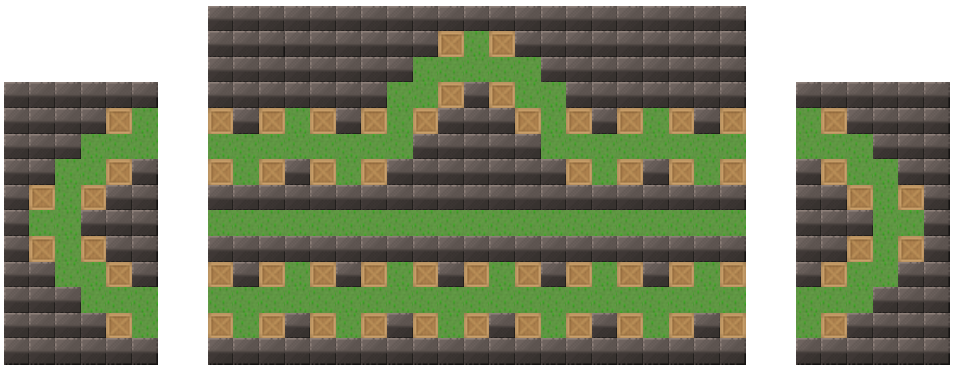


Figure 20: corridor fault

3. In the snake pattern we can observe a kind of corridor. A corridor is the straight line in which you can walk without interruptions of crates but still passing by all the crates.
   Rule:
   A crate cannot be placed in or next to a corridor.
   Motivation:
   As previously stated a corridor should be a straight line without any interruptions. Observe that we can place bombs in the corridor and in half of the cases we also destroy up to two crates as can be observed in Figure 15. If a crate is placed in the corridor we immediately create a problem because we can destroy three crates with one bomb which violates rule 4 from Section 5. We can conclude that in a single component this rule does not add anything, however when we combine two or more components this may not be the case anymore. An example of this can be seen in Figure 20. We have three components, one component is placed between two other components. To the left of the center we have a **cleft** instance and it correctly connects to the **wleft**, the same goes for the right side where we have an instance of a **wright** connecting with **cright**. The individual components satisfy all the previously described rules. However when we combine them, the crate in the middle of the center component gets destroyed when you place a bomb with range 3 or more in the corridor on either the left or right component.

Finally there is still one problem that exists even if you follow all of the rules; a shift in the system. The most notable effect can be seen in Figure 21. Here we see two wires in the horizontal direction, however the u shape on the left does not connect correctly, as the component on the right does. On the bottom left of the connection point we have a wright trying to connect to a cleft, which is not allowed. This problem can occur when a 2-shift exists somewhere in the wire. In this example we have two wires, at the start the crates are synchronised with each other and so are the walls. However at the end the crates are still in sync, but the walls are now on opposite sides of the corridor. This is caused by the "hill" in the middle of the upper wire.
To make this more precise we define a 2-shift by looking at the inner walls i.e., the wall next to the corridor. An example of these inner walls are the walls used to separate the crates in our straight path that create the snake like form. We say that we have no shift between two inner walls when the Manhattan distance between them is divisible by 4. If we have a remainder x then we say that we have an x-shift. Note that when x is 0 we have a 0-shift, so no shift as expected. This is illustrated in Figure 21. At the right side of the picture the distance between the inner walls from the upper and lower wire all are dividable by 4, whereas at the left side the distances have a remainder of 2 when you compare the walls from the upper and lower wire. We conclude that a 2-shift has occurred at the left side. Luckily this shift is not a problem in many cases, when you have two 2-shifts happening in the same wire we "add" the remainders and eliminate the 2-shift. Shifts of 1 and 3 will not occur. This can be seen when we look at Figure 22. Here we see that the distances between inner walls from the top wire and the bottom wire have remainders 1 and 3 modulo 4. These two wires make up one component in which we have both a 1 and a 3 shift. However note that this component violates rule 2 from Section 5 because the x-coordinates of the crates in the top wire have a different remainder modulo 2 then the x-coordinates of the crates in the bottom wire, therefore this component is invalid. In general, we see that a lot of the inner walls are adjacent to crates, as this aids rule 4 from Section 5 to prevent multiple bombs to hit more than 2 crates. In this case when two walls have an odd distance, the crates adjacent to them will



(a) u shape left (b) 2-shift (c) u shape right

Figure 21: An example of a 2-shift component where a 2-shift occurs

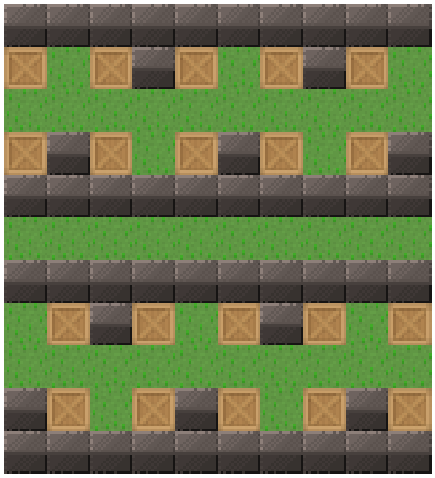
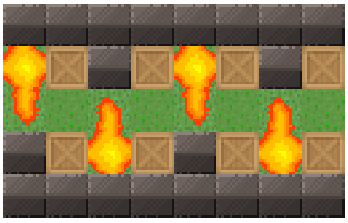


Figure 22: 1 and 3 shift

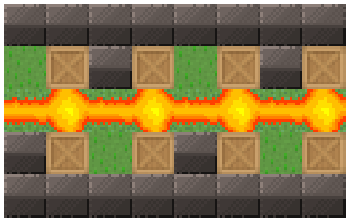
also have an odd distance between each other, again violating rule 2 from Section 5.

7 Components

Making an entire board as one big component can be a lot of work. In this case you only rely on the rules from Section 5 and you have to check every spot to confirm that it is valid. For this reason the rules in Section 6 have been established. By combining both sets of rules we are able to create smaller components that can be combined to make up the final board. As discussed in Section 4 our goal is to make a planar representation of 3-SAT formulae in order to construct a reduction from 3-SAT to Bomb-r-cover and thus prove that Bomb-r-cover is NP-complete. To accomplish this we ran into the issue that two wires could intersect. To solve this problem we introduced the subgraph used in [Lic82] as was shown in Figure 7 and Figure 8. To create a board using these graphs as guidelines we define in this section various components that can be combined using the rules from Section 6.



(a) The straight path evaluated as **true**



(b) The straight path evaluated as **false**

Figure 23: The new snake like structure for a straight path with their respective evaluations

7.1 Straight path

This component is the snake pattern that has already been discussed in detail in Section 6. This snake component is special compared to other components as the length is variable, the shape of the first connection point can be any of the four different types, the shape of the end is determined by the length of the path. These properties follow from to the fact that the component repeats after 4 squares.
In this component the difference between `true` and `false` is visible, see Figure 23. In case the path is evaluated as `true`, the bombs are placed against walls. The crates on the same side of the wire are destroyed together. When the path is evaluated as `false` the bombs are placed inside the corridor, thus destroying crates together on opposite sides of the wire. For consistency this evaluation is used throughout the rest of the thesis. We could have used another definition of `true` and `false` even without this straight path and still have a working reduction.

7.2 Corners

In order to make a loop in our square world we have to introduce at least four corners. In this thesis we introduce two types of corners, called big and small corners, with different connection points. They can be seen in Figures 25a and 25b, respectively.
Most notable about both types is that they introduce a 2-shift in the system as well as parity switch. For the parity switch, consider Figure 25. When in the horizontal direction, bombs are placed against the walls, then in the vertical direction, they are placed in the corridor, and vice versa. We have to take this into account when we arrive at the clause component (see Section 7.4). The 2-shift remains present after a second corner into the same direction (left or right), but is cancelled out by a corner in the other direction. IF the total number of corners to the left in a wire minus the total number of corners to the right is divisible by 4, then the 2-shifts cancel out, and we do not have a problem closing the loop as in Figure 21b. Otherwise, we may have to explicitly use the 2-shift component from that figure. Surprisingly, this may be necessary by the presence of clause components.

7.3 Straight parity switch

Another important aspect of our system is the ability to swap the evaluation or parity of the wire. A corner achieves this, however we cannot use this if we want to stay at in the same direction. To this end the parity switch is introduced, which can be seen in Figure 26. The small "hill" in the

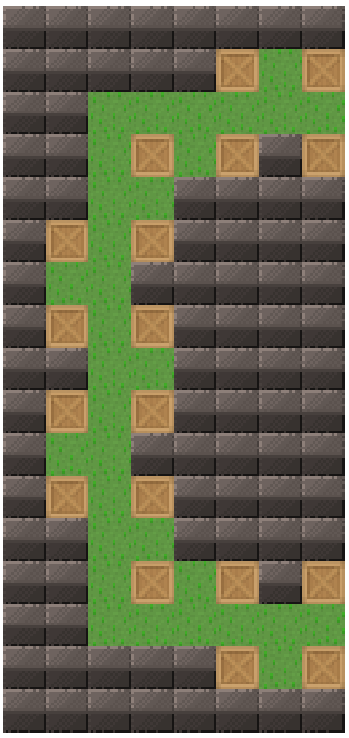
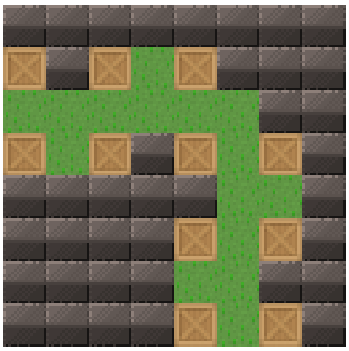
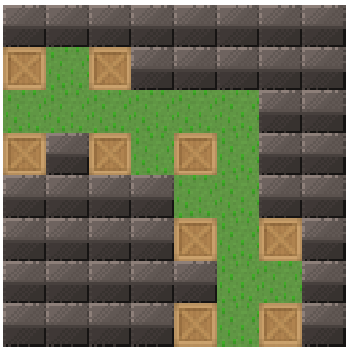


Figure 24: U-turn with corners



(a) Left big corner



(b) Left small corner

Figure 25: The corners used for our reduction.

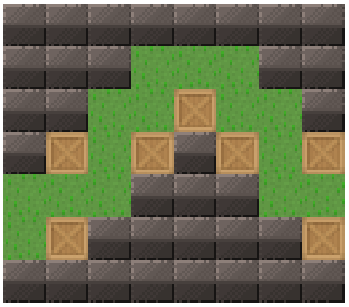


Figure 26: Straight parity switch

component has crates diagonally touching each other. While we would otherwise avoid this, this is used here to make the parity switch happen. Swapping the parity of the wire is the only effect of this component.

7.4 Clause components

Now that the basics of the wire shape and parity have been described we conclude with the clauses. As explained in Section 5, a clause is different from other components by the presence of clause crates. These crates may be destroyed in sets of three instead of the usual two, allowing this crate to be destroyed and still progressing the wire. The number of bombs required to finish the wire stays the same. If the wire is `false` the crate is not destroyed and if this is the case for each wire involved in the clause you have to use at least one extra bomb to remove all the crates on the board, which is not provided. This would make the clause `false`, and therefore the corresponding 3-SAT formula would also be `false`, the correct behaviour. Note that in our case at least one of the wires has to be evaluated as `true`, but this does not imply that the variable corresponding to the wire is `true`. The parity switches in the wire are meant to act as a not for the wire, the only thing that matters is that the evaluation of the wire correctly corresponds to the way the variable is used in the clause. As a result of a parity switch, a wire may locally be evaluated as `true`, whereas the corresponding variable is `false`.

7.4.1 Two variables

In 3-SAT the clauses always consist of three variables. However in our system clauses of two variables are used to make up the intersection. In this component we have two wires connecting to one clause crate as depicted in Figure 27. When the top or bottom wire is evaluated as `true` the crates of that wire are destroyed in horizontal pairs. This way the clause crate gets destroyed when the two crates next to the clause crate are targeted. However when the wire is evaluated as `false` the crates are destroyed in vertical pairs and thus miss the clause crate when Bomberman passes by. If both the wires are evaluated as `false` the crate is not destroyed and you need an additional bomb to destroy this crate, which again, is not provided.

7.4.2 Three and four variables

The component for clauses of three variables can be seen in Figure 28. This component is in its current state simple, however the path towards this design was not. Similar as in the two variable

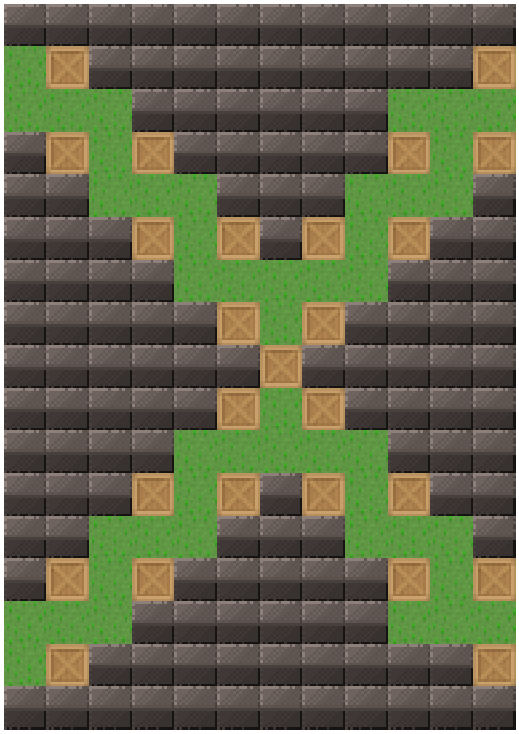


Figure 27: 2 clause

clause component we need to bring wires together using one or multiple clause crates. These crates should be "free" when at least one wire is evaluated as `true` and require at least one more bomb otherwise. Unlike the case with two wires we can no longer use just one crate. The clause crate in Figure 27 is already surrounded by crates, adding another wire would cause intersections problems.

The most logical conclusion would be to use three clause crates. This creates a new problem: in case that only one wire is evaluated as `true` the other clause crates should be destroyed without an additional bomb. The first solution that came to mind was to introduce a wire between the clause crates which can only be accessed if at least one clause crate is destroyed. Now by destroying the crates inside you can take out the other clause crates for free, they are also destroyed in a set of three. This solution had one mayor flaw however, the component created 2-shifts, the 1 shift was created by diagonally placed crates that were used to ensure that the player would pass the clause crate without destroying it. Whilst the 2-shift can be solved, the 1 shift may not exist on the edges of the component by rule 2.

After some tinkering the final design depicted in Figure 28 was created. Unlike the other designs attempted, there is no wire in the middle, but only 2 additional crates for every clause crate. Here the distance and offset between the clause crates do not matter too much, as long as there is a wall blocking explosions between different clause crates. This freedom allows for much more flexible use and can even solve 2-shifts between wires without much trouble. In a similar fashion we can create clause component for four variables as depicted in Figure 29. Similar to clauses of two variables, clauses of four variables do not occur in a 3-SAT formula itself, but are needed for the intersection. Note that the u-turns that the wires make in a 3-clause or a 4-clause do not introduce

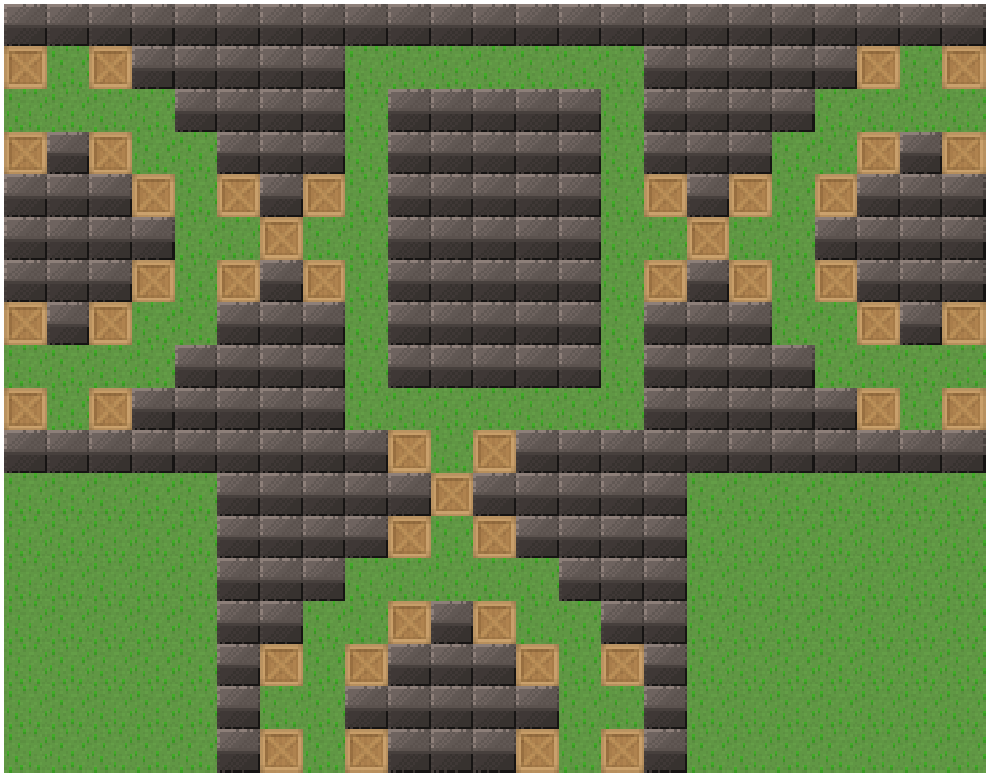


Figure 28: 3 clause

2-shifts, which is different from the u-turn caused by two corners (see Figure 24).

8 Intersection

We use the component from Section 7 in our reduction from 3-SAT tot Bomb-r-cover. However, one important component is still missing. In his reduction from 3-SAT to Planar SAT, Lichtenstein [Lic82] solves the problem of intersection edges by introducing the planar formula. The subgraph described by the planar formula has four connection points, two for each edge. This subgraph can be seen in Figure 8. In this section, we describe a component implementing the subgraph with wires from Bomberman.
The biggest challenge for this component is the parity of the wires: are the variables evaluated in their normal form or are they negated? An additional goal would be to make the component as symmetrical and compact as possible. In Figure 8 we have a clause at every intersection of edges, indicated by a dot. The variables used by the clause are the ones that the edges derive from. The formula for each clause can be seen in Table 1. Using this setup we have constructed the component shown in Figure 30.

In our intersection we distinguish between outer wires and inner wires according to their positioning in the component. The outer wires correspond to a_1, a_2, b_1 and b_2, denoted in blue and red, whilst the inner wires correspond to α, β, γ and δ with the colours green, purple, yellow and orange

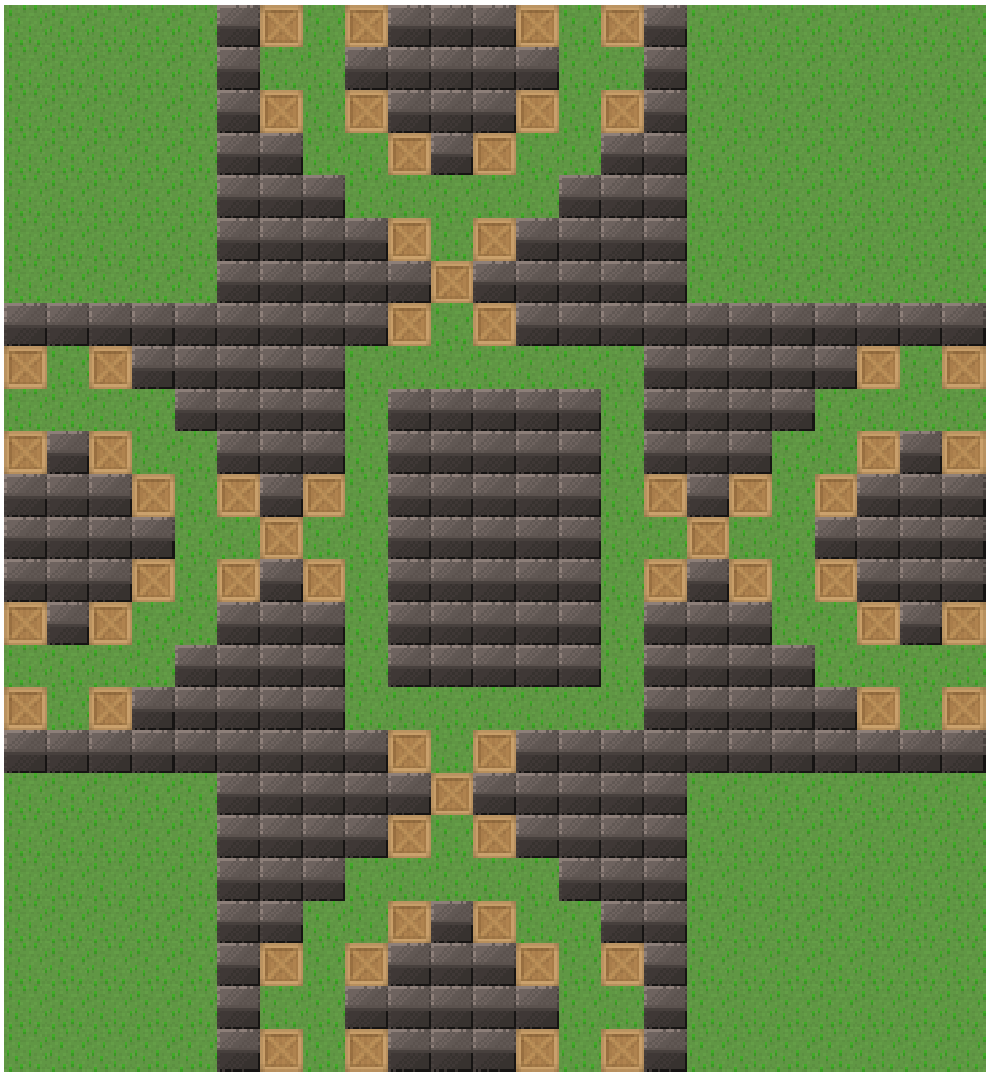


Figure 29: 4 clause

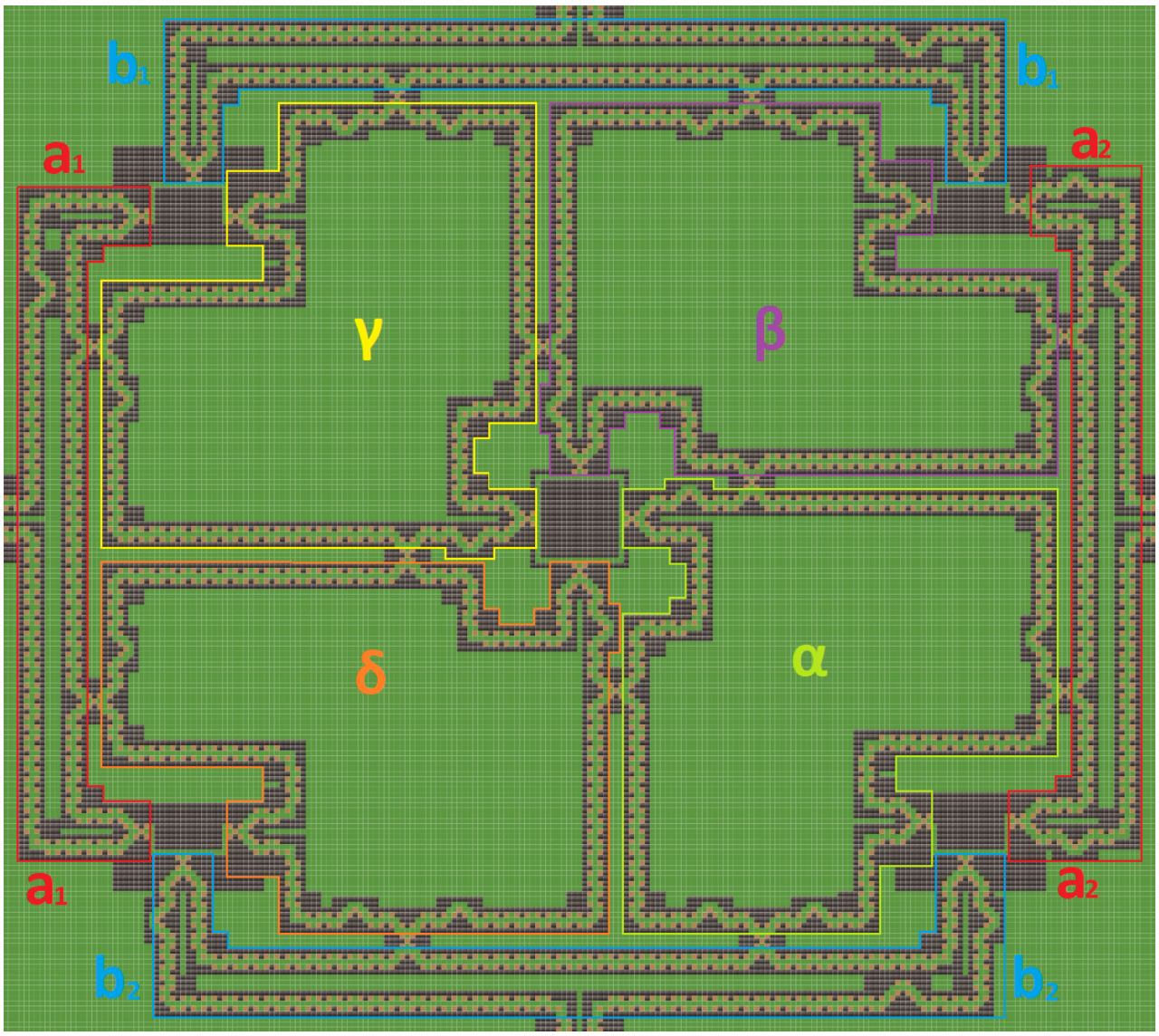


Figure 30: The final intersection with colours representing the various variables

respectively. The clauses present in Table 1 have a clear pattern which will be used to simplify the component and add symmetry. Every outer wire has two clauses with three variables and two clauses with two variables. The three variable clauses connect with the neighbouring outer wires and one inner wire which is unique for every clause. The two clauses of two variables connect the outer wire with each of the inner wires that are also present at the three variable clauses. Every inner wire goes to one clause of three variables, four clauses of two variables and one central clause with four variables. The clauses with three variables and two of the two variable clauses have already been mentioned because they are connected with the outer wires. The other two clauses of two variables and the clause with four variables are created to connect the inner wires to each other. The two clauses of two variables connect two neighbouring inner wires and the clause with four variables is a clause consisting of all inner variables.
To solve the problem of parity for every clause we can use parity switches when necessary. However, as mentioned in Section 7, a corner also swaps the evaluation of the wire, thus functioning similar to the parity switch. This has been used to minimize the number of parity switches used in the intersection.

One can see that (the wires corresponding to) a_1 and b_1 are nearly a rotation of each other. The same goes for a_2 from b_2. The variables a_1 and b_1 enter their respective three variable clauses like normal, without negation. The two corners leading the wires towards the clauses do not change the parity of the wires, thus leaving them in the positive execution. When they meet the inner wires in the two clauses of two variables their occurrence in the clauses should be negated, which is achieved by the corners before and after the clauses. The same goes for a_2 and b_2, however their evaluation are opposite to those of a_1 and b_1. By introducing a parity switch before every three variable clause this problem is solved as the connection from the three variable clause to the two variable clause is again negated correctly as was the case with a_1 and b_1.
However the u-shape of corners for outer wires creates a 2-shift that is not solved, for this reason the outer wires have a 2-shift component to compensate for this problem.

The inner wires are a bit more complicated. To this end we follow the inner wire β seen in purple to understand its behaviour. The other inner wires are rotation of each other meaning that they have the same behaviour, however from a different viewpoint. We start at the four variable clause, $(\alpha \vee \beta \vee \gamma \vee \delta)$, in the upwards direction. In this brief moment we are executed without any negation, if β is `true` then so is the wire. We now proceed towards the two variable clause, $(\neg\beta \vee \neg\gamma)$, with our yellow neighbour γ. This clause is placed in our direction, so we do not need any corners to reach it. However β is used in its negated form so we require a parity switch before connecting to it. After this clause we proceed towards the two variable clause, $(\neg\beta \vee \neg b_1)$, with b_1 seen in blue. In this clause β is also negated, however due to the corner our value going into the clause is positive, thus requiring another parity switch. Now our wire proceeds towards the three variable clause $(\neg a_2 \vee b_1 \vee \beta)$. Here β is used like normal, whilst we exit $(\neg b_1 \vee \neg\beta)$ in the negative form. The two corners connecting the clause do not swap our parity, so we again require a parity switch to ensure that the wire connects with the correct parity. We leave the clause without negation. We again need to go to a two variable clause, but this time with a_2 marked in red, $(a_2 \vee \neg\beta)$. We require three corners to connect our wire to the clause. Now this number of corners is odd, so our parity is also switched as desired. After the clause with a_2 we proceed toward the clause $(\neg\alpha \vee \neg\beta)$ with our other neighbouring inner wire, α seen in green. Similar as with γ we need a parity switch to ensure the correct behaviour in the clause as the corner swaps our parity.

To connect back to the four variable clause we use three corners, similar with our exit from the three variable clause the number of corners is odd, leading to a parity switch. This is what we want as the value of β was negated in $(\neg\alpha \vee \neg\beta)$.

Finally our intersection does not require any passages from one wire to another. This is due to the behaviour of clauses, when our wire is `true` we destroy the clause crate and we have free access to the other wire(s) involved in the clause. In case of the two variable clause, we destroy the clause crate and thus get an opening to the other wire. With the three and four variable clauses we can destroy the remainder of the clause crates when we destroy the clause crates from the inside. This should be done even if all the wires involved in the clause are `true`, so this does not change the number of bombs used.

Our intersection ensures that a wire makes at least one clause `true` due to the properties of the formulae in Figure 1. Say we start in a_2, our variable may be `false` at the three $\neg a_2 \vee \neg b_2 \vee \alpha$, however the variable is then `true` at the two variable clause $a_2 \vee \neg\alpha$, as our wire is now negated. So a_2 either has access to both b_2 and α or only to α. This implies that we will always have access to the wire of α. Similar to this we can use α to get to b_2, then from b_2 to δ and we can keep doing this until we have access to all the wires. By following a path from outer wire to inner wire we can guarantee that we get to every wire from any wire.

With all of these properties we can conclude that the constructed component correctly represents the planar graph from Figure 8. This component satisfies all the rules from Section 5 as it only consists of components defined in Section 7 and connects them using the rules from Section 6. We can therefore use the component to implement the intersection of two wires, which we need in our reduction.

9 Graph

All the components used in the reduction have now been defined. In this section the reduction will be described. To prove that Bomb-r-cover is NP-hard we construct a reduction from 3-SAT to Bomb-r-cover as mentioned in Section 4. We use the intersection component from Section 8 at intersections of wires. The shape of the overall graph is similar to the solution proposed in [Lic82]. Figure 31 shows a schematic representation of a Bomberman board for a 3-SAT formula with three variables and two clauses. Every variable is represented by a wire, each with a different colour, and these wires start next to each other at the bottom of the board. A wire has an extension to every clause it is used in, keeping enough distance from each other to ensure that the intersections get enough space. Every variable has a hole on the bottom where the explosions from outside cannot hit any crates, to allow Bomberman a passage to the other wires.

As our board represents a 3-SAT formula, all the clauses on the left support exactly three variables. Every clause is of a T-shape, as seen in Figure 28, that is pointed towards the incoming wires. We need to make sure that the wires enter the clause with the correct parity specified by the clause. Sometimes this is achieved by the parity switches caused by the corners, however sometimes a parity switch component is required. We assume that the value going upwards from the initial

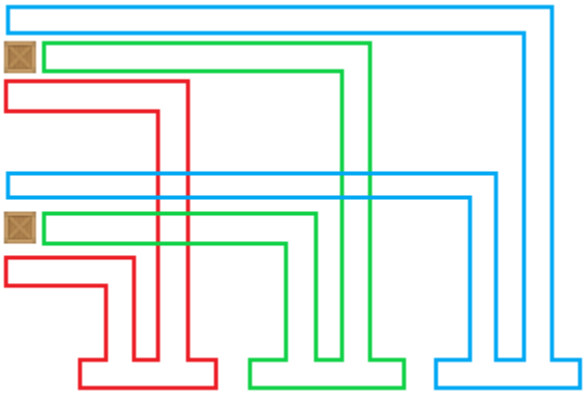


Figure 31: An example of board depicted with a schematic representation

wire is the value as the variable. The top and bottom wire require two corners from this point on, whereas the central wire only uses one. We conclude that the top and bottom wire are evaluated in the same manner as its variable, whereas the wire approaching from the front is negated. To correct this for the center or to negate the top and bottom wire, one has to introduce two parity switches, for the incoming and outgoing wire.

Using the intersection component we are able to connect any number of variables to any number of clauses. When a variable is used by a clause, the corresponding wire moves from the initial position to its clause crate, ensuring the correct parity of the wire. This way the clause crates can be destroyed for free when the clause in the 3-SAT formula is `true`. When a clause is `false` in 3-SAT, the player is required to use one extra bomb to destroy at least one of the clause crates. The total number of bombs is as such:

$$(\text{number of bombs}) = \frac{(\text{number of crates}) - (\text{number of clause crates})}{2}$$

As seen the bomb that is now required is not provided, resulting in a `false` instance of Bomb-r-cover. This implies that Bomb-r-cover instance is a yes-instance if and only if the 3-SAT formula is satisfiable. Obviously, the height of the board is proportional to the number of clauses. As each clause is connected to three wires, deriving from the variables at the bottom of the board, the width of the board is also proportional to the number of clauses. Therefore this reduction is polynomial in the number of clauses in the 3-SAT formula. We conclude that Bomb-r-cover is NP-hard. As [Vla19] already proved that Bomb-r-cover is in NP, we can conclude that Bomb-r-cover is NP-complete.

10 Conclusions and Further Research

We have shown that the decision problem Bomb-r-cover, are k bombs with a given radius $r \geq 1$ enough to clear a Bomberman board of crates, is NP-complete. Bomb-r-cover was already proven to be in NP in [Vla19]. The proof that Bomb-r-cover is NP-hard was not concluded as the intersection proposed might introduce parity switches in the wire. We have solved this problem, using the crossover gadget introduced in [Lic82]. The subgraph connects to the intersecting edges and introduces new clauses that connect to each other without needing to intersect. This gadget was then

modeled as a Bomberman board using components described in Section 7. This gadget together with the construction of the board use several rules described in Sections 5. These rules ensure that player is only able create a `true` instance of Bomb-r-cover if and only if 3-SAT is `true`. By combining the components using the rules from Section 6 we can construct a graph for every 3-SAT formula that validates for the reduction.

Our current intersection component has a lot of open space and some components are made bigger just to make them more understandable. It might be interesting to look at ways to compact the intersection, what would be a minimal intersection component? The corners gave issues in our construction as they introduced 2-shifts and locally invert the value of the wire. It may be worth examining corner components that do not have such side-effects. Another interesting problem that has not yet been covered is trying to reduce Planar 3-SAT to Bomb-r-cover without using intersection components. In this paper, we reduce 3-SAT to Bomb-r-Cover and we can use the exact same construction for a reduction from Planar 3-SAT to Bomb-r-cover. However, since the incidence graph of a planar 3-SAT formula is planar, it can be drawn without any intersections, e.g., with the algorithm from [CP95]. It should be possible to extend this algorithm to produce an equivalent instance of Bomb-r-Cover. The main challenge would be the space limitations as the wires take up more space and have limited directions compared to arbitrary edges in the actual planar graph.

References

[CP95] M. Chrobak and T.H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman, 1979.

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[Vla19] R.A. van de Vlasakker. Bomb-cover een verzamelingenoverdekkings-probleem in de bombermanwereld. Thesis Bachelor Informatica. Leiden University, 2019.