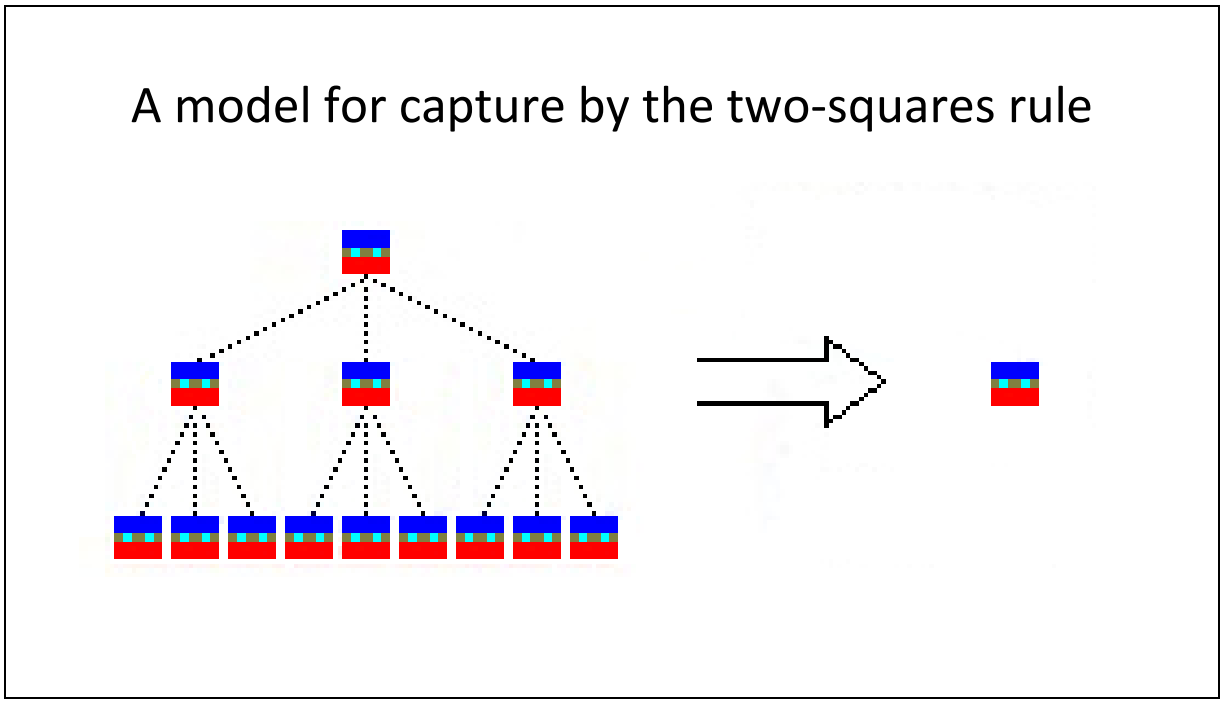


Summary

This document describes a model for game positions where a capture can be enforced by the two-squares rule. A study of moves in some current Stratego programs shows that apparently no static look ahead is used for situations where the program should foresee a capture that may be enforced by the two-squares rule.

There are good reasons for this shortcoming. A barrier is the number of moves required to evaluate positions where the two-squares rule is in force. This number is too large for analysis by brute force search trees. But a conceptual barrier too exists. Although human players in almost one glance distinguish capture or escape by the two-squares rule, the application in a program should cover all possible conditions and that is a rather complex matter.

An exhaustive analysis of all possible conditions in game positions is required. The resulting model is a base for effective algorithms, that only use the characteristics of a current game position instead of search trees. This kind of static look ahead probably will improve the playing strength of any Stratego program significantly.

Contents

1 Introduction

In International Stratego Federation rules contain two chapters that define aspects of the two squares-rule:

- Chapter 10:The two-squares rule
- Chapter 4, paragraph 4: Unsporting behaviour.

The following paragraphs contain extracts of these chapters.

1.1 ISF chapter 10: The two-squares rule

10.1 Continuous moves on two squares
It is not allowed to move a piece more than 3 times non-stop between the same two squares, regardless of what the opponent is doing. It does not matter whether a piece is moving and thereby attacking an opponent's piece, or just moving to an empty square.
10.2 Continuous scout moves on one line
When a scout is involved in the Two-Squares Rule, a scout is considered to start on the starting position of his move plus all the squares he steps over, and he ends on the final position of his move plus all the squares he steps over.

1.2 ISF chapter 4, paragraph 4: Unsporting behaviour.

4.4 Unsporting behaviour
Examples of unsporting behaviour include:

- …
- prolonging the game deliberately by trying to capture one or more of the opponents pieces which cannot be taken because of the Two-squares rule or the More-squares rule.
- …

1.3 Problem definition

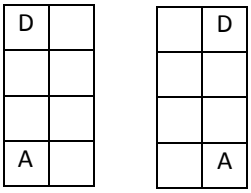
This is a discussion about captures, not about other uses for the two-squares rule.

Obviously most on Internet available Stratego programs are compliant with the rules shown in paragraph 1.1. But that's all.
Most current programs apparently do not use some kind of look ahead that enables them to foresee a capture in game situations where the two-squares rule is in force. If distances are (very) short then incidentally they see a possible gain or threat. But in most game positions they are not able to realise possible gains or to evade possible threats that may be enforced by the two-squares rule. And moreover by lack of sufficient look ahead they afford themselves lots of senseless and annoying chases that a strict enforcement of rule 4.4. should forbid.

It's a fact that any somewhat experienced Stratego player in one glance sees whether a capture will result by enforcement of the two-squares rule in a game position. This strongly suggests that a thorough analysis will enable the definition of a model that encompasses all possible and relevant conditions. The resulting model will enable the construction of algorithms that provide a static look ahead to programs and thereby will improve playing strength significantly.

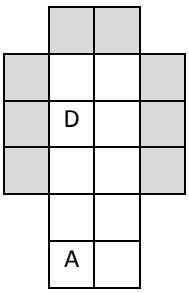
2 Steps towards a capture by use of the two-squares rule

The two-squares rule may lead to a capture in positions where attacker and defender stand on one line and attacker has free access to defender. A second requirement is restriction of the mobility of the defender to a rectangular or square area with a width of two lines.



A look ahead function has to do a search for open lines between attacker and defender. And a search is required for square or rectangular areas with a width of two lines and containing both attacker and defender.

Ideally all relevant border squares of the rectangular area should present a barrier for movement of the defender, for example:



The most common restrictions to the mobility for the defender can originate from:

1. The border of the board
2. A square occupied by a piece of the defenders side
3. An square that has an attacker with a higher rank next to it.

The defender can stand on the attackers half of the board. This does not happen much but is it a theoretical possibility that should be taken into account. Barriers in the border may originate from:

1. The border of the board
2. A square occupied by a bomb if the defender is not a miner
3. A square that has an attacker with a higher rank next to it.

But it may happen that not all border squares are presenting a barrier. In that case it is necessary to check whether accessible border squares enable an escape. The evasion to a resort through an accessible border square is the most complex part of the look ahead function, but it is possible to analyse all relevant possibilities completely and distinguish between a capture and escape.

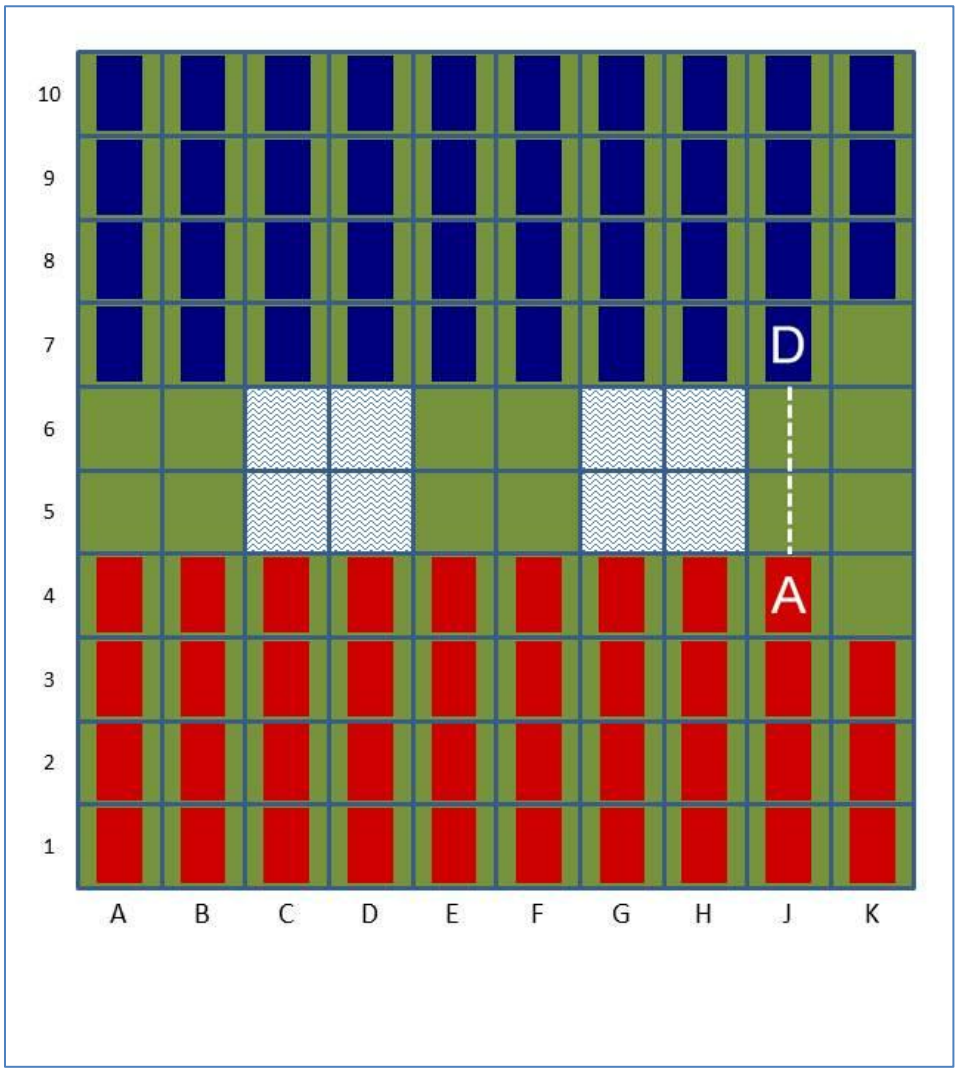
All conditions mentioned here will be illustrated by examples in the following chapter. The sequence is from simple to complex.

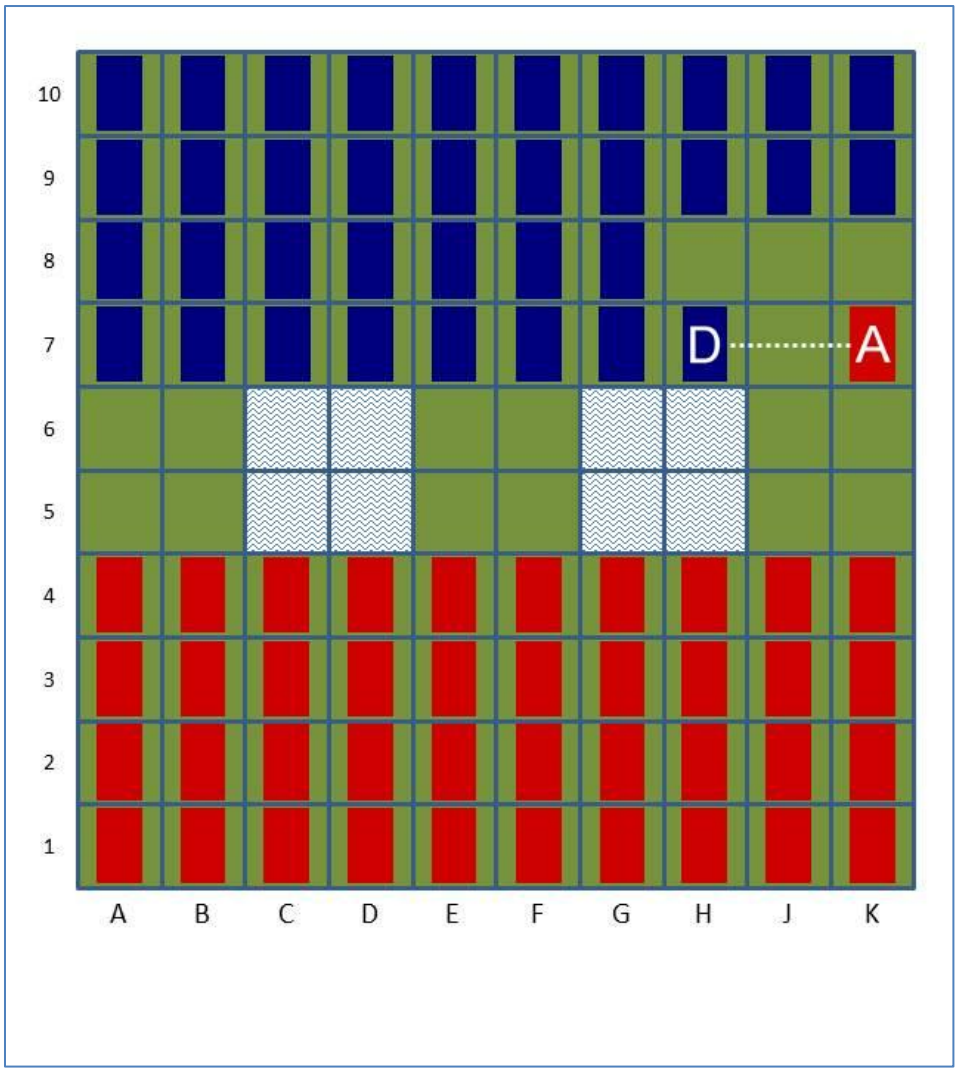
3 Step 1: A line gives attacker access to defender

3.1 An open line connects attacker and defender

3.1.1 Example of a straight and open connection

In most game positions this is a trivial condition.

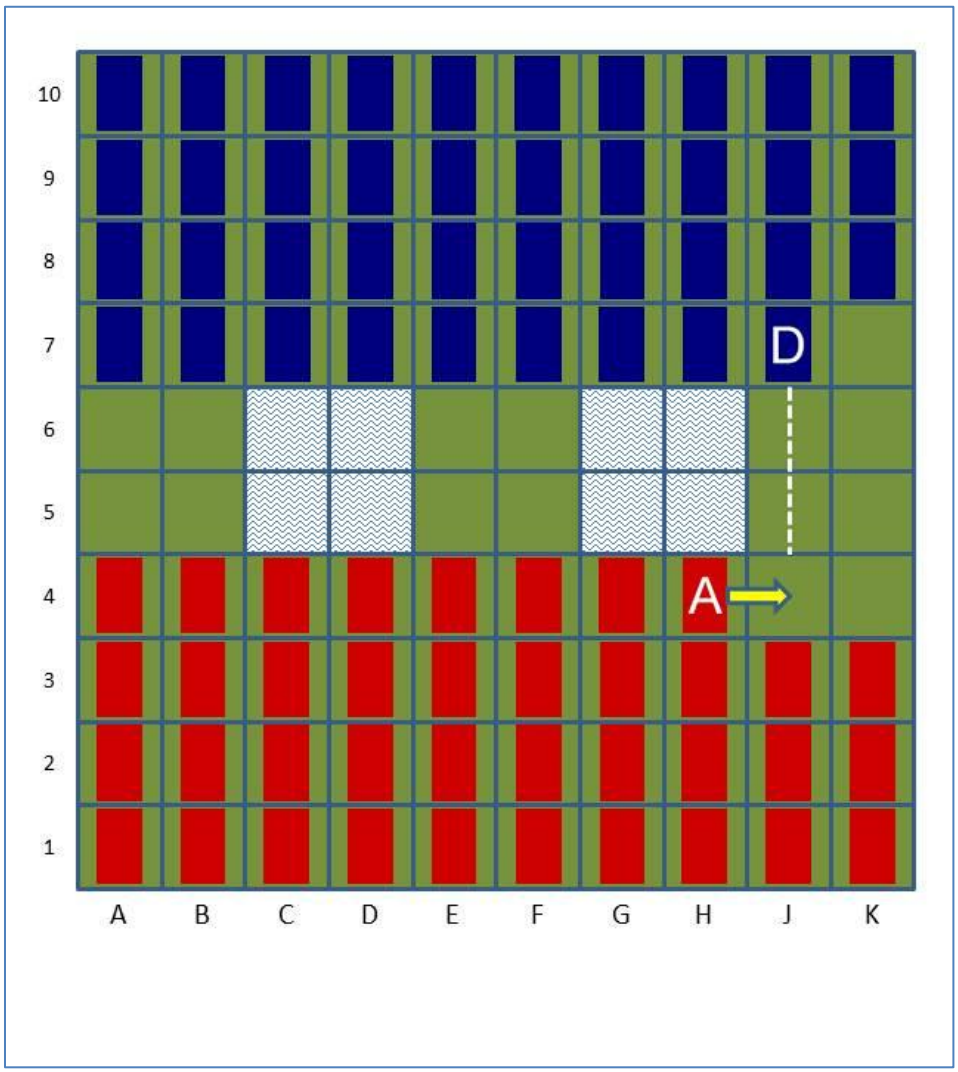




Horizontal lines are possible too.

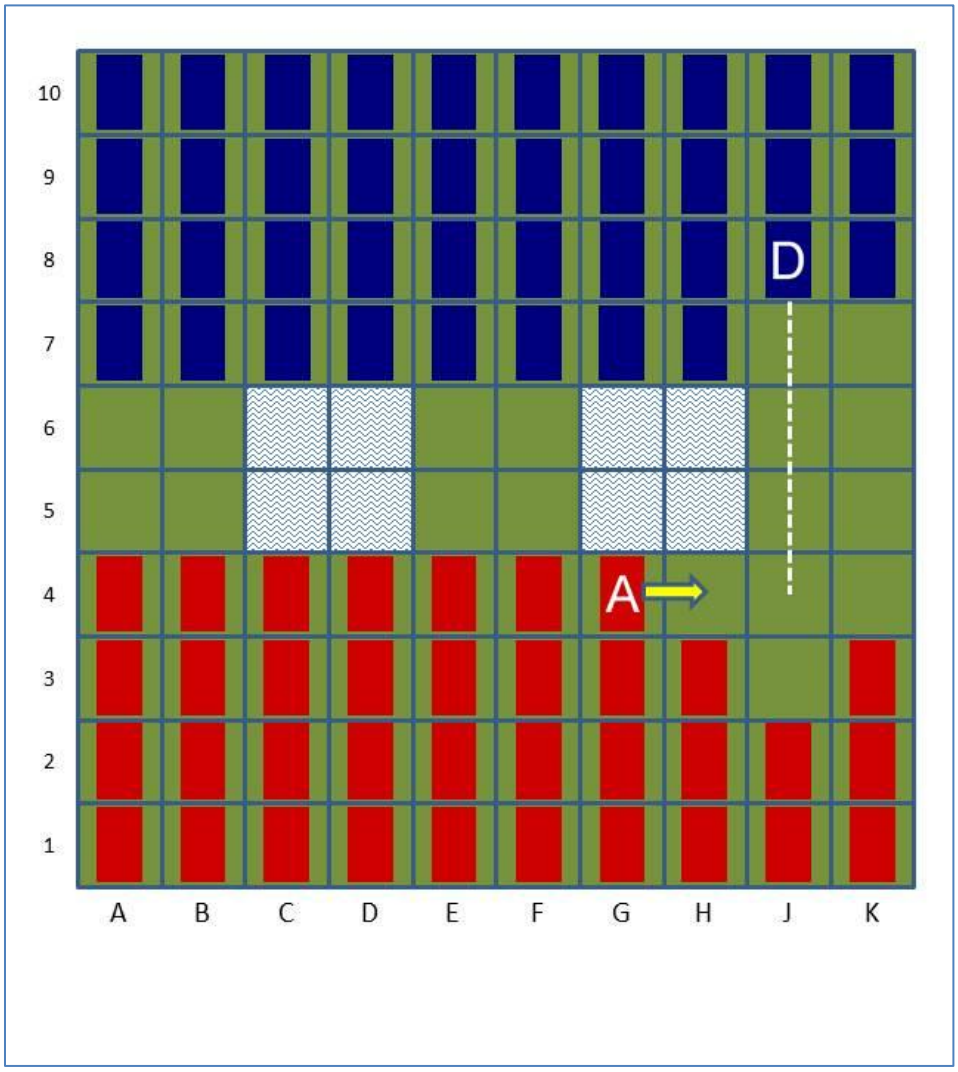
3.2 Attacker threats to attain an open line between attacker and defender

3.2.1 Example of a threat



A move of the attacker to the right produces a position where an open and straight connection exists between attacker and defender.

3.2.2 Example of a more complex threat



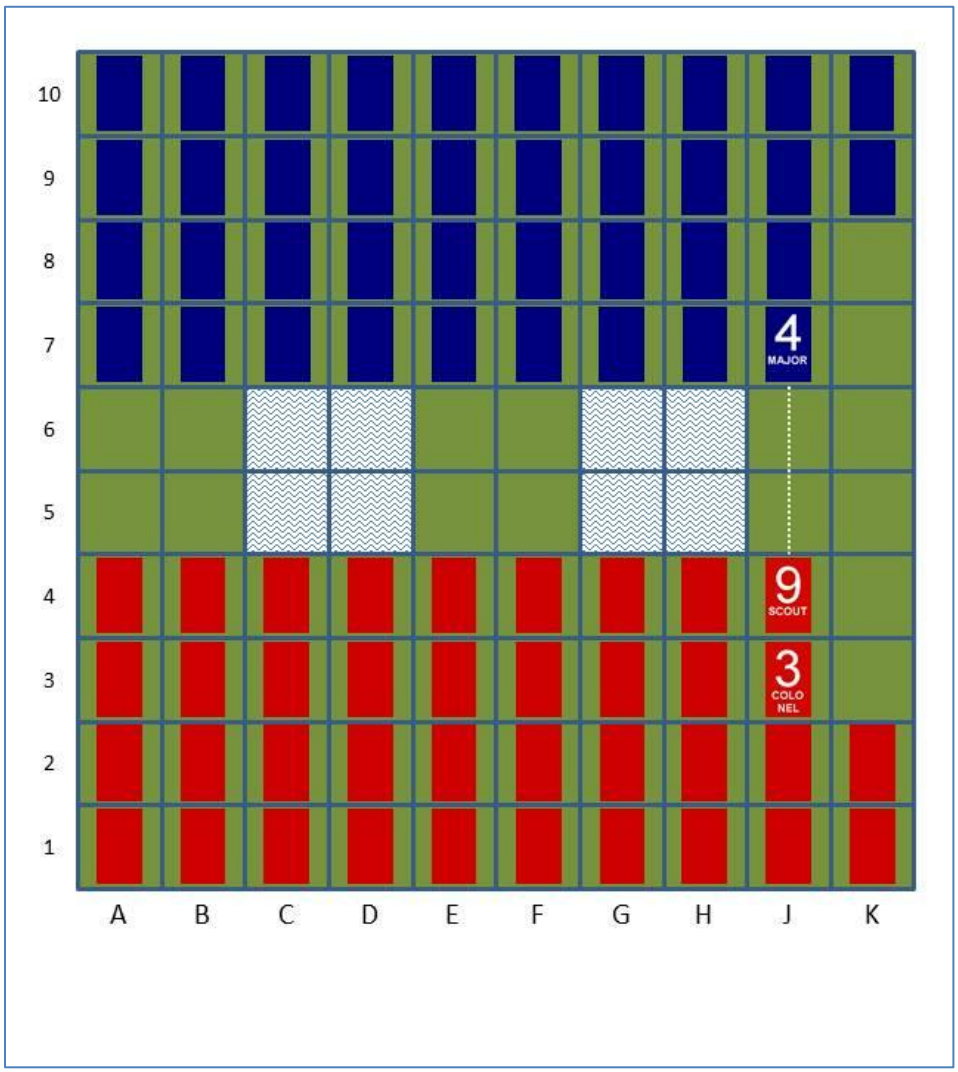
After the detection of the rank of the blue defender a red piece should be chosen with a sufficient rank. In this position the only available attacker with a higher rank stands on G4.
If the attacker is in turn then the move G4-H4 prepares the move H4-J4 by which attacker and defender will be present on the same line (the J column). The defender cannot evade this attack in time by a move to the K-line.

If blue is in turn then blue can prevent this threat by enlarging the mobility of the defender. This can be done by a move of the defender to J7. Mobility of D7 also can be affected by K8-K7 if the unknown piece on K8 has not the rank of a bomb (or – very unlikely – the rank of the flag).

This position emphasises the vulnerability of known pieces that have a restricted sideway mobility. Normally in such situations there is need for enlarging the sideway mobility.

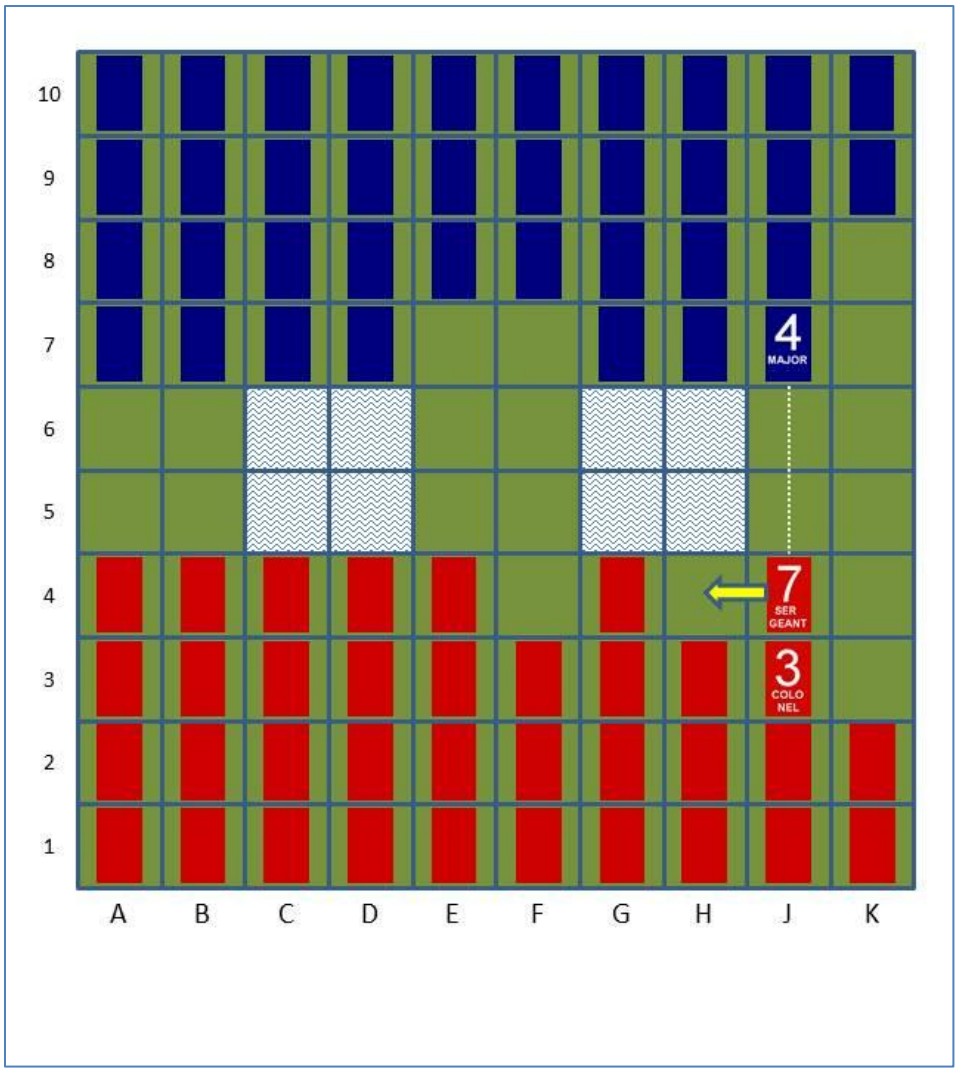
3.3 A piece blocks the line between attacker and defender

3.3.1 Access is possible by elimination of a blocking piece of the attackers side



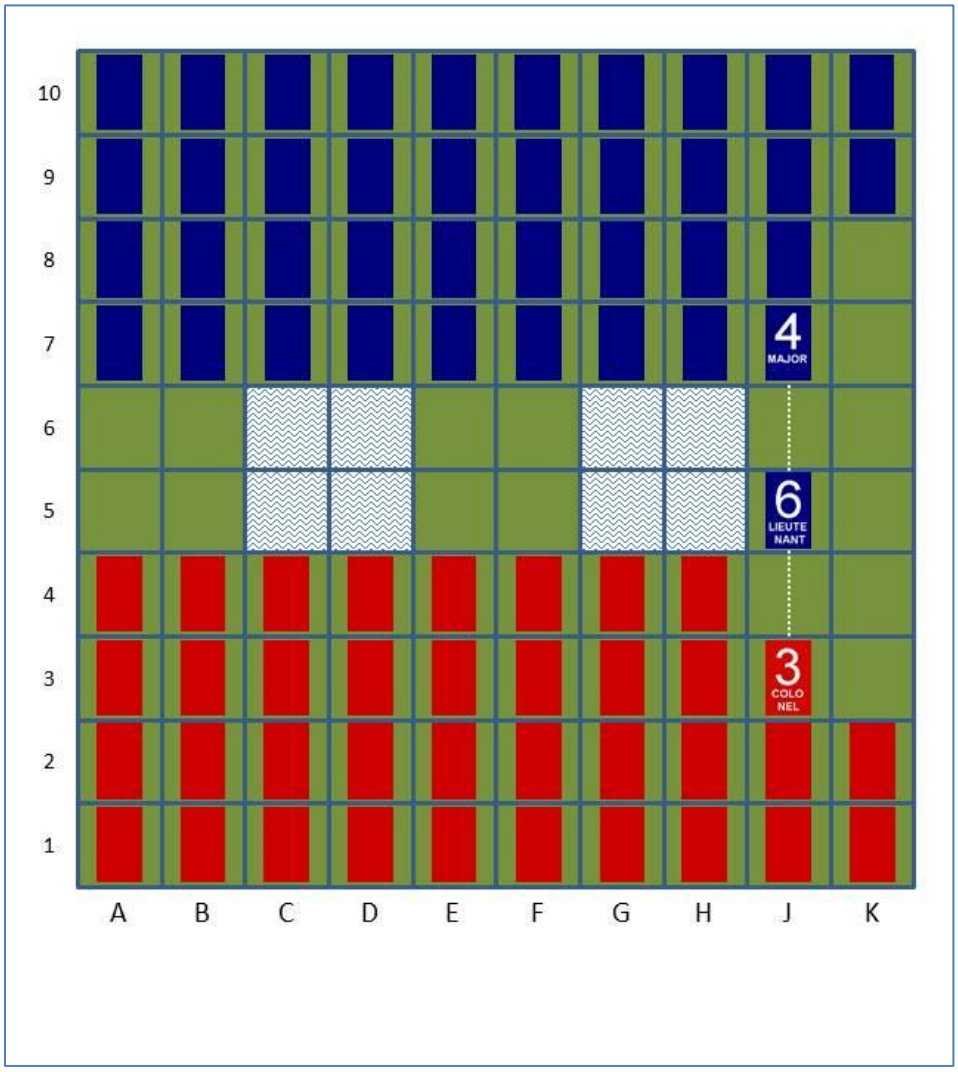
If red sacrifices the scout by the move J4-J7 then an open line has been created between the red colonel and the blue major.

In principle instead of a scout blocking pieces with a different low rank like miner, sergeant or lieutenant can be sacrificed too, but the sacrificing will require more moves.



In this example no sacrifice is necessary. The blocking piece is able to do a sideway move in order to create a free access to the blue major.

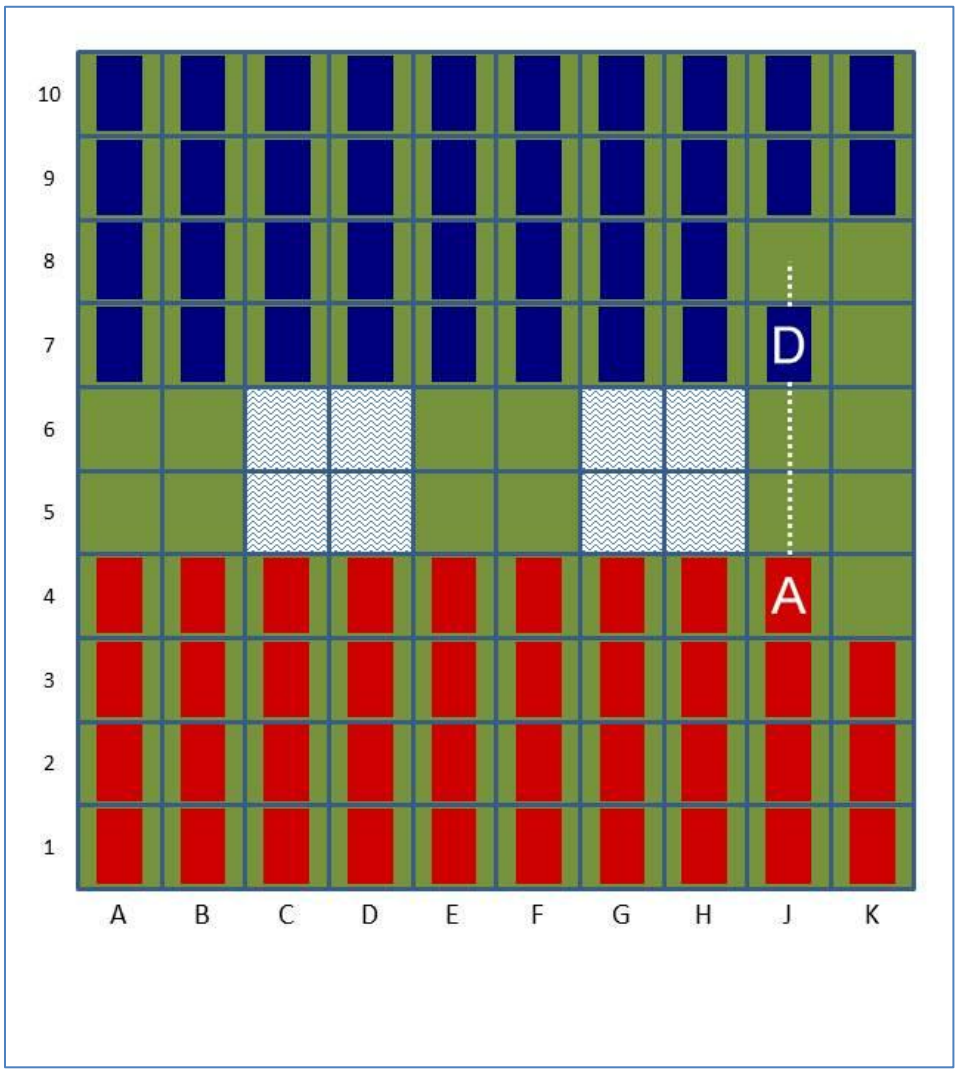
3.3.2 Access is possible by elimination of a blocking piece of the defenders side



Two blue pieces with low ranks on the same line do not present a problem. The colonel is able to get access to the major.

3.4 Open squares behind defender

3.4.1 An increase in mobility leads to an increase of the line



The defender can move the square J8. This increases the mobility of the defender so this square should be included in the line.

3.5 Actions by a look ahead function

A look ahead function has to detect the existence of a line between attacker and defender through which attacker can attain access to defender. Logically the detection process can be split up in 6 steps:

- Check 1: are attacker and defender on one line?
- Check 2: or stands attacker on line next to defender?
- Check 3: lead open squares behind defender to en extension of the line?
- Check 4: is the connection line accessible for attacker?
- Check 5: if pieces stand on the connection, are they removable?
- Check 6: is attacker able to attain the line in time?

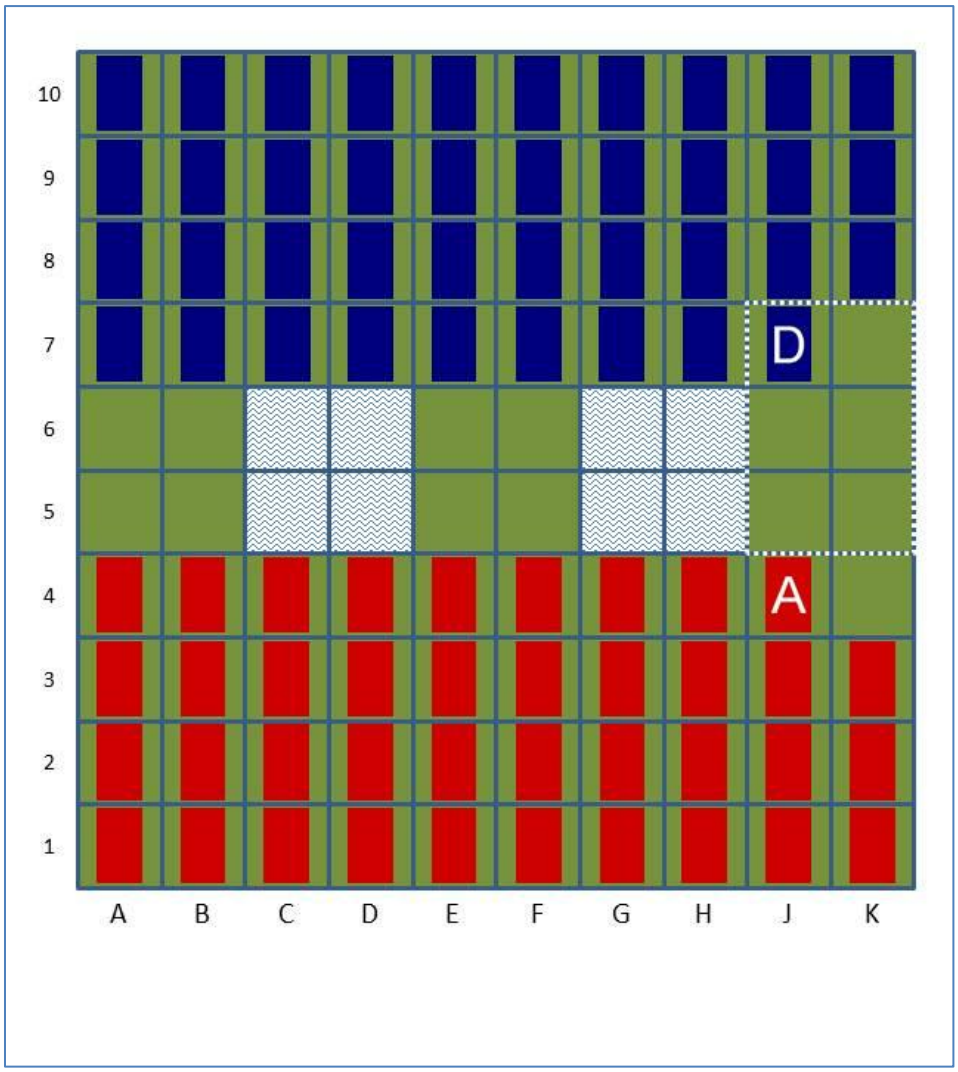
These steps can be programmed straightforward. This part of the look ahead function is achievable by relatively simple code. Efficiency is important. It depends strongly on the representation chosen for the board and pieces.

4 Step 2: Neighbouring lines may constitute a trap area for defender

Potential trap areas consist of two lines that restrict moves of the defender.

4.1 A neighbouring and open line exists

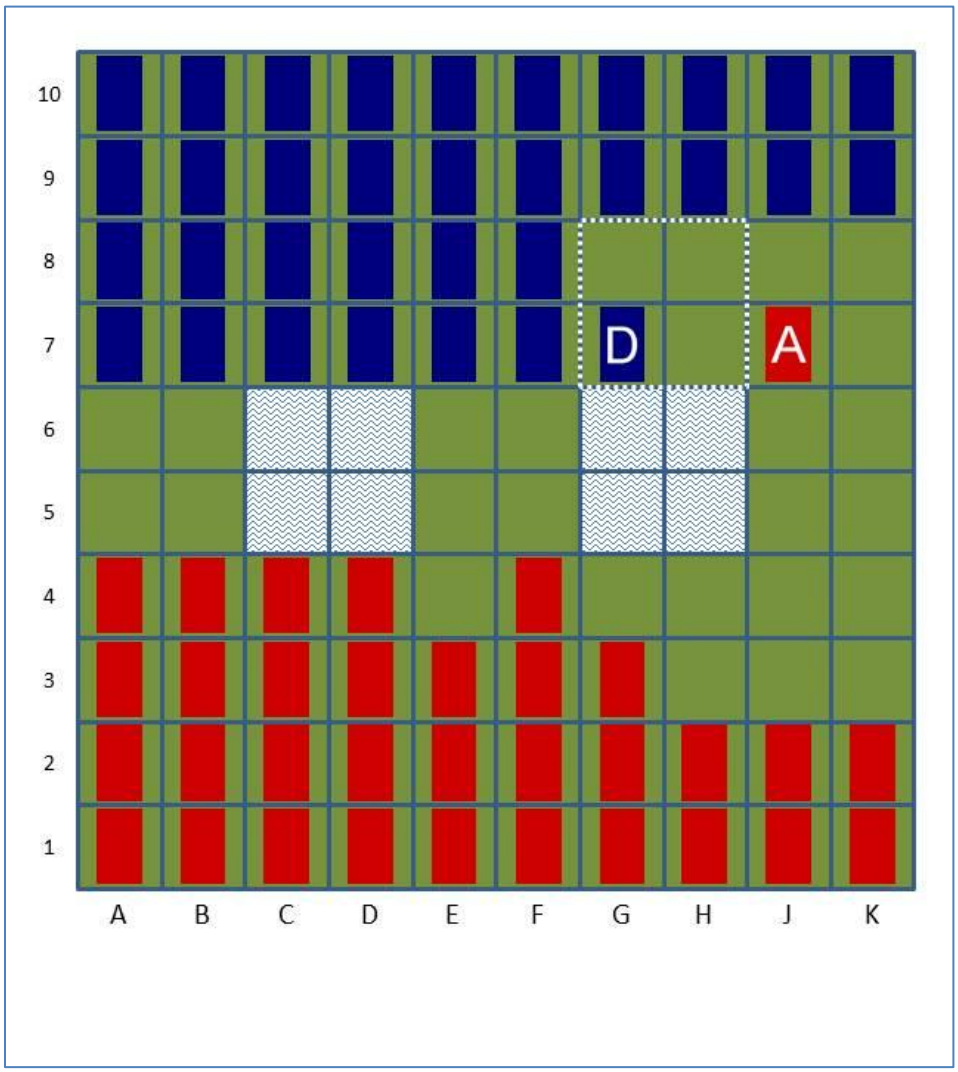
4.1.1 Example 1: attacker and defender stand on one line



This is the most simple example of capture by the two-squares rule. Blue is in turn.
The red attacker is able to capture the blue defender in 24 or less moves. The distance between attacker and defender influences the number of required moves. No distance between attacker and defender requires 8 moves, a distance of 1 square requires 16 moves, etc.
So if red is in turn then the capture of the blue defender will require 17 or less moves.

For blue it is important to recognise that the loss of the defender is inevitable. Sporting behaviour in accordance with the ISF rules requires that blue only once does a sideway move in such situations. The sideway move only serves to test the alertness and competence of red. If red does a sideway move too then blue should stop further senseless evasive moves.

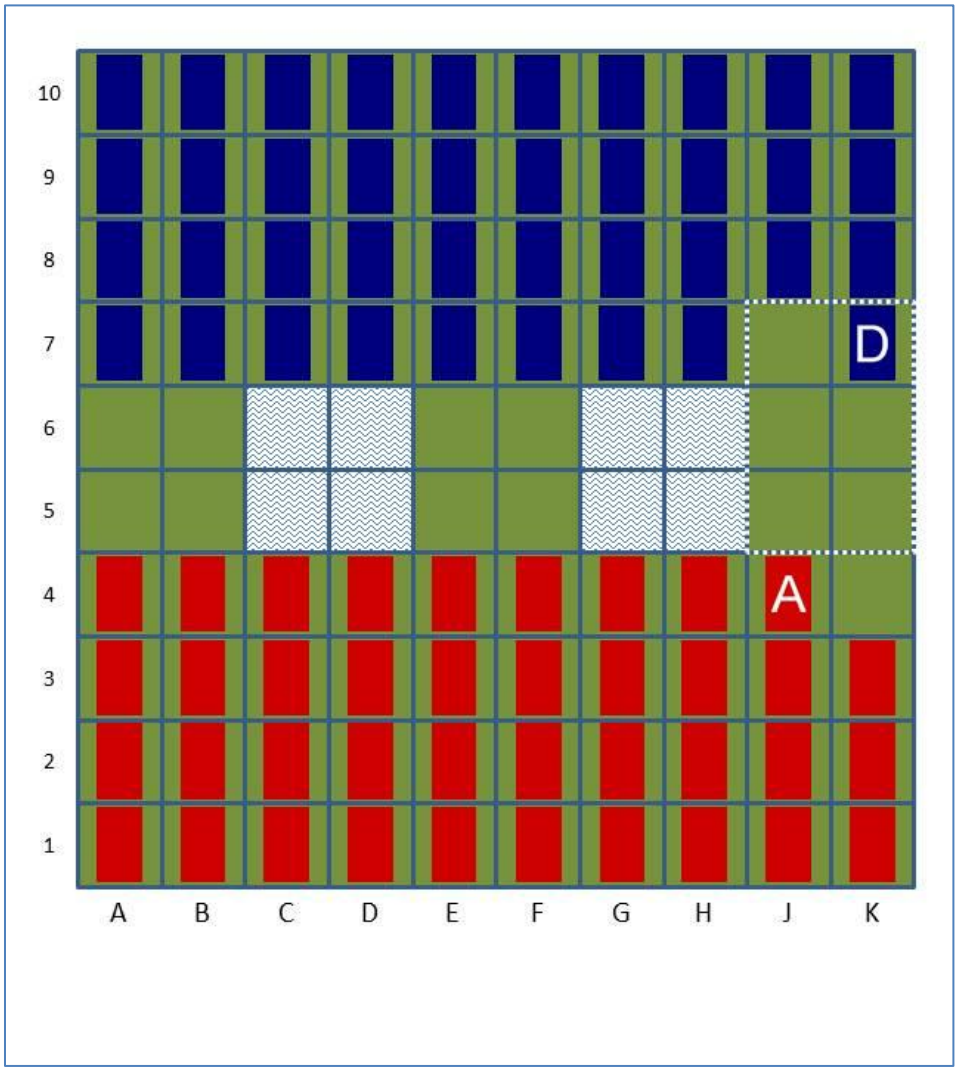
This is the most elementary condition. Attacker and defender stand on one line and the mobility of defender is restricted by a closed rectangular trap area with a width of two lines.



In this example a row instead of a column connects the positions of the red attacker and blue defender. So the attacker is able to capture the blue defender by use of the two-squares rule.

What is in force for vertical lines is in an analogue way in force for horizontal lines. In the following part of this document the choice has been made to discuss only examples with vertical lines between attacker and defender.

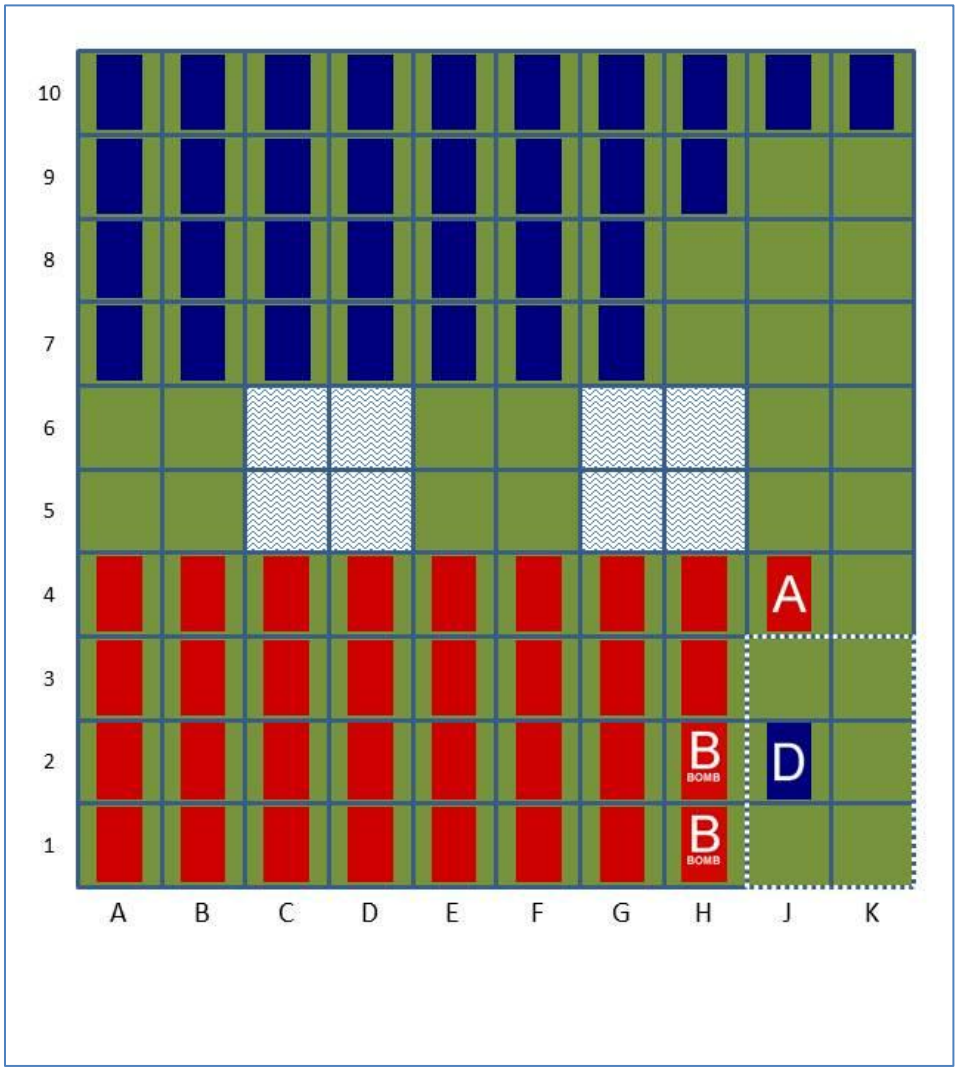
4.1.2 Example 2: attacker and defender stand on neighbouring lines



This example too represents the most simple condition for enforcement of the two-squares rule. Both sides have not moved sideways in a closed rectangular trap area with a width of two lines that restricts the mobility of the defender.
Here the two-squares rule prevents the capture of the blue defender.

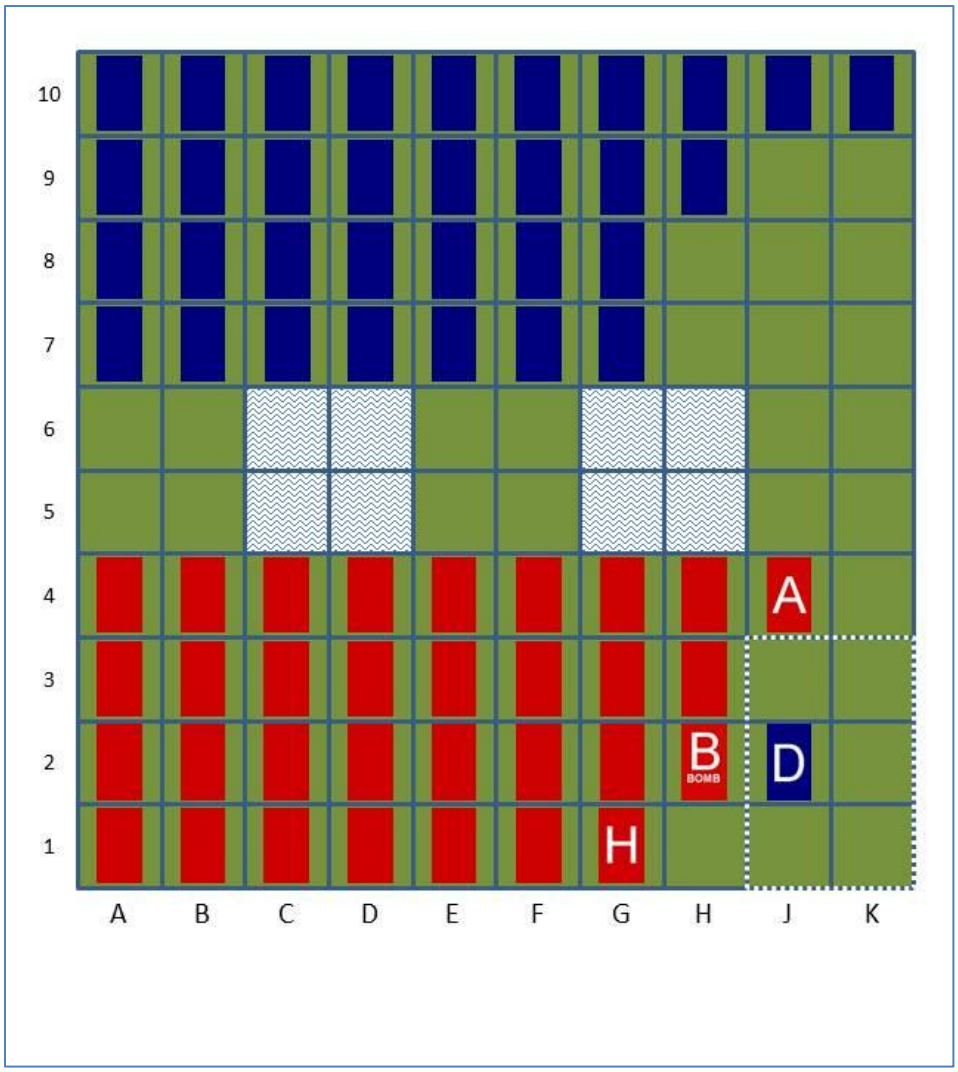
Sideway moves of the attacker do not make sense, but one sideway move may be done to test the alertness and competence of blue.

4.1.3 Example 3: The two-squares rule in the own half of the board



This kind of use for the two-squares rule is relatively rare. For a quick score in practical playing strength a programmer may decide to omit this part initially from the look ahead function and concentrate on other parts that occur more frequently. But a complete analysis should mention this possibility and its variations.

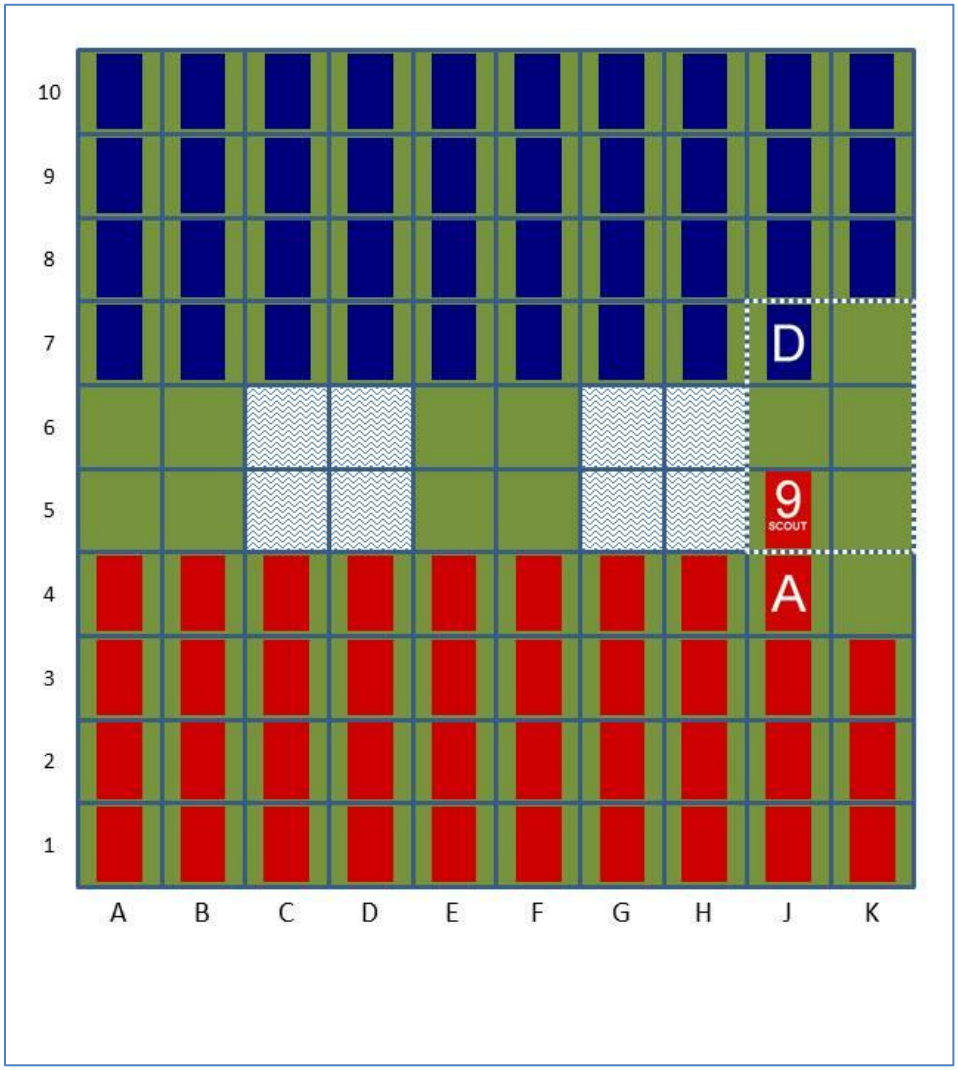
If defender is a miner then the bombs are no obstacle and then the two-squares rule is not in force.



An empty square in the border area may behave as a barrier for the defender if it lies next to an attacking piece with a higher rank than defender. But then it is not necessary to use a look ahead function that detects a capture by the two-squares rule; a look ahead function for encirclement will do the job too.

4.2 Special conditions in the candidate trap area

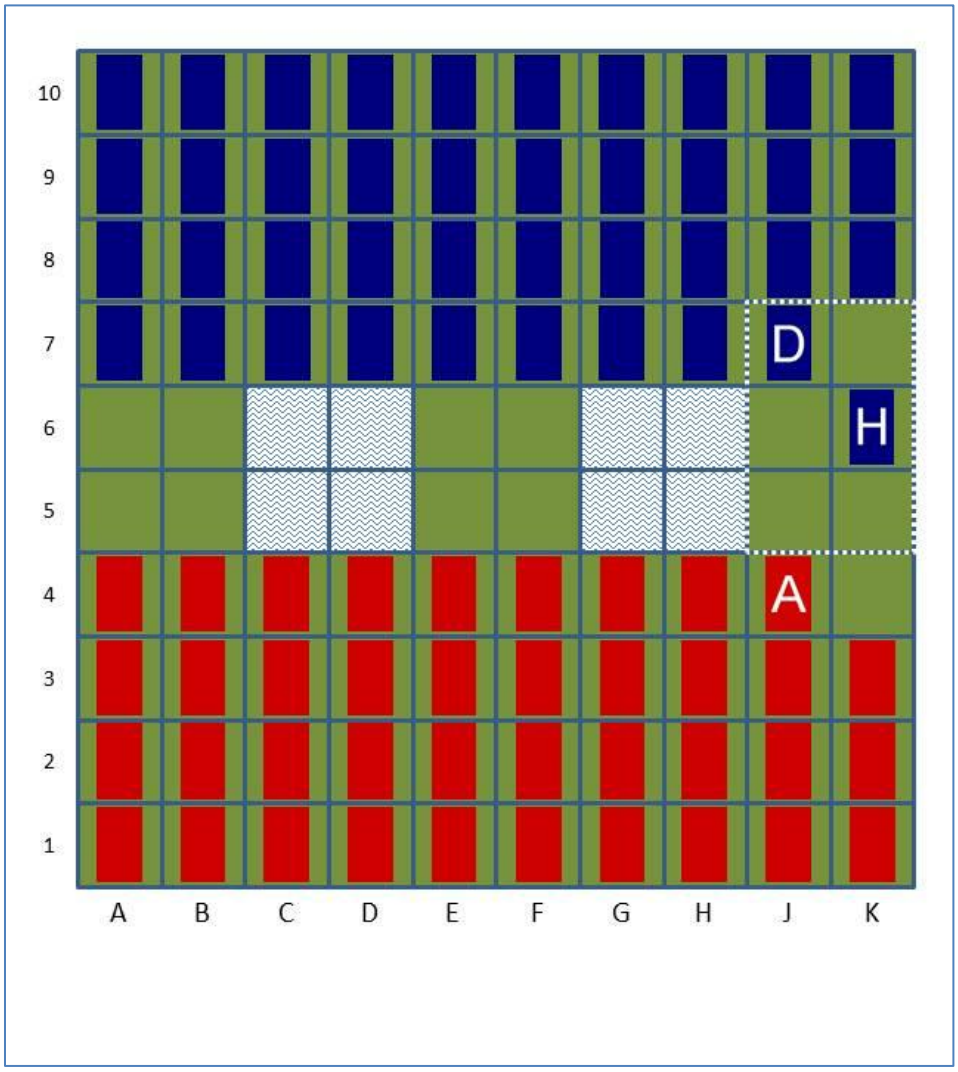
4.2.1 An own piece blocks access



The two-squares rule requires that all squares in the trap area between attacker and defender are free accessible for the attacker. If one of the squares is being occupied by an own piece then in most positions the two-squares rule cannot be in force. But there are exceptions. It may be possible to sacrifice the blocking piece in order to attain a position where the two-squares rule is in force.

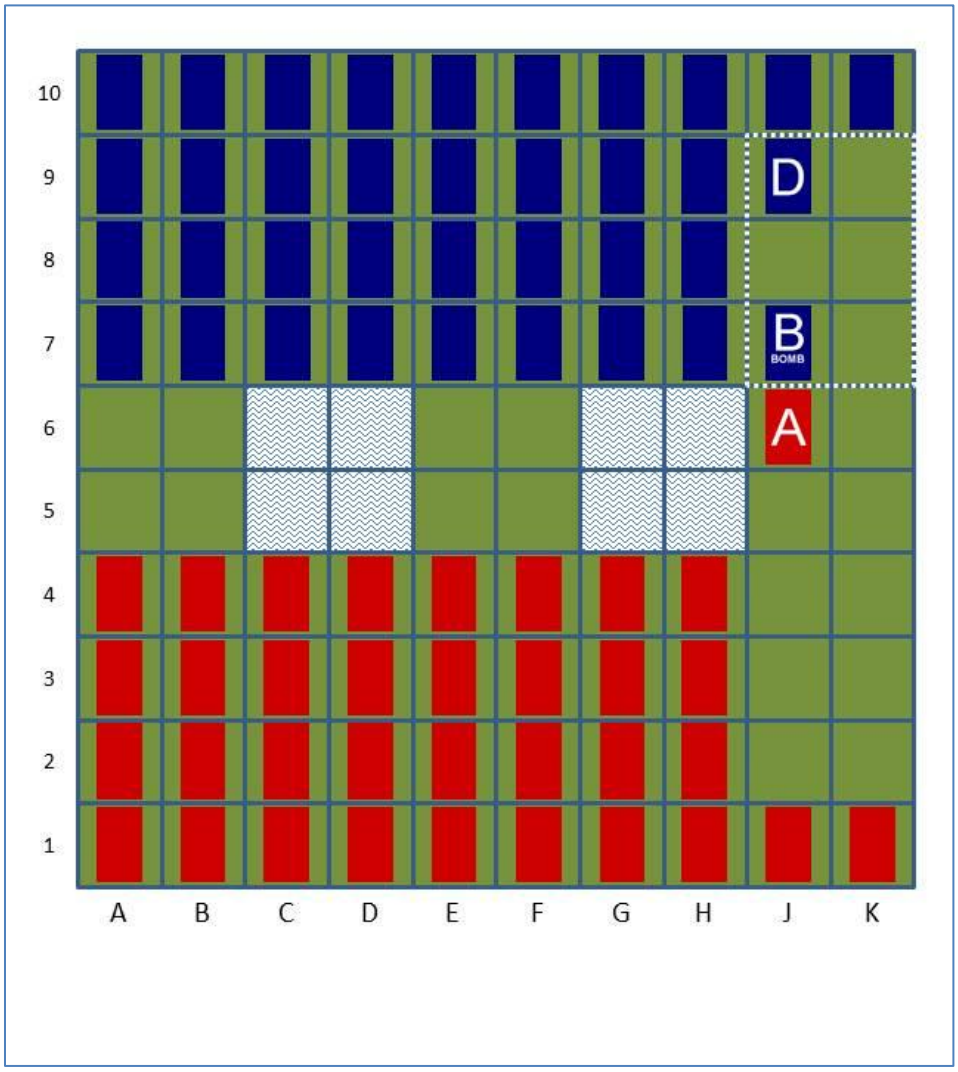
In this example the attacker is able to capture the defender if the scout on J5 does not block access to the defender.
Red may consider to sacrifice the scout on J7 and attain a position where the two-squares rule enables the attacker to capture the defender.

4.2.2 A piece of the opponent with a high rank blocks access



In this position a defender with a high rank effectively blocks access for the attacker. It is not possible to capture defender by use of the two-squares rule.

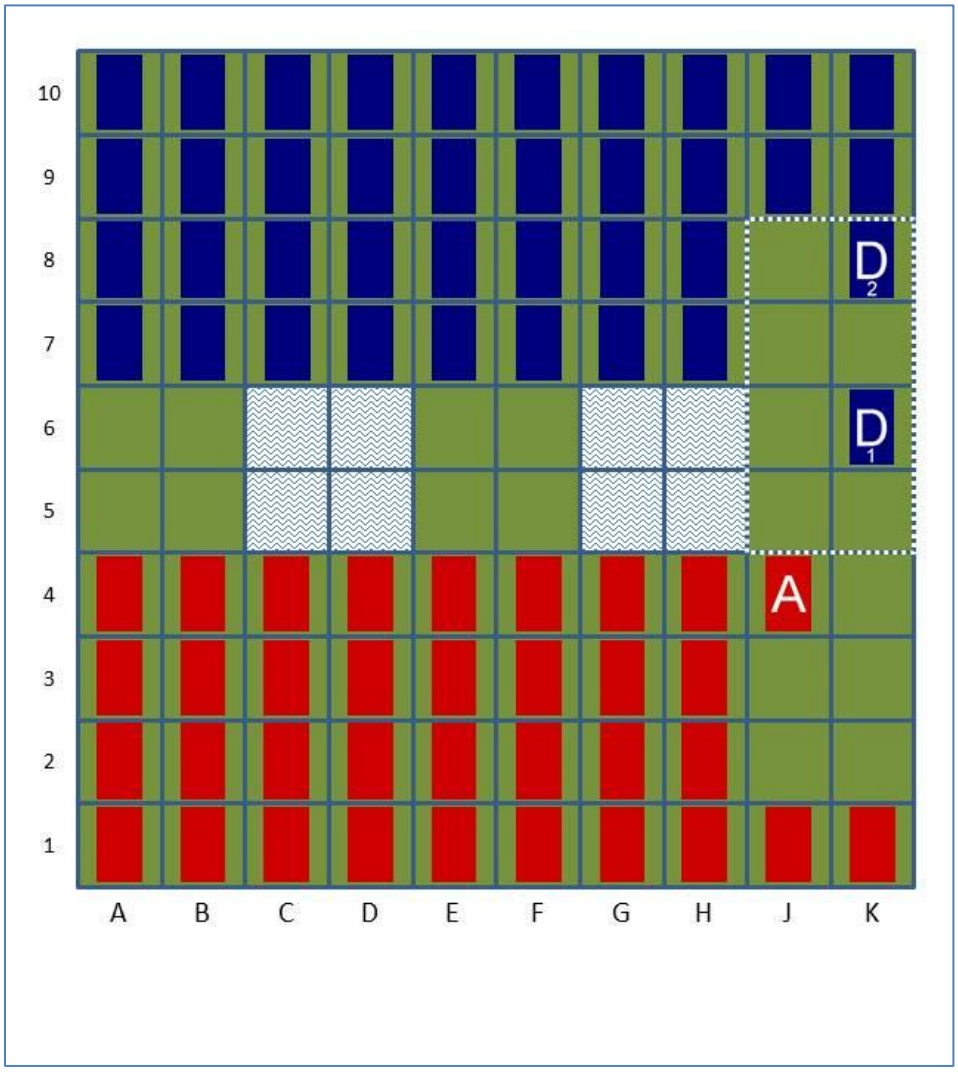
4.2.3 A bomb blocks access



Here a bomb blocks access to the defender. The two-squares rule is not in force.

Here an exception may be distinguished, but it is a rather impractical one. If the attacker is a miner then it may capture the bomb and theoretically the miner should be able to capture a scout on J9. But in an look ahead function this possibility can be omitted without any loss of effectiveness.

4.2.4 Two defenders in the trap area



The presence of two defenders in the trap area may be favourable. If both defenders have a lower rank than the attacker then it always is possible to capture one of them by use of the two-squares rule.

In this example the attacker is not in line with the two defenders. But it will be able to come in line with one of them and enforce a capture of one of them. Blue has to give up one of the pieces and normally will evade capture with the most valuable piece.

4.3 Actions by a look ahead function

A look ahead function has to check the two lines that are neighbouring the line between attacker and defender through which attacker can attain access to defender. Logically the detection process can be split up in 5 steps for each neighbouring line:

- Check 1: Does a line exist next to the line between attacker and defender?
- Check2: Has attacker access to the neighbouring line?
- Check 3: Is the length equal to the line between attacker and defender?
- Check 4: Is the connection line accessible for attacker?
- Check 5: if pieces stand on the connection, are they removable?

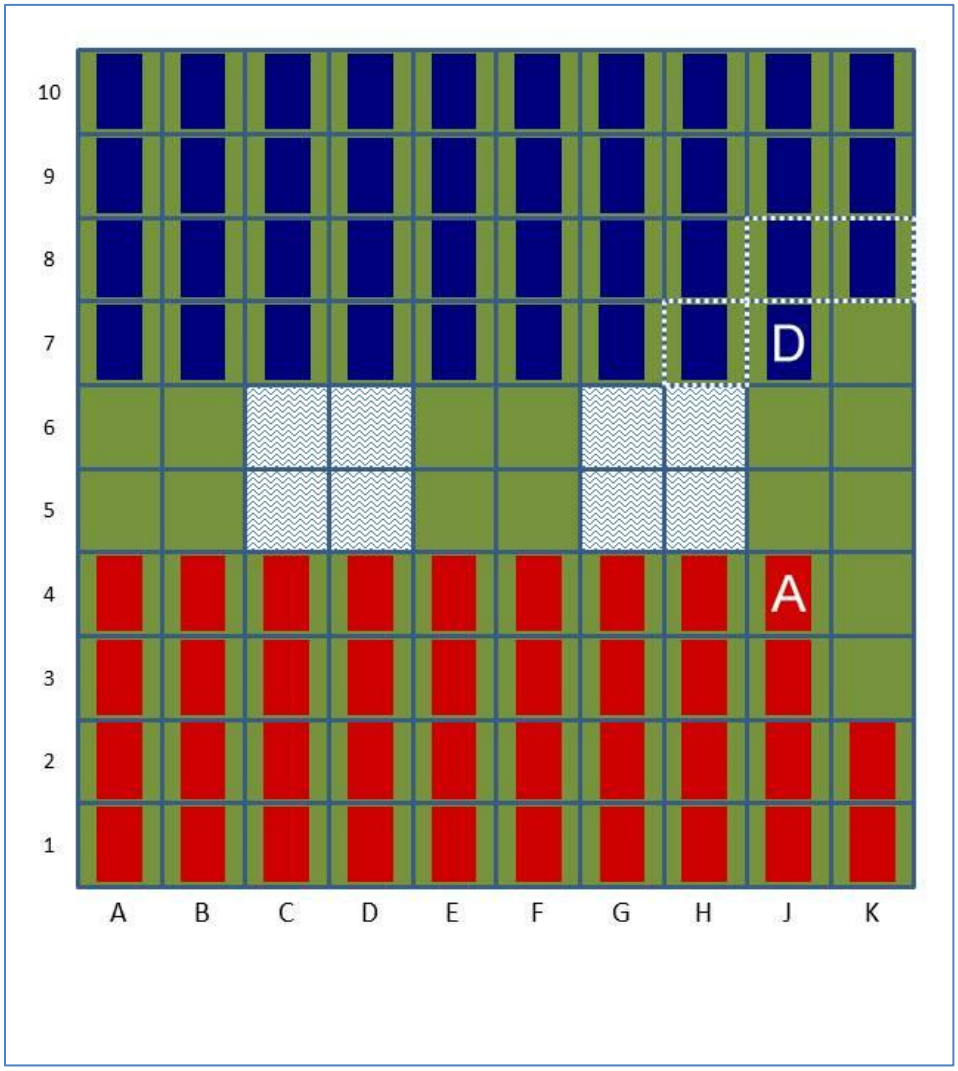
If two neighbouring lines exist and satisfy the conditions then these lines produce a rectangular or square area that may restrict the mobility of the defender and thereby enforce the two-squares rule. The two-squares rule only can be enforced if this candidate trap area has closed borders. It too may lead to a capture of the defender if eventually empty squares do not offer access to a safe position for the defender.

After the scan for open lines next to the connection line of attacker and defender the look ahead function should check the borders of candidate trap areas.

5 Are the borders of candidate trap closed?

5.1 Candidate trap areas with closed borders

5.1.1 Completely closed borders with unknown ranks in border squares

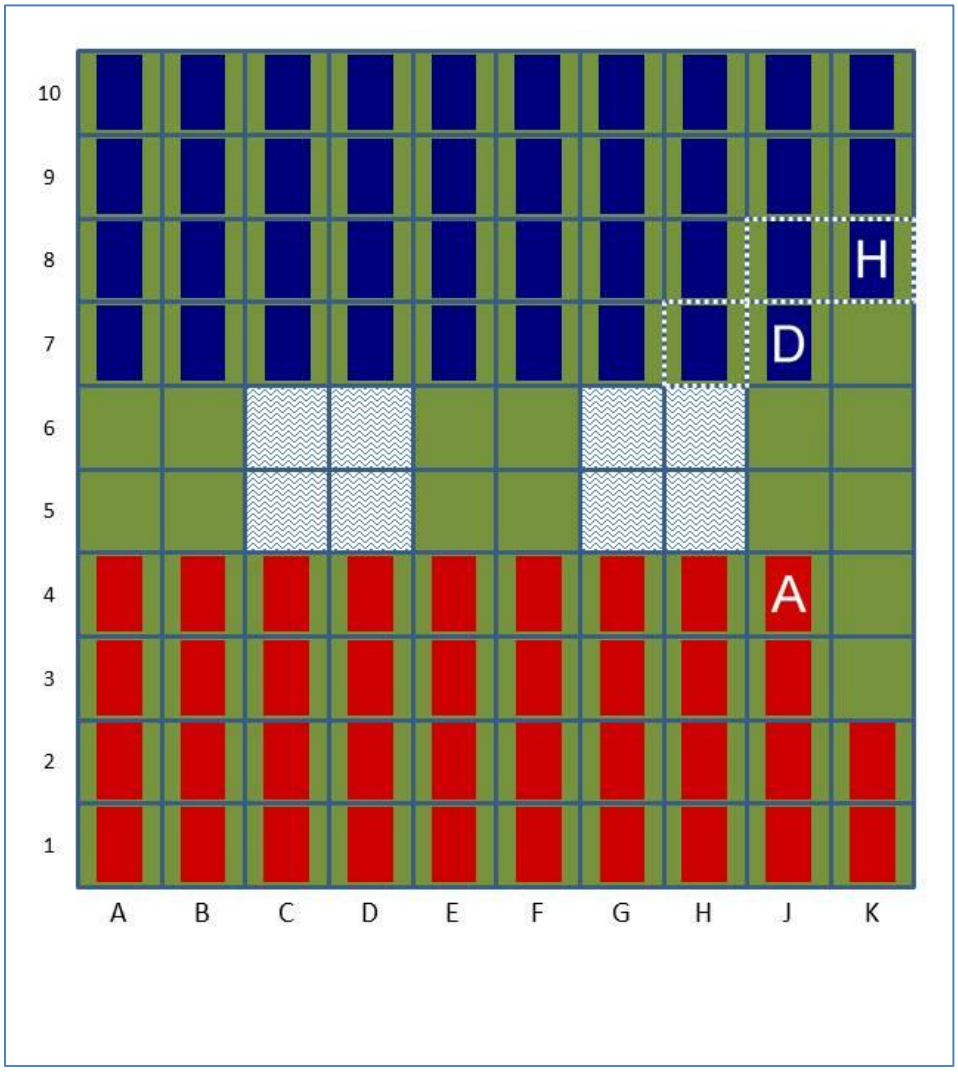


In this example the trap area has closed borders:

- The attacker prevents access to the first row
- Part of the left border lies in a lake
- The right border lies outside the board
- The remaining part of the border consists of squares occupied by pieces with an unknown rank.

If the unknown pieces on H7, J8 and K8 have a rank lower than or equal to the rank of the attacker then a capture of the defender will produce a material gain. If one of these pieces has a higher rank then the capture of the defender will lead to a capture of the attacker.
Probability should be taken into account. But this part of the decision making that consists of probabilities and values should not be done by the look ahead function. The decision to chase the defender has to be taken by a risk management function.

5.1.2 Completely closed borders with a known high rank in a border square



Border squares may contain a known high rank that has the capability to protect the defender.
If this occurs the defender may move to a square where it gets protection from the high rank. There the attacker will be lost after the capture of the defender.

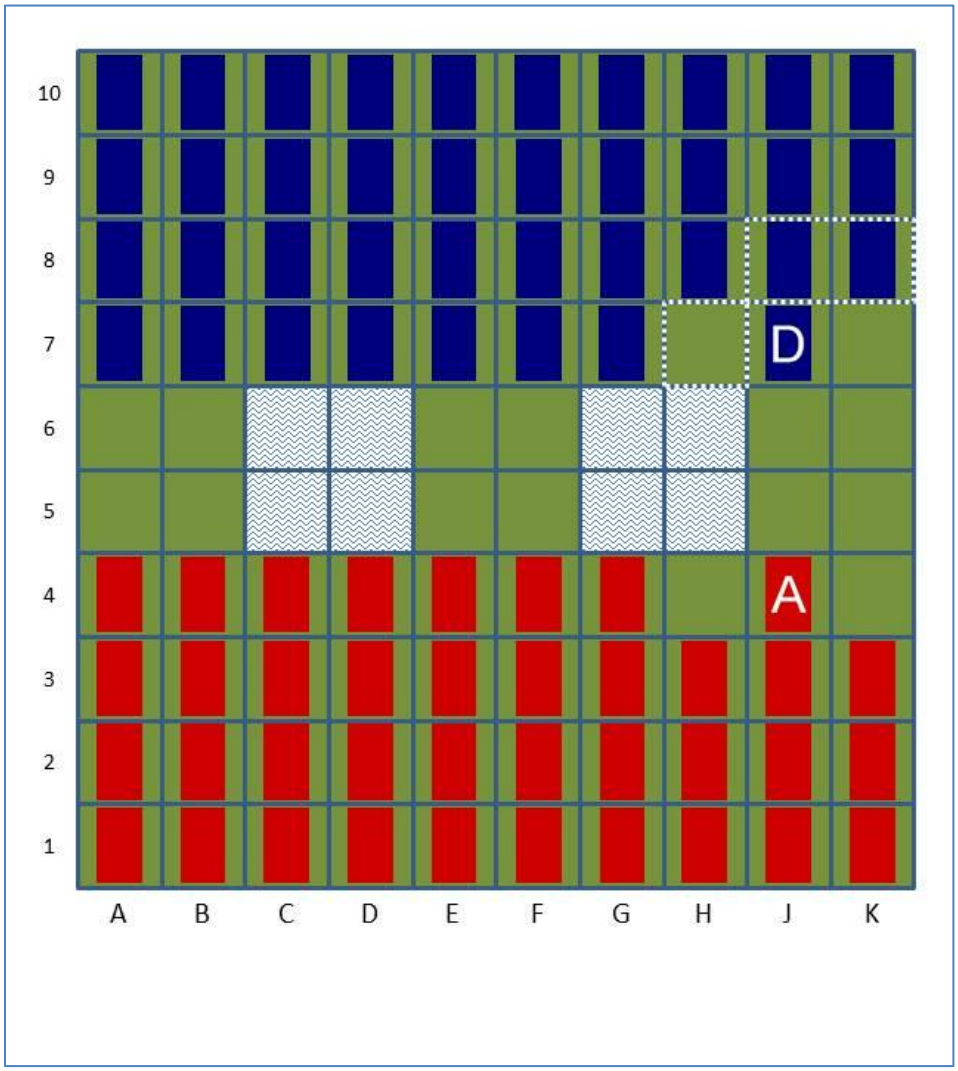
An attacker with a higher rank may have a value that is less than the value of the defender. Then the protection by a higher rank on this position is not effective and the defender will be captured.

But most exchanges do not give a favourable result. The look ahead routine should not take the decision about to chase or not to chase defender. The risk management function in the program should cooperate with the look ahead function by evaluating probabilities and long term consequences of material gain or loss.

5.2 Candidate trap areas with incompletely closed borders

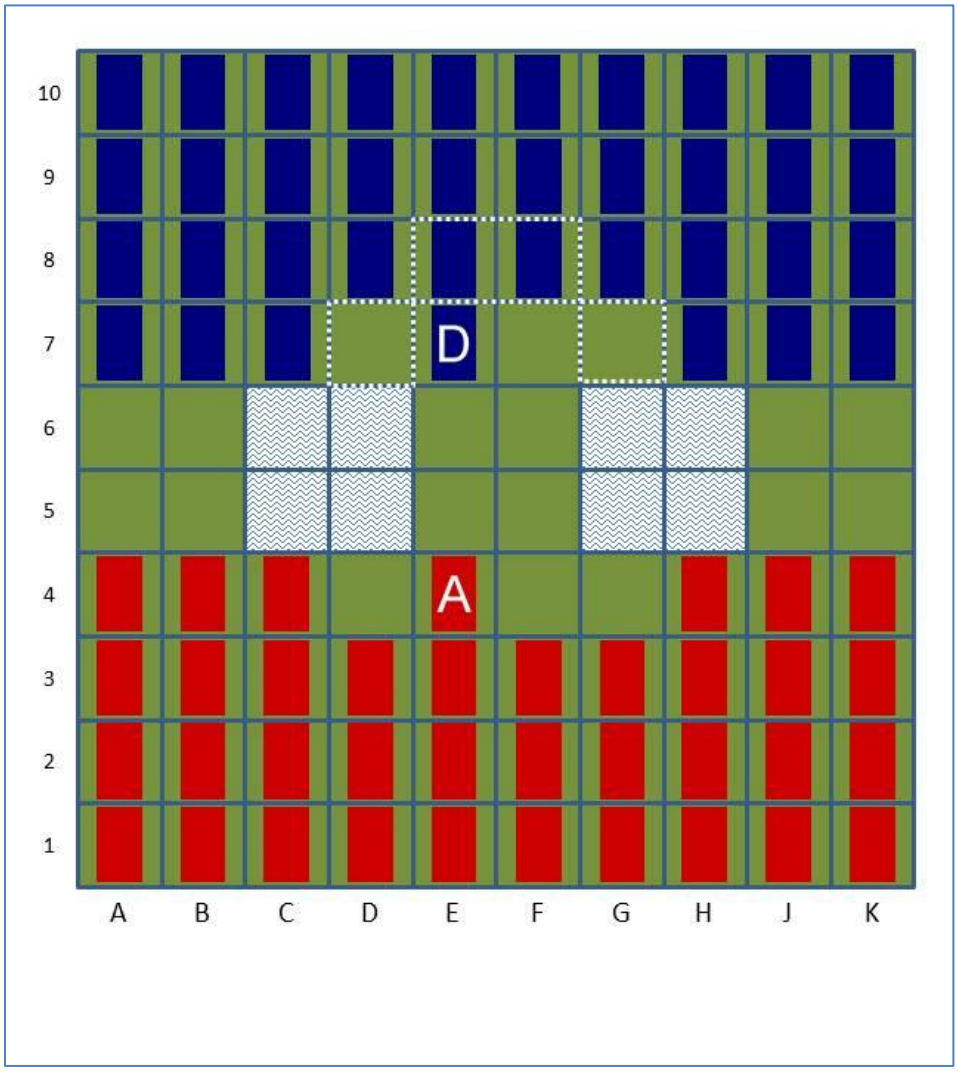
Until now the attacker and defender were situated in a trap area with closed borders. In many practical game positions one or more border squares are empty. Then a check is necessary for the possibility that the empty border square enables an escape. At first some trivial configurations are mentioned. Later the more complex variations will be analysed.

5.2.1 One empty border square with no escape



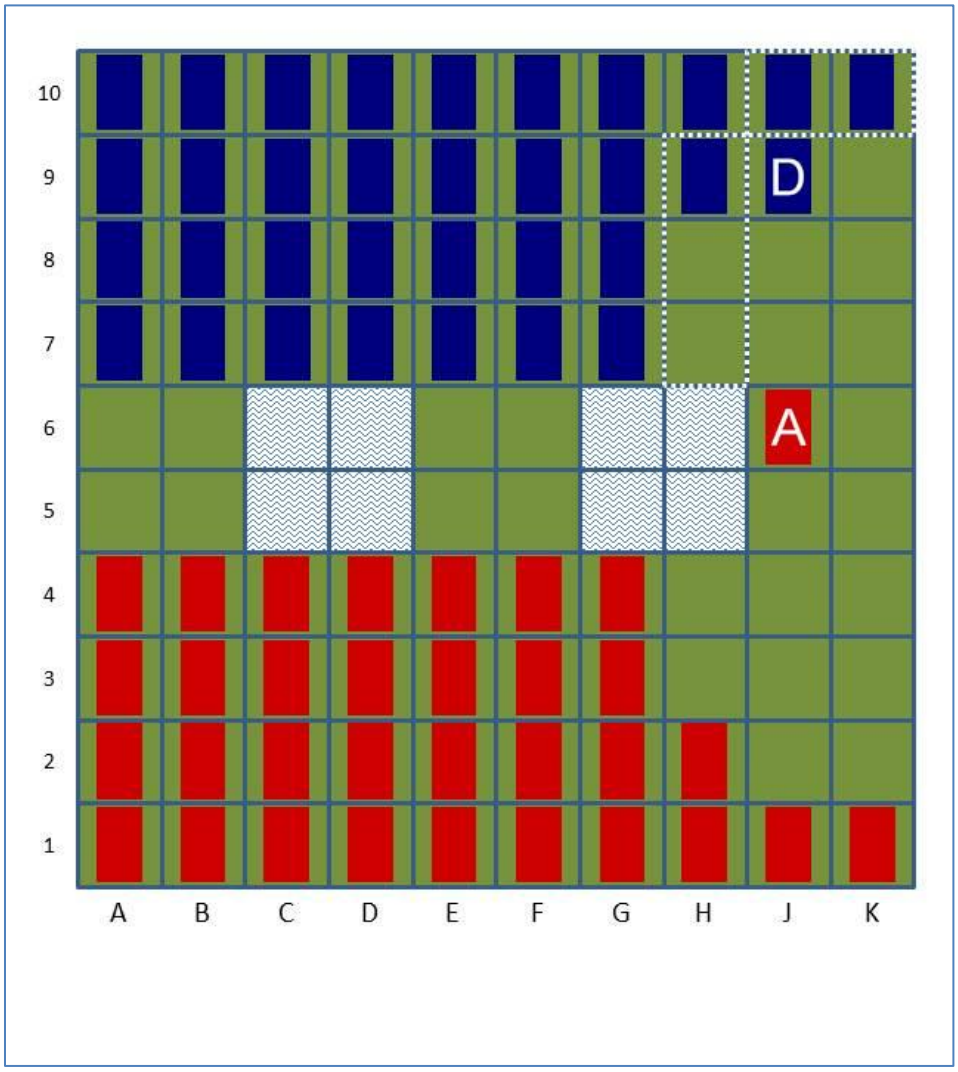
One square in the border of the trap area is empty. Because this square is being surrounded by blue pieces it does not present an escape for the defender.

5.2.2 Two separate empty border squares with no escape



Here the border of the trap area contains two empty and separate squares. None of these squares presents an escape for the defender.

5.2.3 Two neighbouring empty border squares



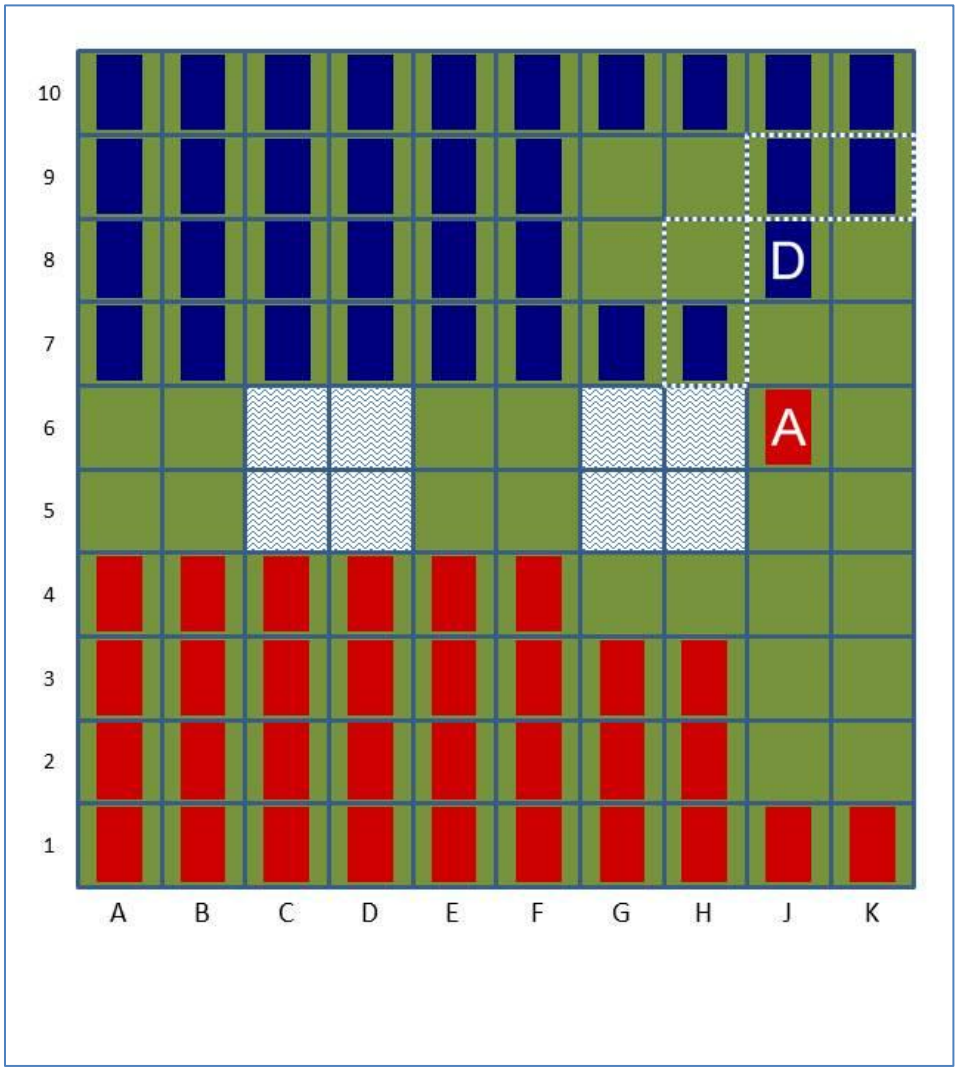
Two empty squares in the border of a trap area may offer an escape if they are neighbours.
In this example safety for the blue defender depends on which side is in turn.

If blue is in turn the move J9-J8 offers an escape for the defender. After this move it is not restricted to an area with a width of 2 lines, but stands in an area with a width of 3 lines. Then the two-squares rule cannot be in force.

If red is in turn then the move J6-J7 prevents the move J9-J8. Defender stays contained in a trap area with a width of two lines and will be captured.

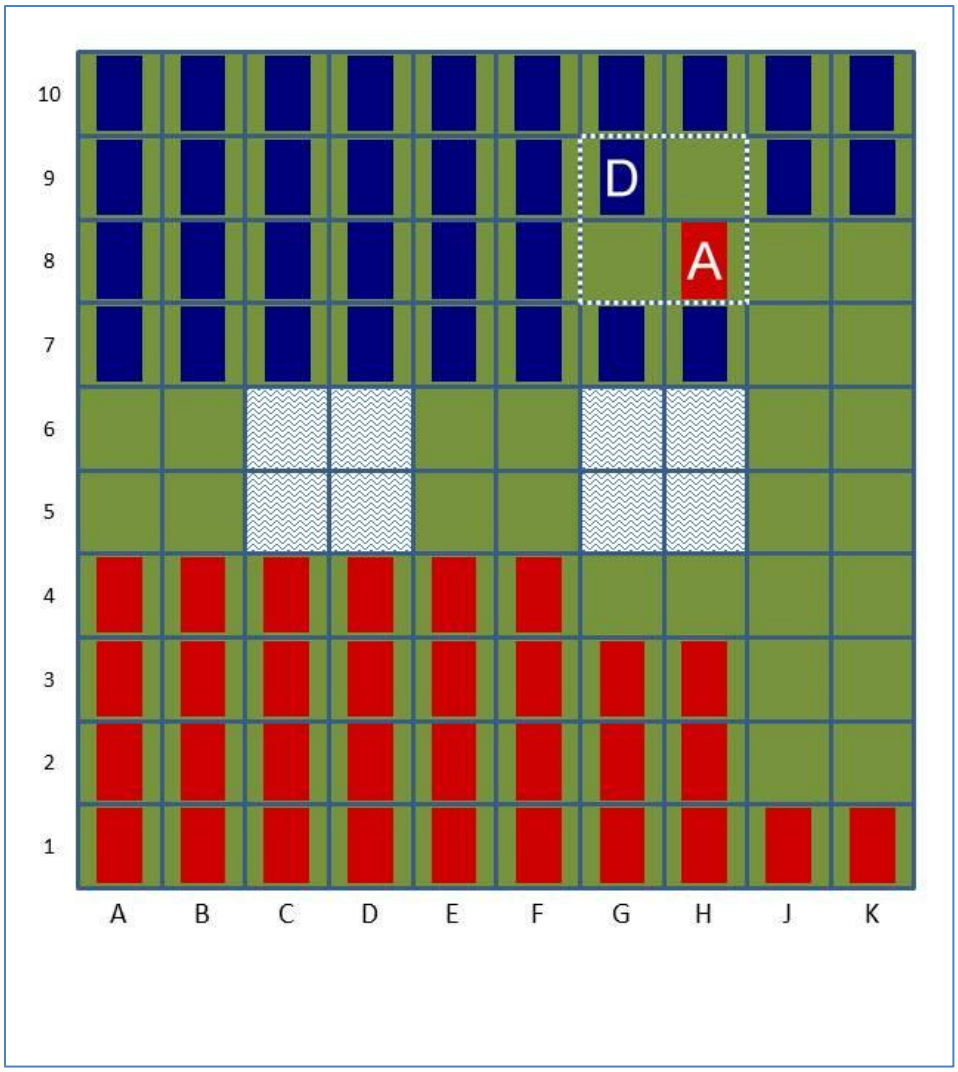
If the attacker stands on J5 instead of J6 then the defender always will be able to escape to the area with a width of 3 lines. So in this kind of position the distance of attacker and defender is relevant too.

5.2.4 An empty border square gives access to a 2 x 2 area



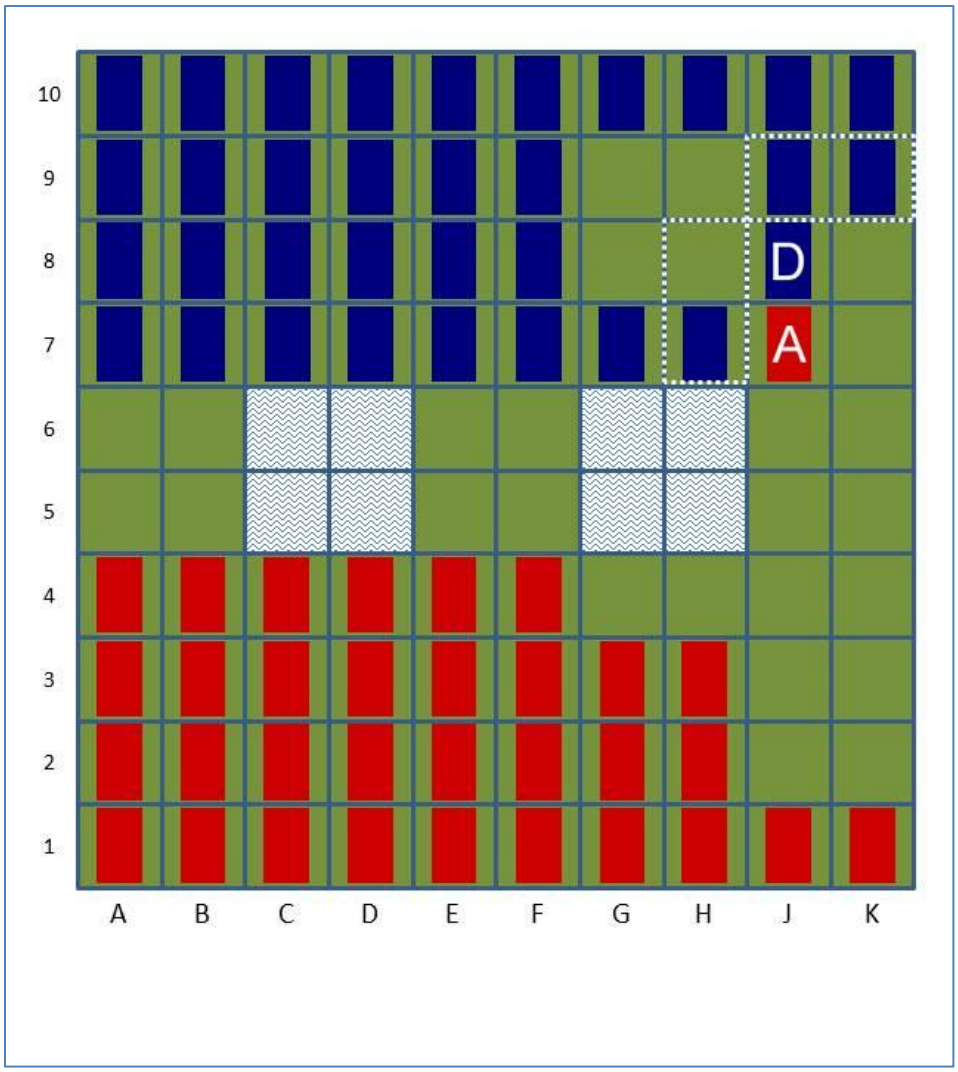
In this example the trap area has an empty border square on H8.
This square gives access to a 2 x 2 square area.
If blue is in turn then blue is able to retreat the blue defender to a safe spot in the 2 x 2 area:
J8-H8: defender moves to the left.
J6-J7: attacker tries to capture the defender.
H8-H9: defender keeps the required distance; H8-G8 will do the job too.
J7-J8: the last futile attempt to attain a winning position
H9-G9: this attains definitely a safe position
J8-H8: a move just to show what kind of position offers safety for blue.

This series of moves produces a position that is being shown on next page.



Attacker and defender have moved to a new trap area, but here defender cannot be captured and the two-squares rule prohibits an endless chase by red.

This position is exemplary for similar positions where defender cannot be captured. In most positions a 2 x 2 area suffices to attain safety for a defender if defender moves in time to a safe spot in the 2 x 2 area.

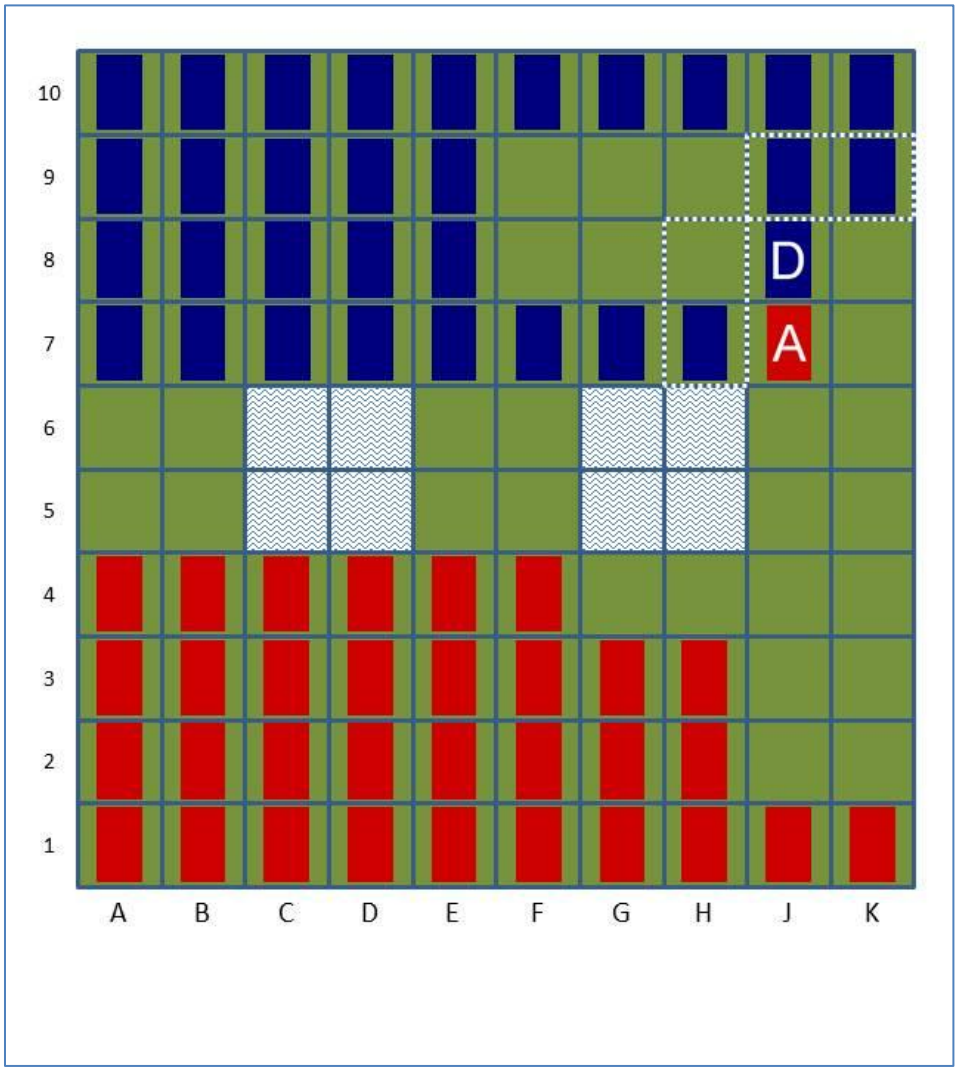


It may happen that a 2 x 2 area does not provide safety for a defender. If attacker is next to defender then defender is not able to attain in time a safe spot in the 2 x 2 area. Attacker is able to use the two-squares rule in the 2 x 2 area and will capture the defender.

This is the only condition that disables an escape in a 2 x 2 area for the defender. In all other conditions defender is able find a resort in the 2 x 2 area.

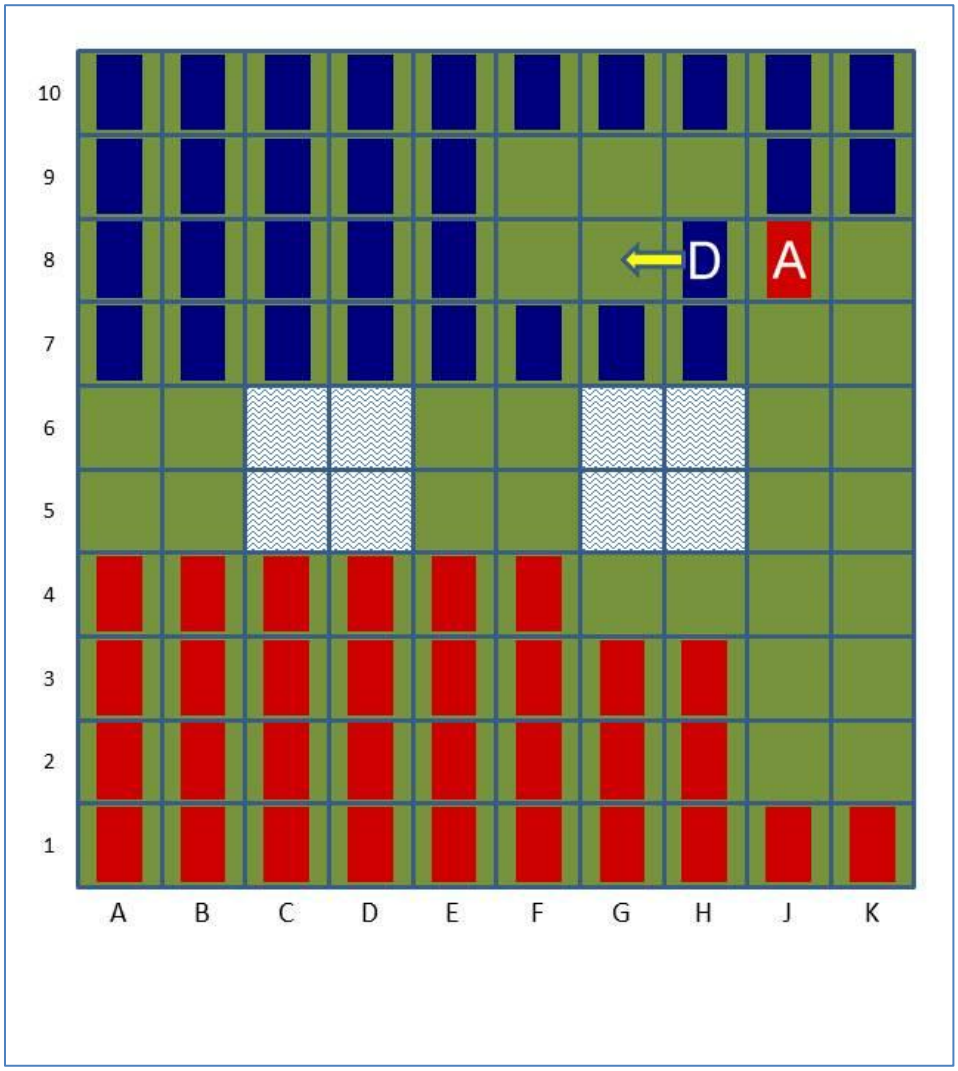
Just as in former examples of capture blue pieces may have a rank that enables the capture of the attacker. Paragraphs 5.1.1 and 5.1.2 mention considerations with regard to the ranks of known or unknown blue pieces, that may protect the blue defender. In a similar way such considerations are in force here too.

5.2.5 An empty square gives access to a 3 x 2 area

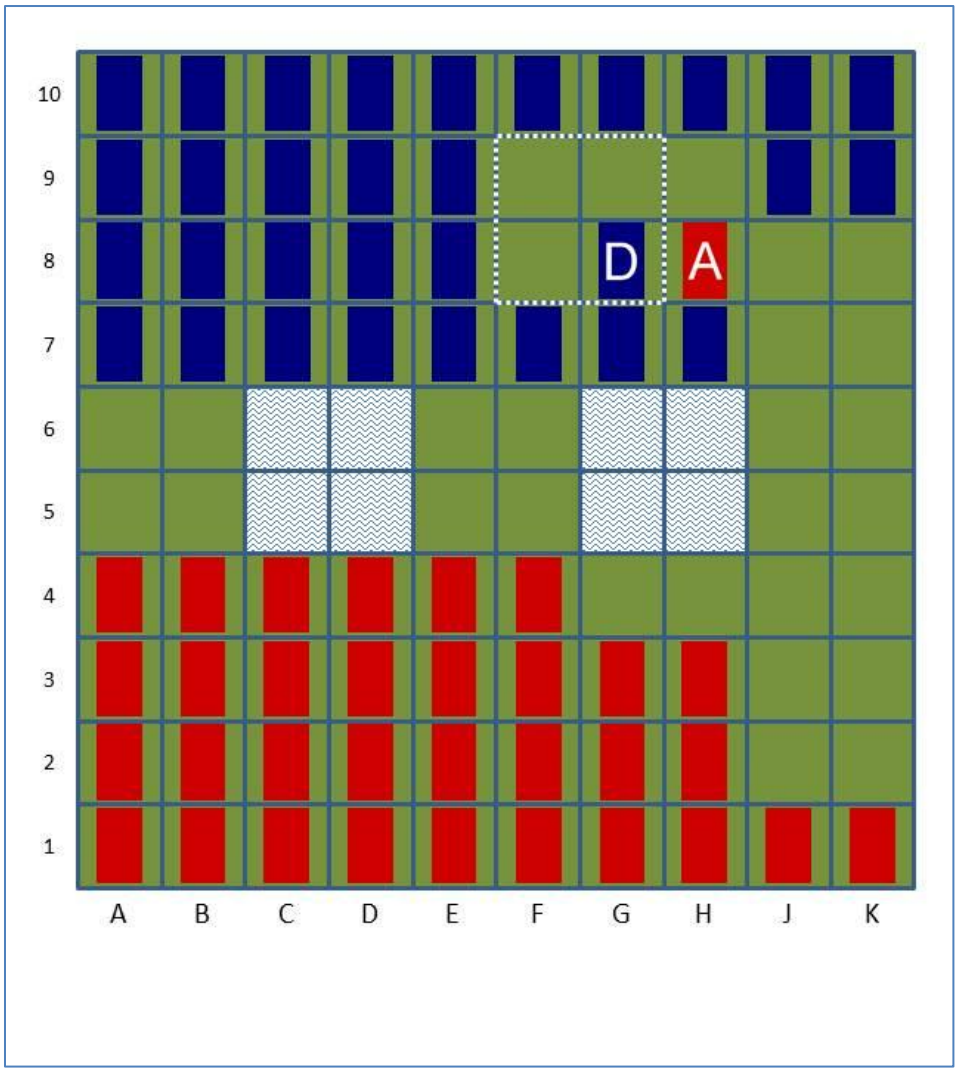


If attacker is next to defender then only a 3 x 2 area may enable an escape for defender.

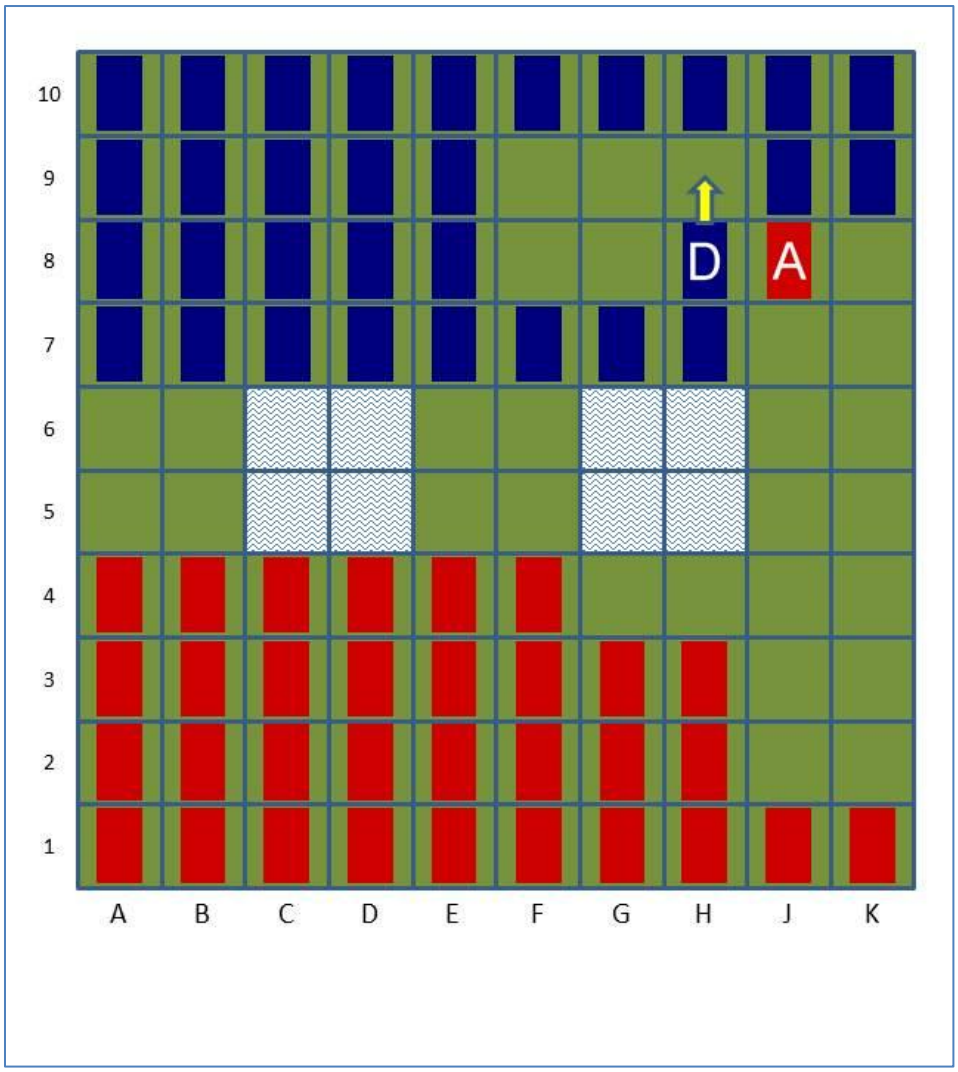
Defender has to move to H8 in order to evade a capture in this position.



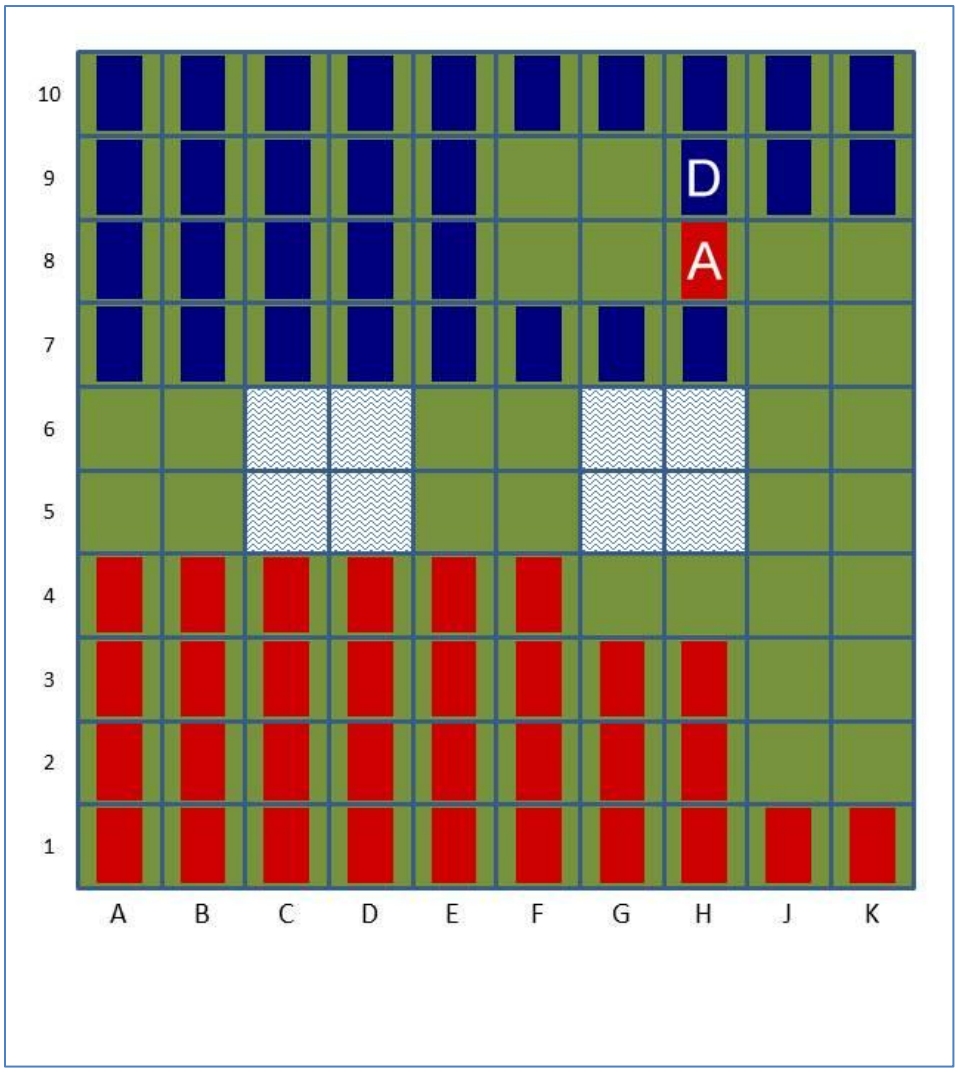
If attacker follows the defender then defender should not do the wrong move H8-G8.



By this move the defender achieves that the 3 x 2 area behaves as a trap area with a width of 2 lines. After this move attacker is able to use the two-squares rule and capture the defender.

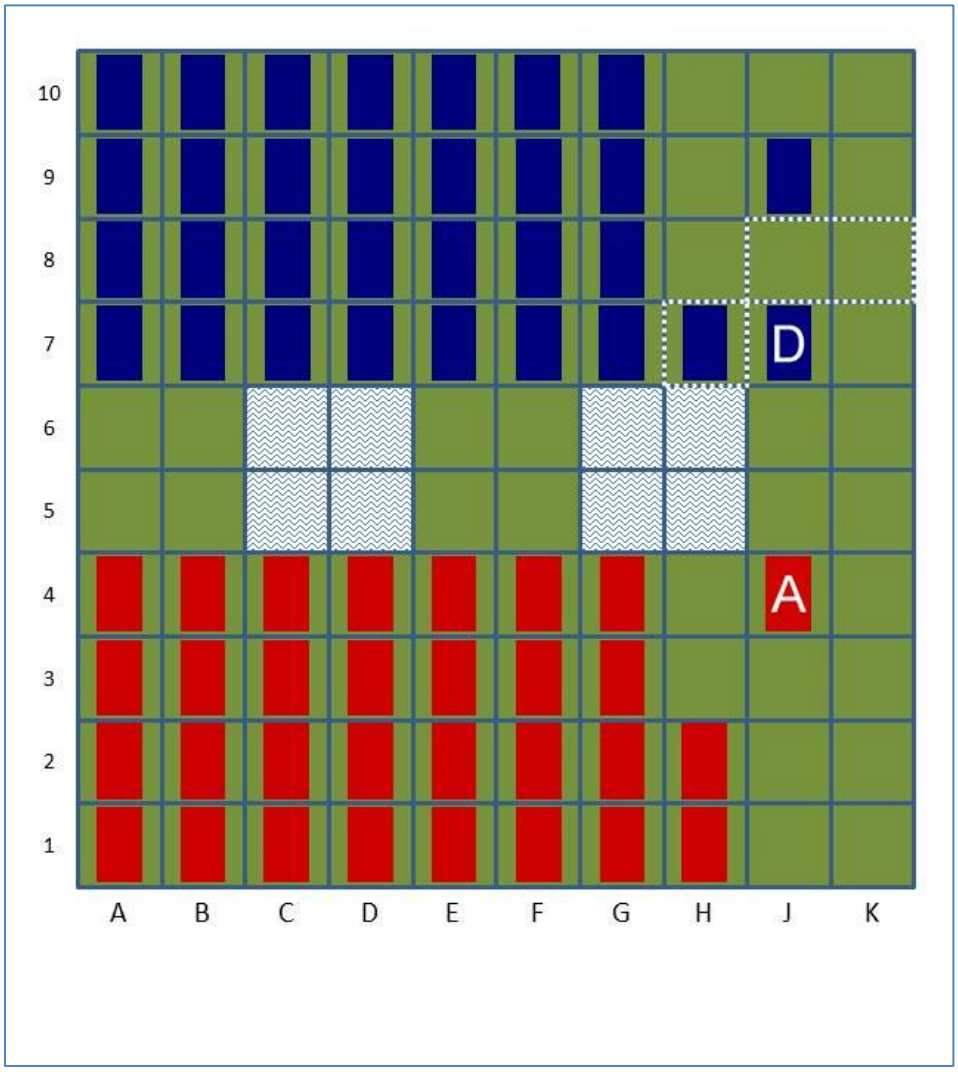


The right move is H8-H9.

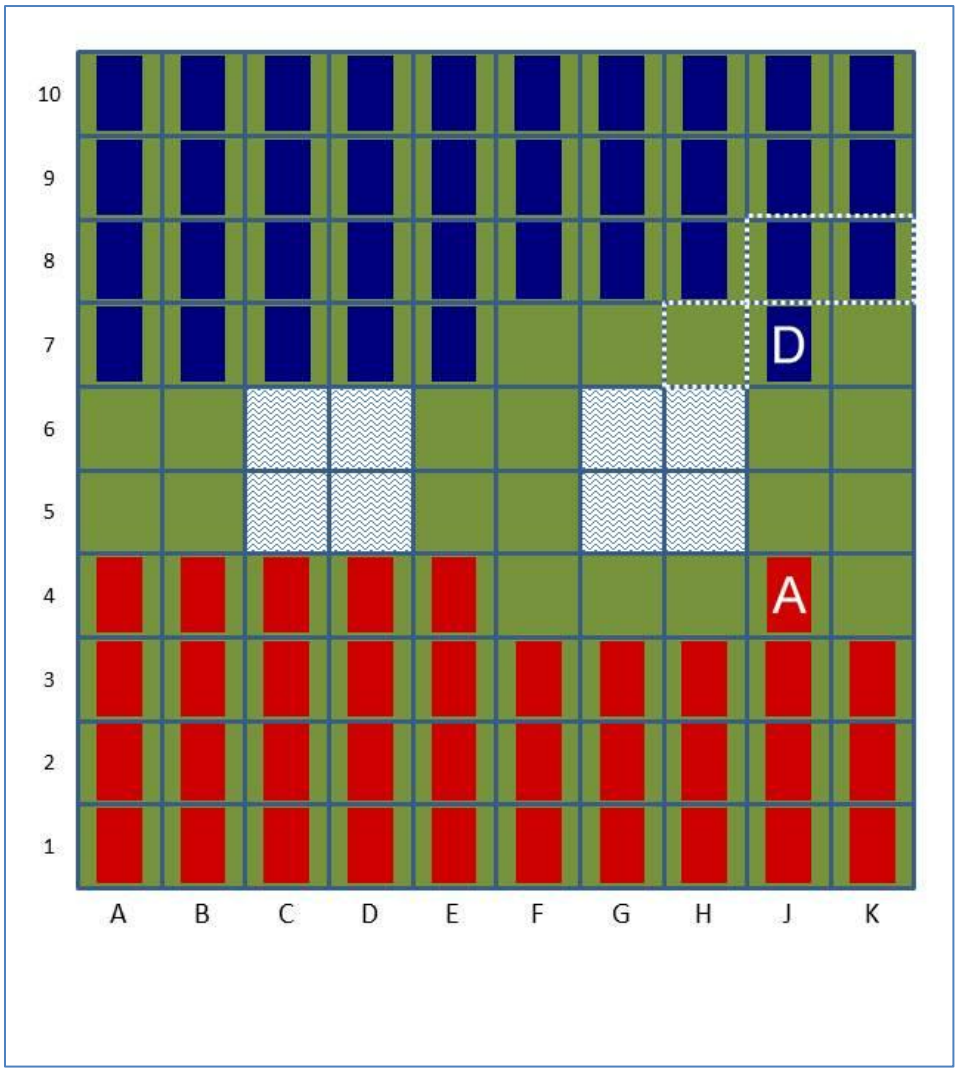


The defender now is able to do evading moves over the 3 squares F8, G8 and H8.
Attacker cannot capture the defender.

5.2.6 An empty square gives access to a circular path



Positions like this do not occur in many games, but they offer a trivial kind of escape for defender. So it is better to include them in the logic of a look ahead function.



The circular path may lead defender through the territory of the opponent.

In this example the blue defender may have to pass red pieces. If red attacks then the actions of blue will depend on what blue knows about the ranks of the red pieces.
This requires an extensive communication of the look ahead function with risk management.

5.3 Actions by a look ahead function

With regard to the coding of a look ahead function this part of the problem offers the most interesting challenges.

Starting point is the candidate trap area. It consists of two lines that contain the defender and are accessible by the attacker. The candidate area is a real trap area if its borders are not accessible for the defender. The candidate trap area is a real trap are too if border squares are accessible but do not lead to an escape for the defender. If a border square is occupied by a high rank then the high rank will offer protection to defender and the candidate area is not a trap area.

If two neighbouring lines exist and satisfy the conditions then these lines produce a rectangular or square area that may restrict the mobility of the defender and thereby enforce the two-squares rule. The two-squares rule only can be enforced if this candidate trap area has closed borders. It too may lead to a capture of the defender if eventually empty squares do not offer access to a safe position for the defender.

So the border squares – if they exist – have to be scanned. Any empty border square is the starting point for a search for other empty squares. During this search checks are necessary for the presence of 2 x 2 areas, 3 x 2 areas, circular paths or high ranked pieces that may protect the defender.

More specifically; the scan consists of 6 kinds of checks:
Check 1: Is the border square not accessible for the defender?
Check 2: Is the border square occupied by a high rank that gives protection?
Check 3: If accessible is the border square being surrounded by inaccessible squares?
Check 4: If accessible does the border square get protection by a high rank?
Check 5: Does an accessible border square lead to a 2 x 2 area?
    Additional sub check: stands attacker next to defender?
    If so: does the accessible border square lead to a 3 x 2 area?
Check 6: Does an accessible border square lead to an open circuit?

Technically there are many ways to cope with this challenge in program code. Efficiency is important, so a programmer should look for techniques in the programming language that promote efficiency in the processing of squares and paths on the game board.

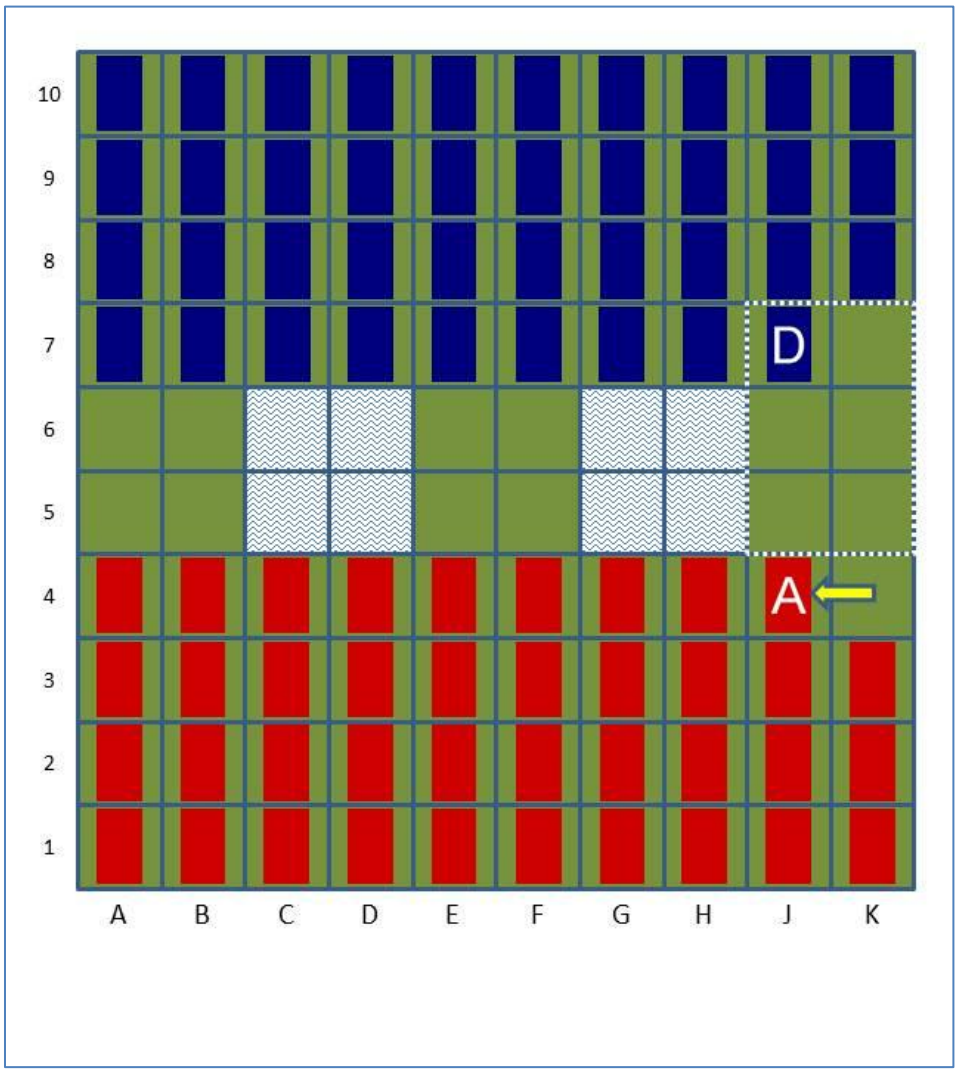
These checks definitely lead to a decision whether the candidate area is a real trap area or offers an escape.

If the area is a real trap area then a count of the sideway moves between the two lines will decide about gain or loss. See next chapter.

6 The count of sideways moves in a trap area

If the borders of a two line area are closed then the two-squares rule is in force.
A count of sideways moves on the 2 lines decides about capture or non-capture.

6.1 Defender and attacker stand on one line

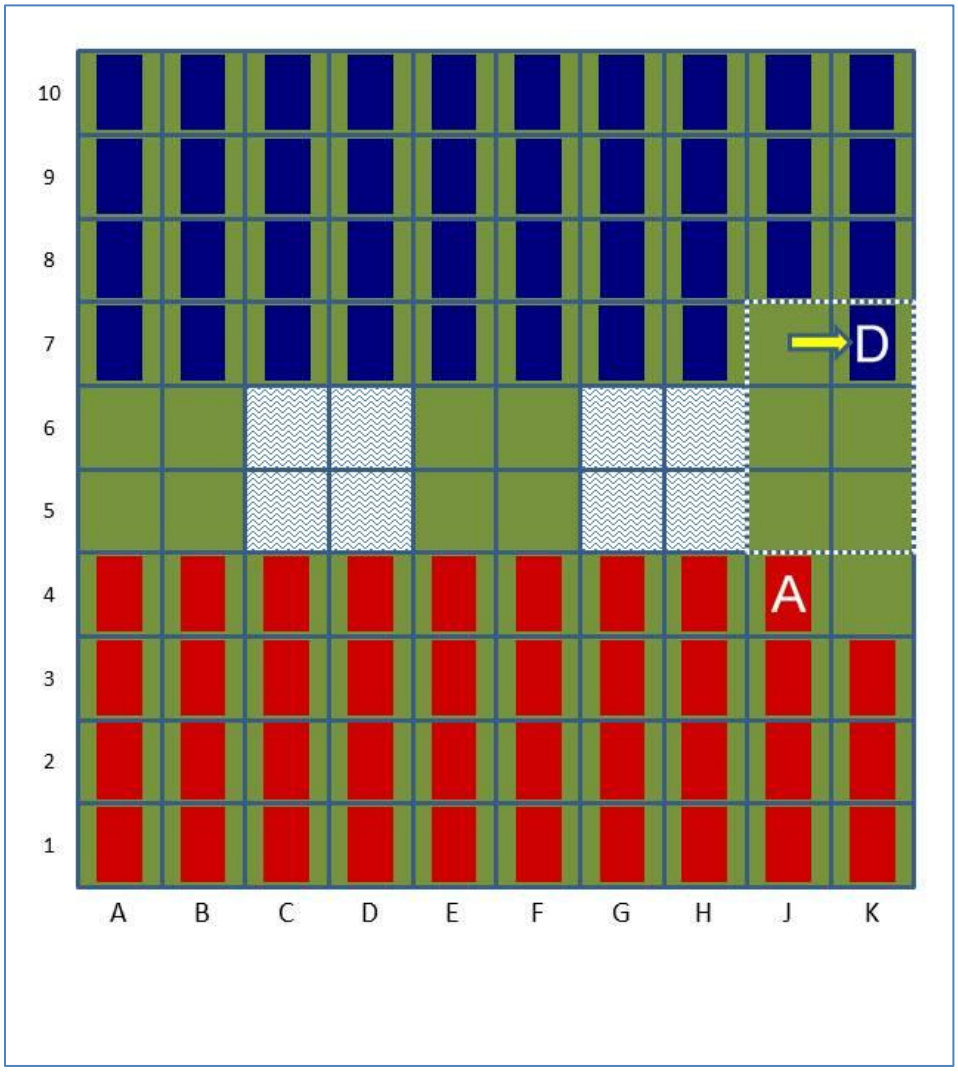


The last move of the attacker was a sideway move to the current position. A count should be made of the number of sideway moves between the two columns for both attacker and defender.

If the number of sideway moves of the attacker is equal to or less than the number of sideway moves of the defender then the attacker will capture the defender.

If the number of sideway moves of the defender in this position is less than the number of sideway moves of the attacker then the defender is able to evade a capture by the attacker.

6.2 Attacker and moved defender stand on neighbouring lines



The last move of the defender was a sideway move to the current position. A count should be made of the number of sideway moves between the two columns for both attacker and defender.

If the number of sideway moves of the attacker is less than the number of sideway moves of the defender then the attacker will capture the defender.

If the number of sideway moves of the defender in this position is equal to or less than the number of sideway moves of the attacker then the defender is able to evade a capture.

6.3 Actions by a look ahead function

After the complex scans in the previous paragraphs this is a trivial part of the look ahead function. Capture or escape can be computed exactly.

7 A model for capture by the two-squares rule

The goal of this this chapter is to offer a survey of the previous paragraphs by focussing on the essentials and eliminating the details. The first paragraph of this chapter contains a description. The second paragraph contains a scheme. In the third and last paragraph the communication with a risk management function is mentioned.

7.1 Actions to be taken by a look ahead function

Globally a capture of a defender may be detected in three steps:

1. Search for an open line that connects attacker and defender
2. Search for a trap area
3. Check border squares
4. Count the sideway moves.

7.1.1 Search for an open line that connects attacker and defender

Between attacker and defender an open connecting line exists.
Or attacker threats to attain a position that complies with this condition.

Paragraph 3.2.2 mentions a more complex case. This requires a more complex check. Because that kind of position seldom occurs it may be omitted without relevant consequences.

7.1.2 Search for a trap area.

Next to the connecting lines lie two lines. A scan of these lines enables the discovery of two trap areas that contains the defender and have no relevant obstacles for the attacker. It may be necessary to include squares behind the defender in the trap area.
Normally only a defender is contained in the trap area. Paragraph 4.2 mentions specific conditions that enable or disable a capture of defender if one additional piece is present in the trap area.

7.1.3 Check border squares

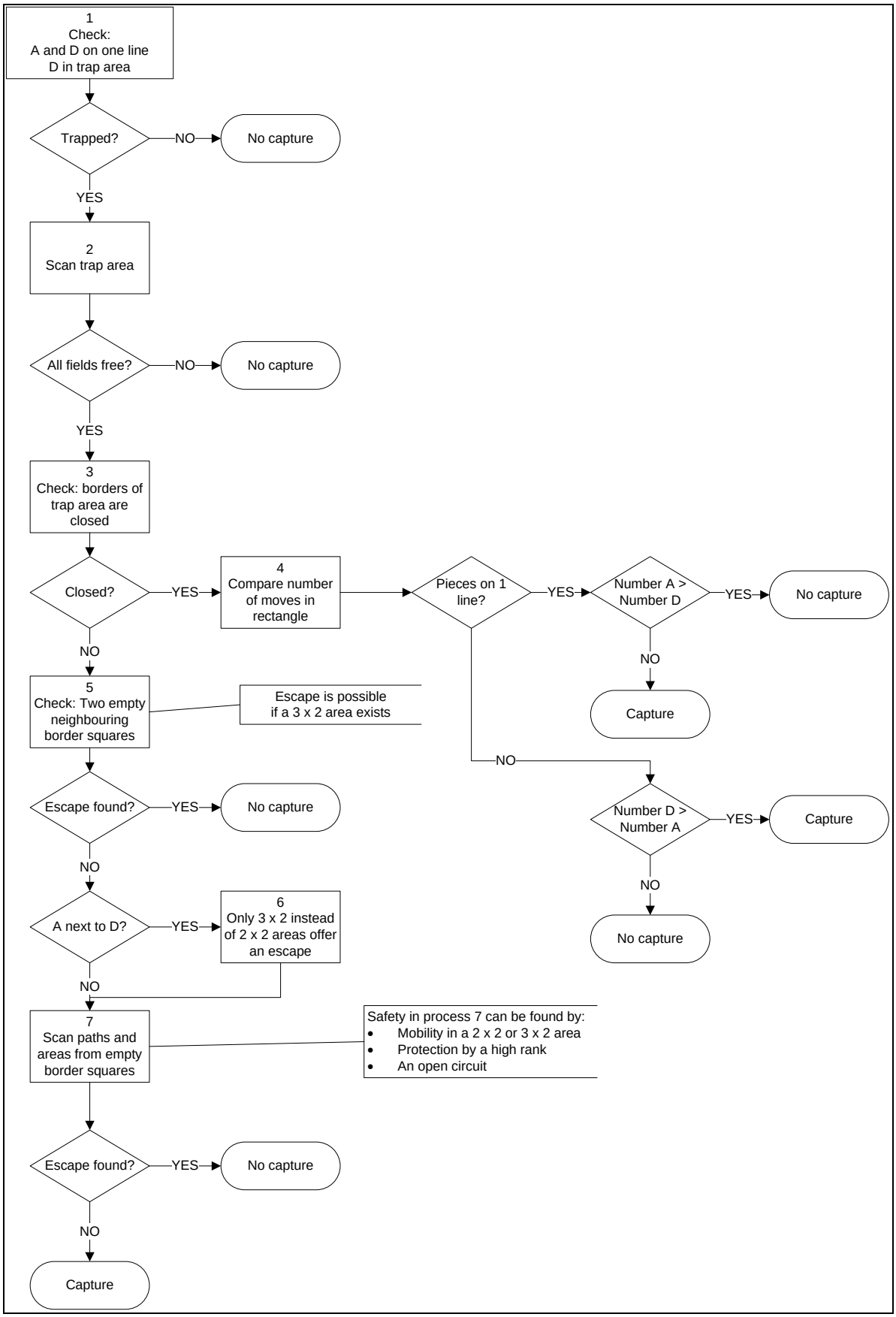
This is the most complex part of the detection of a capture. Possibilities:

- Separate border squares are empty but do not give access to new empty squares
- Two neighbouring border squares are empty and do not give access to new empty squares
  Depending on distance between attacker and defender a resort can or cannot be found in them
- A border square is occupied by a known piece that offers protection to defender
- A border square is empty and gives access to new empty squares
  The following conditions should be distinguished:
  - The neighbour of the empty border square may contain a rank that protects defender
  - The border square gives access to an open circuit
  - The border square gives access to an empty 3 x 2 area
  - The border square gives access to an empty 2 x 2 area and attacker stands next to defender
  - The border square gives access to an empty 2 x 2 area and attacker does not stand next to defender.

7.1.4 Count sideway moves

The scheme on next paragraph shows the rules for a capture.

7.2 A scheme


1
Check:
A and D on one line
D in trap area
Trapped?
NO
No capture
YES
2
Scan trap area
All fields free?
NO
No capture
YES
3
Check: borders of trap area are closed
Closed?
YES
4
Compare number of moves in rectangle
Pieces on 1 line?
YES
Number A > Number D
YES
No capture
NO
Capture
NO
5
Check: Two empty neighbouring border squares
Escape is possible if a 3 x 2 area exists
Escape found?
YES
No capture
NO
Number D > Number A
YES
Capture
NO
No capture
A next to D?
YES
6
Only 3 x 2 instead of 2 x 2 areas offer an escape
NO
7
Scan paths and areas from empty border squares
Safety in process 7 can be found by:
Mobility in a 2 x 2 or 3 x 2 area
Protection by a high rank
An open circuit
Escape found?
YES
No capture
NO
Capture

7.3 The environment of the look ahead function

A look ahead function that detects a capture by the two-squares rule communicates with other functions in a Stratego program:

1. Tactical decision making uses this look ahead function in local search trees
2. The search tree for the definite choice of a best move uses this look ahead function
3. The look ahead function communicates with the risk management function.

In (local) search trees the look ahead function may lead to substantial cut-offs and gives reasonably reliable values to the resulting end nodes. The look ahead function requires that a preliminary scan detects candidate attackers and defenders in a current game position.

The communication towards risk management consists of:

- A list of unknown pieces of the defenders side that may offer protection to the defender
- A list of pieces of the attackers side that may constitute a barrier for escape to the defender.

Risk management evaluates the possibilities and long term consequences and returns an instruction to enter or abandon the chase for the defender.

For a more elaborate discussion of decision making in Stratego programs, its structure and functional components refer to the article “Starting points for improvement in Stratego programming”.

8 Conclusion

In this article a model has been presented for the capture or non-capture of pieces by the two-squares rule. The presentation is by example. Probably the model covers all relevant cases in practical game positions, but the author has no absolute certainty about its completeness. So it is possible that this article will lead to further amendments or even a more theoretically elaborated model. Comment is welcome.

The potential of this model is large. Its realisation in a look ahead function certainly will enable a significant improvement in playing strength. It too may lead to a decrease in senseless chases by a program. But probably a look ahead function alone is not enough to prevent horizon effects; an effective quiescence algorithm may be required to end that kind of behaviour definitely.

Appendix A: A demonstration of the algorithm

A.1 Select a candidate defender and attacker

Until now the look ahead function has started with a position where attacker and defender are known. But somehow in a game position pairs of attacker and defender have to be detected.

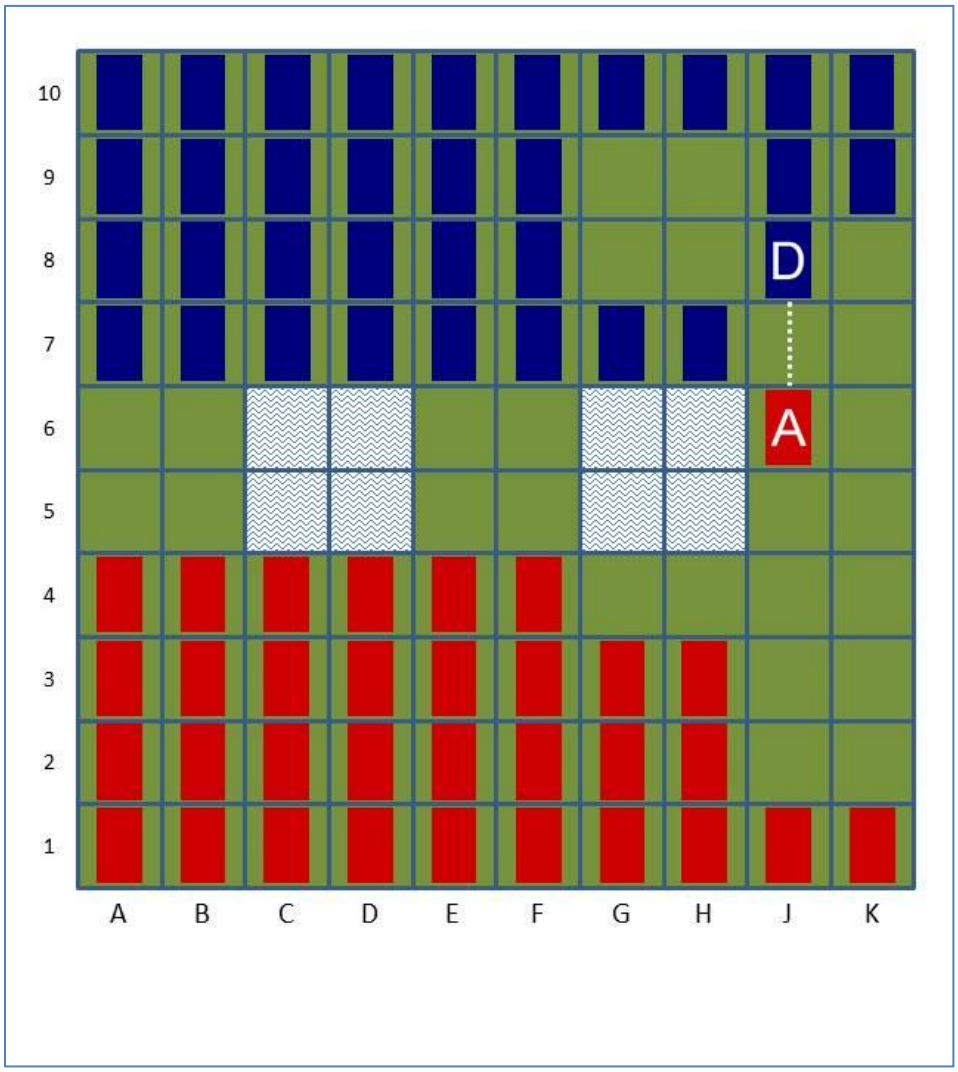
Favourite candidates for capture by an attacker are pieces with a known rank.
Moved pieces may be a favourite prey for (locally) invincible attackers.

If a candidate defender has been found then a search for attackers in the neighbourhood may find a good candidate.

A.2 Check the relative positions of attacker and defender

After the detection of candidate pairs of attacker and defender the positions of these pieces have to be checked. It should be possible for the attacker to come in line with the defender.
If this preliminary work has been done then the starting point of the look ahead function has been attained.

A3. A scan of the line that connects attacker and defender

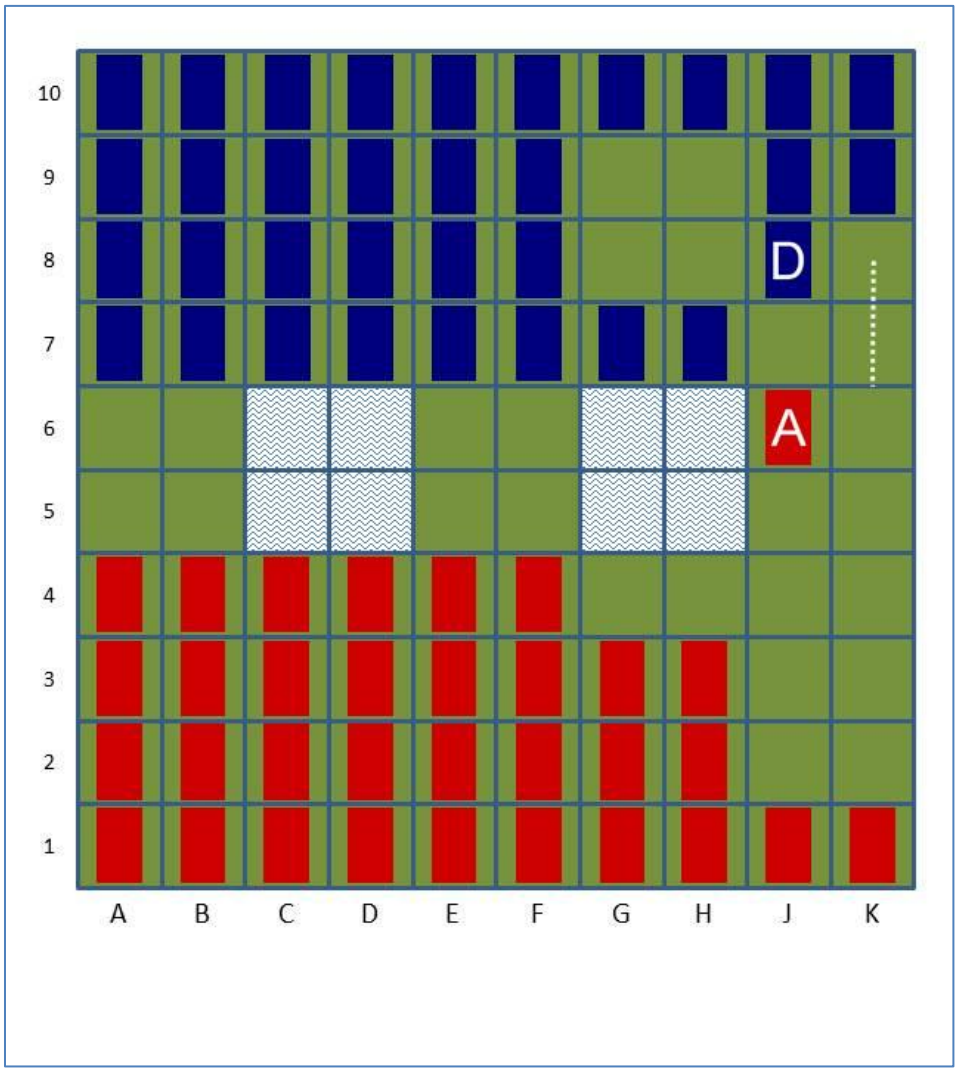


The line between attacker and defender should be accessible for the attacker. So the squares on this line should be:

- Empty
- Occupied by a removable piece of the attackers side
  Examples have been shown in paragraphs 3.3.1 and 4.2.1
- Occupied by a piece of the defenders side with a lower rank than the attacker
  Examples have been shown in paragraph 3.3.2 and 4.2.4
- Unprotected by pieces of the defenders side with a higher rank than the attacker
  An example of protection has been shown in paragraph 4.2.3
- Unoccupied by bombs if the attacker is not a miner
  An example has been shown in paragraph 4.2.4.

If empty squares are present behind the defender then the line should be extended with these empty squares. An example has been shown in paragraph 3.4

A.4 Scan the line next to the line that connects attacker and defender



Lines next to the connection between attacker and defender have to be scanned.
If the connection line is vertical then scans of the lines to the right and left are done.

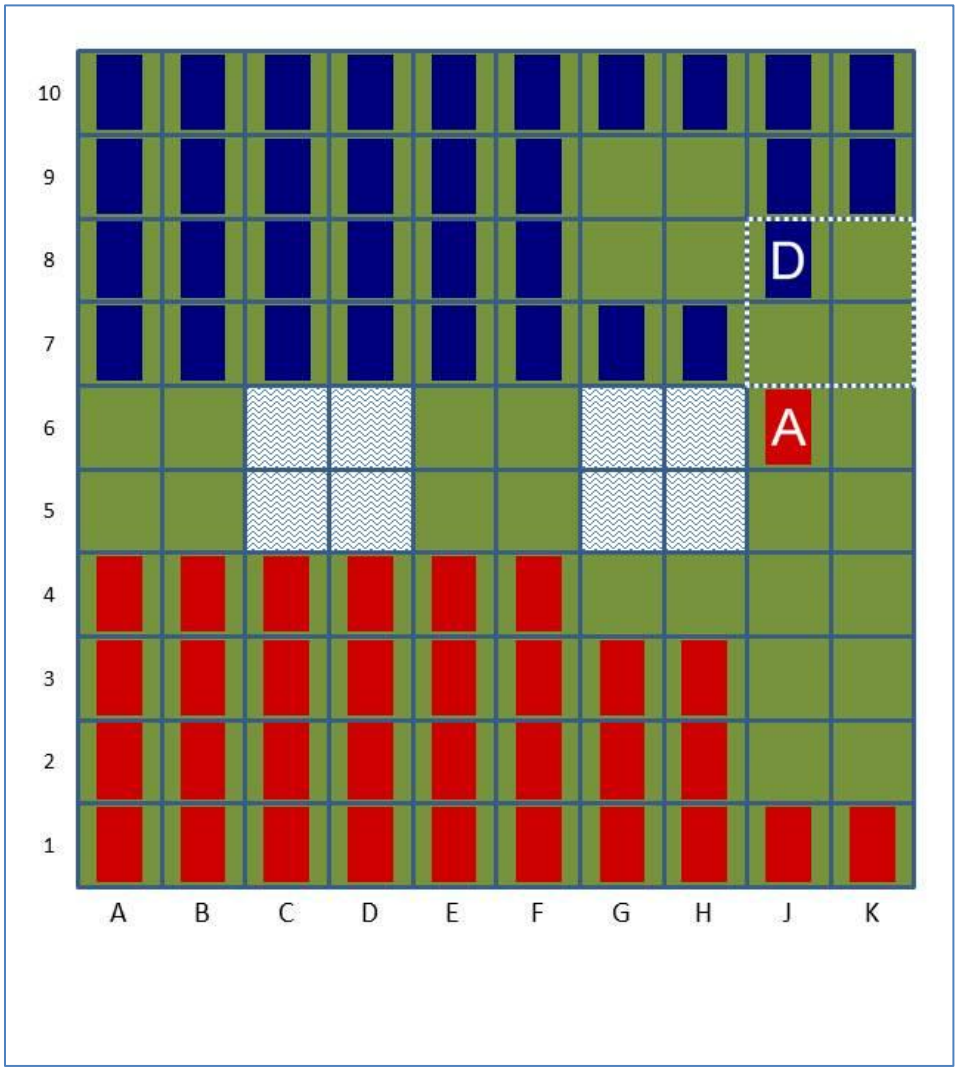
If the connection line is horizontal then lines above and below the connection line are scanned. The scan should include the squares next to both the attacker and the defender. If squares on this line behind the defender are empty then the scan should be extended with those squares.

If all squares on this trajectory are accessible for the attacker then the connection line and the line next to the connection line constitute a candidate trap area.

The conditions for accessibility are the same as for the line that connects attacker with defender. They have been mentioned already in the previous paragraph of this appendix.

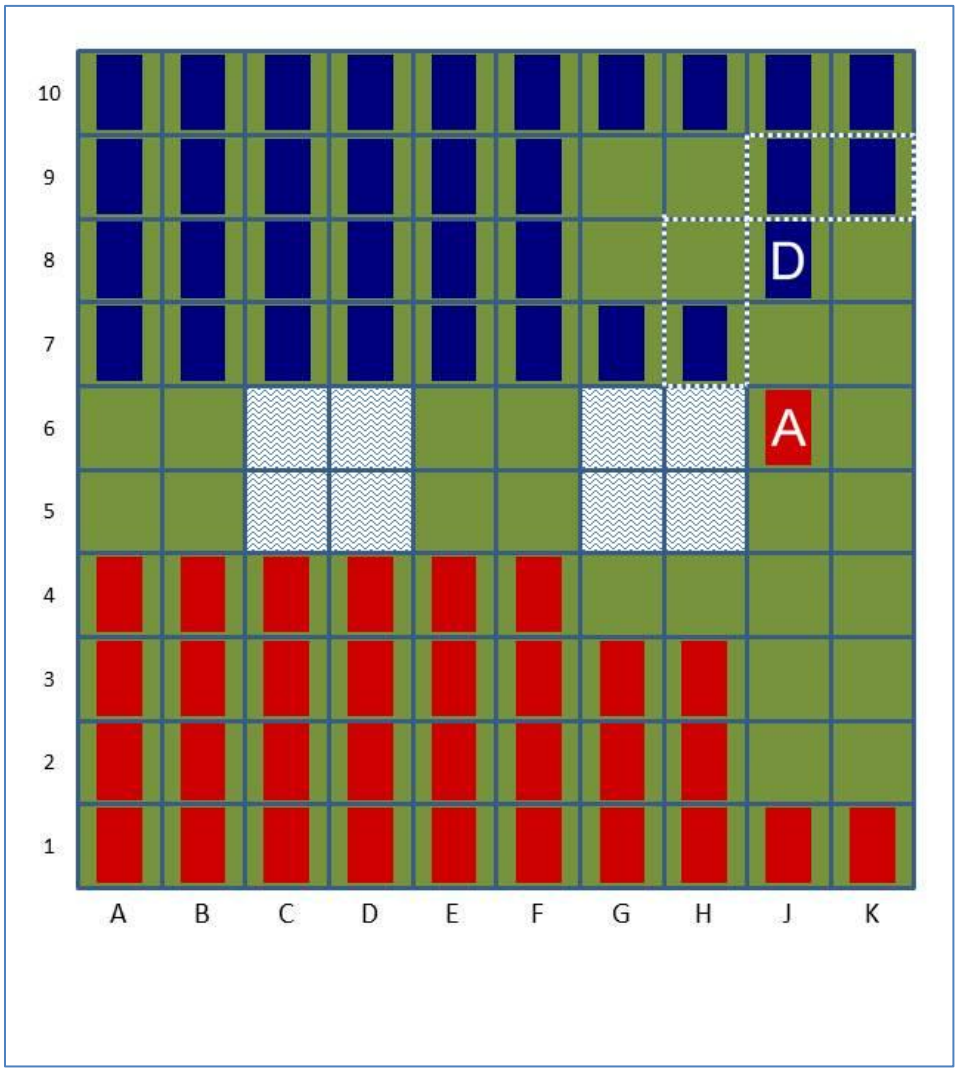
The scan of both neighbouring lines may lead to the condition that the mobility of the defender is being restricted to one line. Then the defender may be captured without use of the two-squares rule.

If all conditions are satisfied a candidate trap area has been found.



A candidate trap area consist of two lines where all squares are accessible for the attacker. Until now the mobility of the defender has not yet been checked. So that is the next action to be done by a scan of the border squares.

A.5 Scan border squares of the candidate trap area



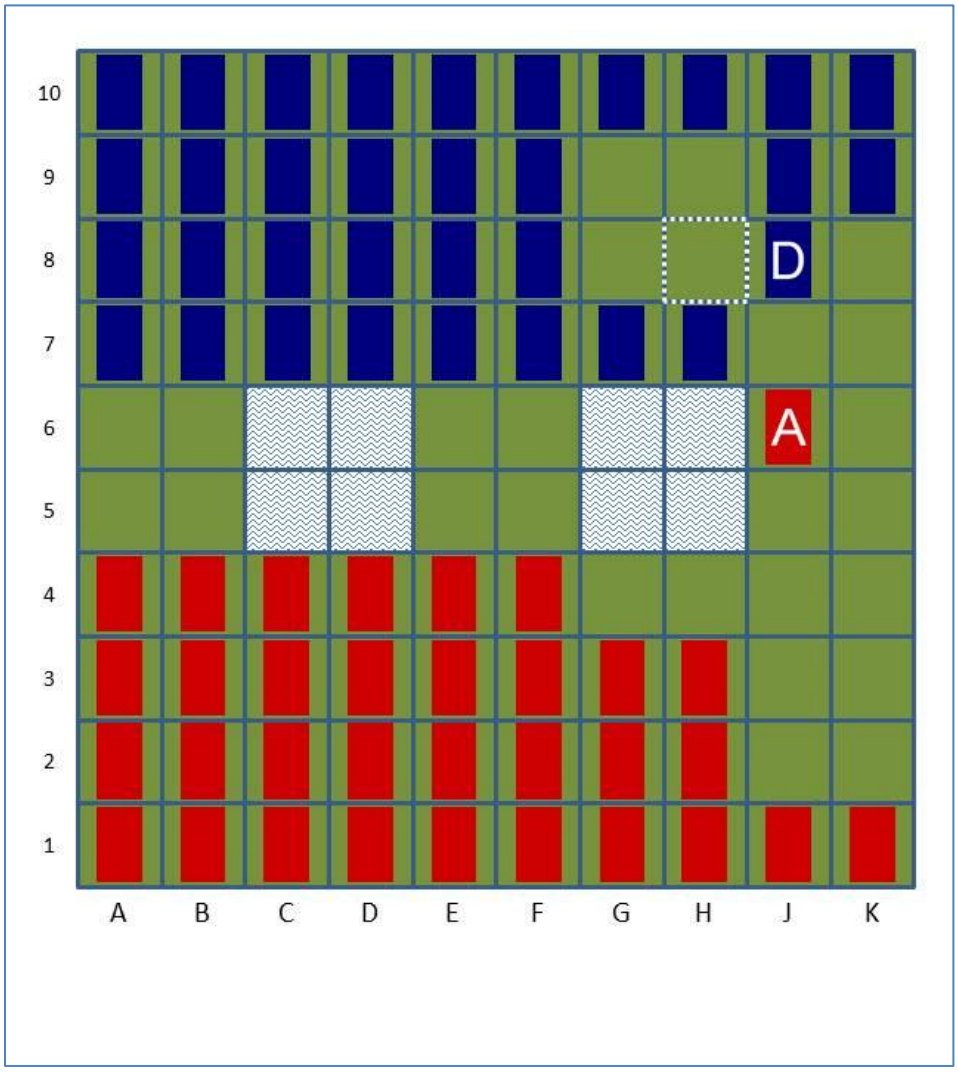
A scan of the border squares reveals the restrictions that are in force to the mobility of the defender.
A square offers a restriction to the mobility of the defender if:

- It is occupied by a piece of the defenders side
- It is imaginary because it lies outside the board
- It is imaginary because it lies in a lake
- It is protected by a piece of the attackers side with a higher rank
- It is occupied by a bomb of the attackers side and the defender is not a miner.

In this example the squares to be scanned are: H7, H8, J9, K9.
Other parts of the border lie outside the board or in the lake.

Squares H7, J9, K7 block the mobility of the defender.

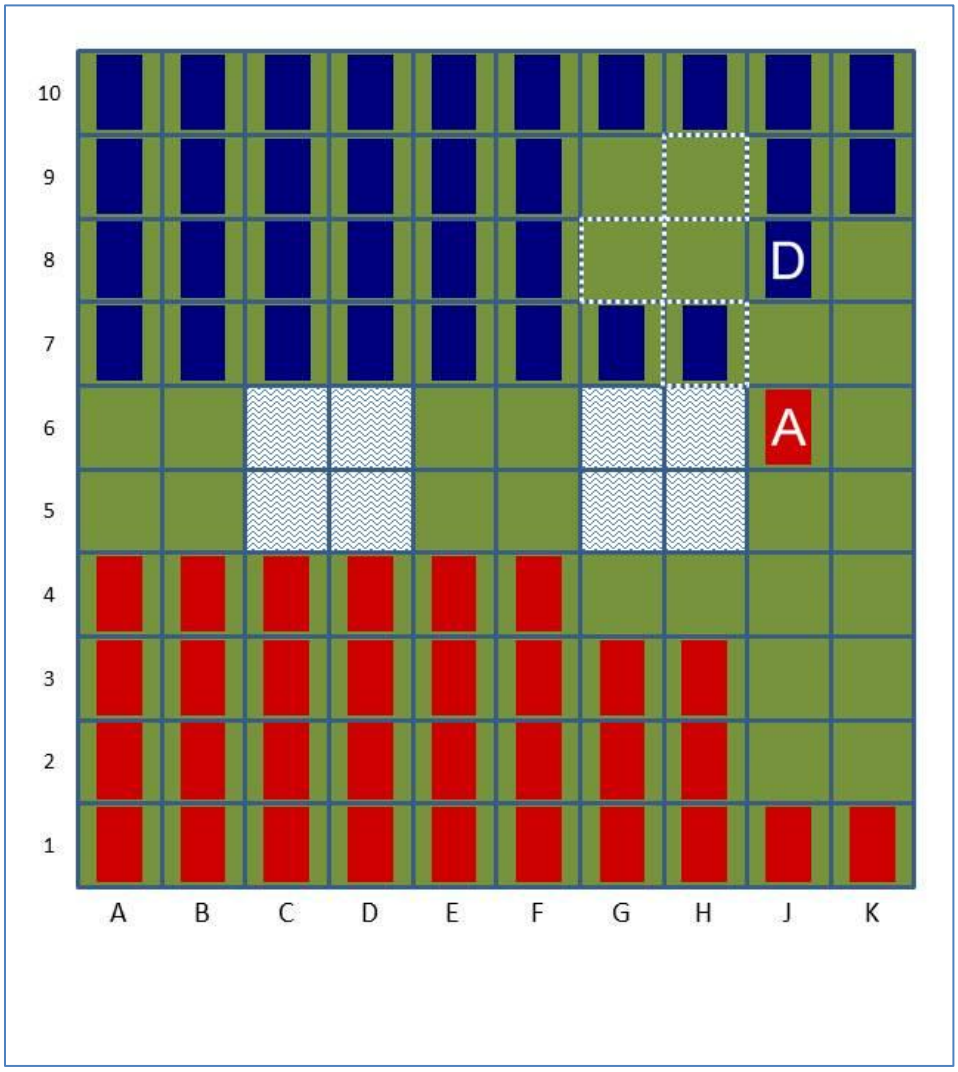
A.5.1 The check of an empty border square



In this example an empty border square has been detected on H8.

If a square is empty an extra and maybe complex check is necessary in order to determine whether the field gives access to a resort or does not provide with a safe spot for defender.

A.5.2 Scan the squares next to the empty border square



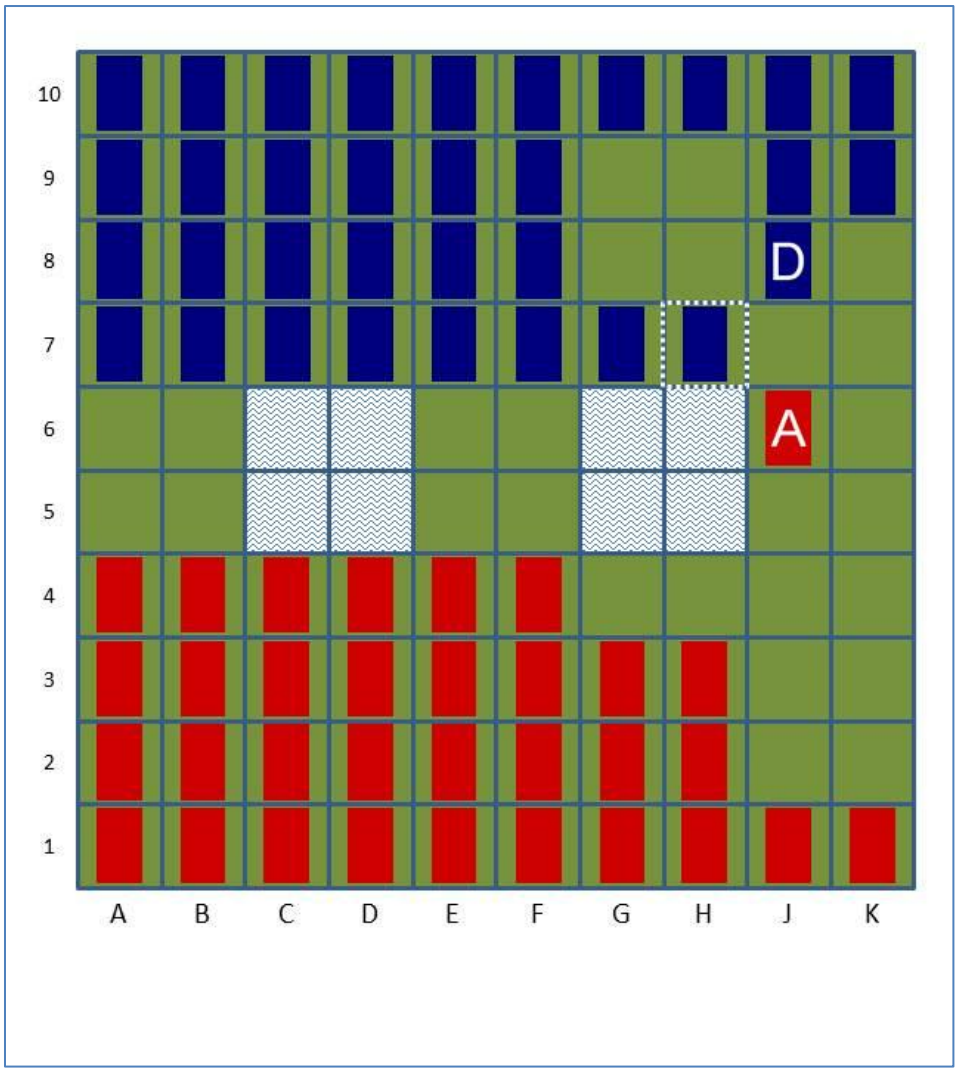
The empty border square is starting point for a scan of surrounding squares. Initially from the border square only squares are considered that lie outside the candidate trap area. This excludes J8 from the initial scan; only the squares H7, G8 and H9 are included in the scan.

For each square a check has to be done:

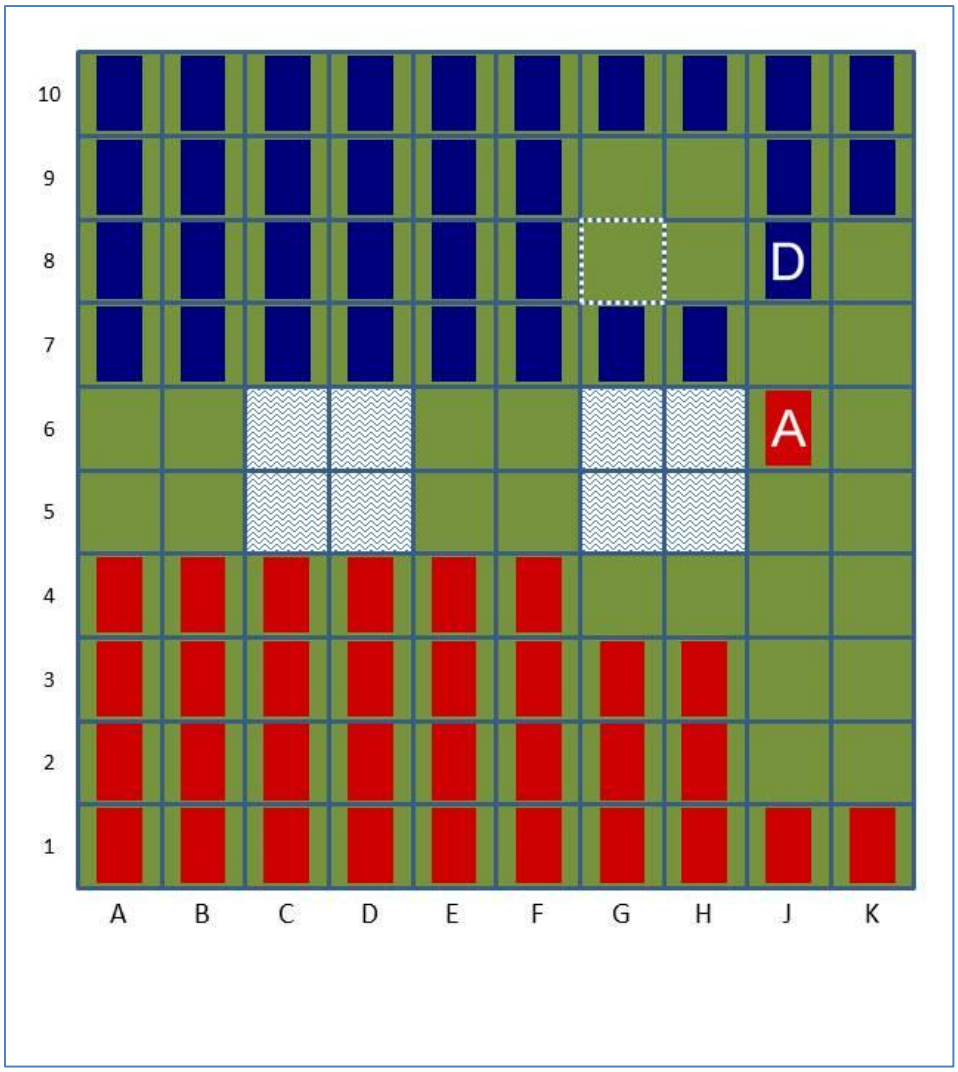
- Is the square empty?
- Is the square occupied by a piece of defenders side with an unknown rank?
- Is the square occupied by a piece of defenders side with a known rank lower than the attacker?
- Is the square occupied by a piece of defenders side with a known rank higher than the attacker?
- Is the square occupied by a piece of attackers side with a rank higher than the defender?
- Is the square occupied by a piece of attackers side with a rank lower than the defender?

Possible results of these checks are:

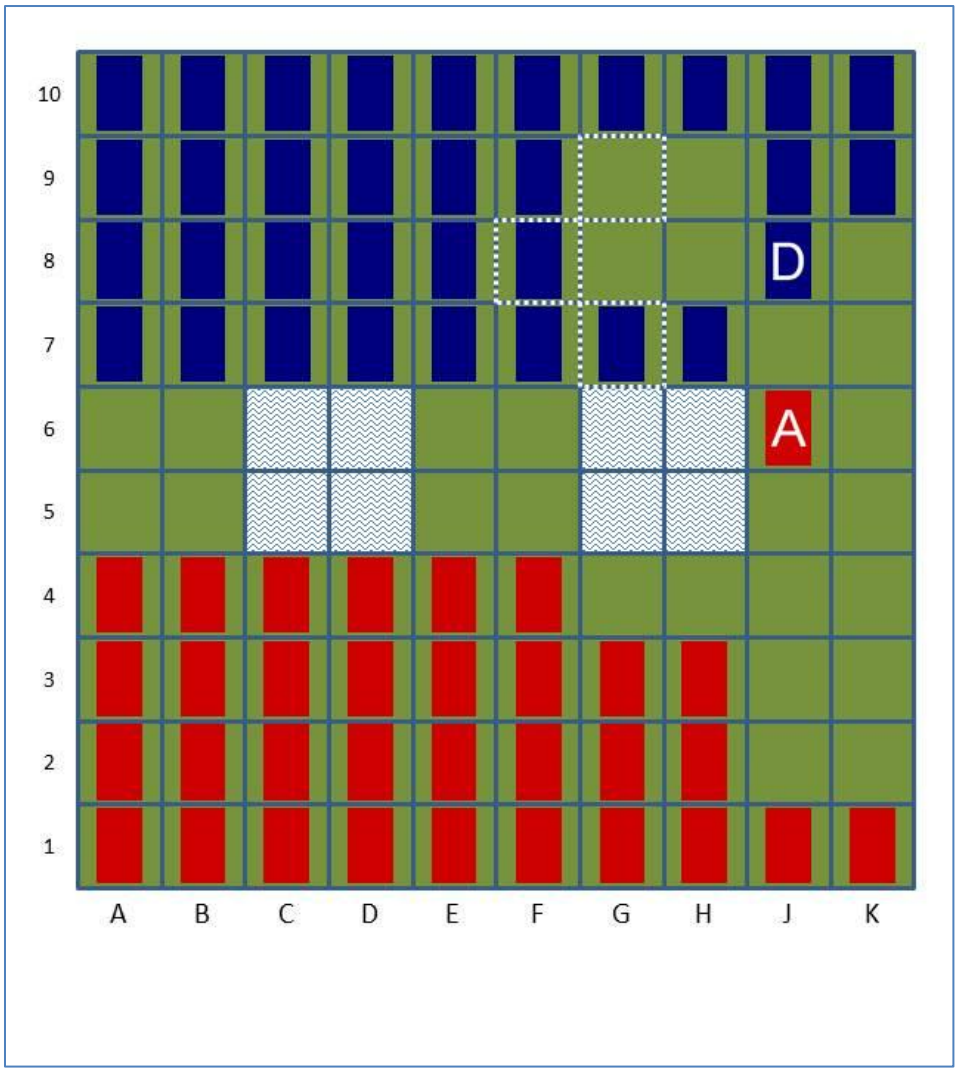
- Defender cannot move to the current square
- Defender may move to the current square and its eventual protection is unknown
- Defender may move to the current square and is protected.



Square H8 is occupied by an unknown blue piece. It will not be possible to find a resort through this square.

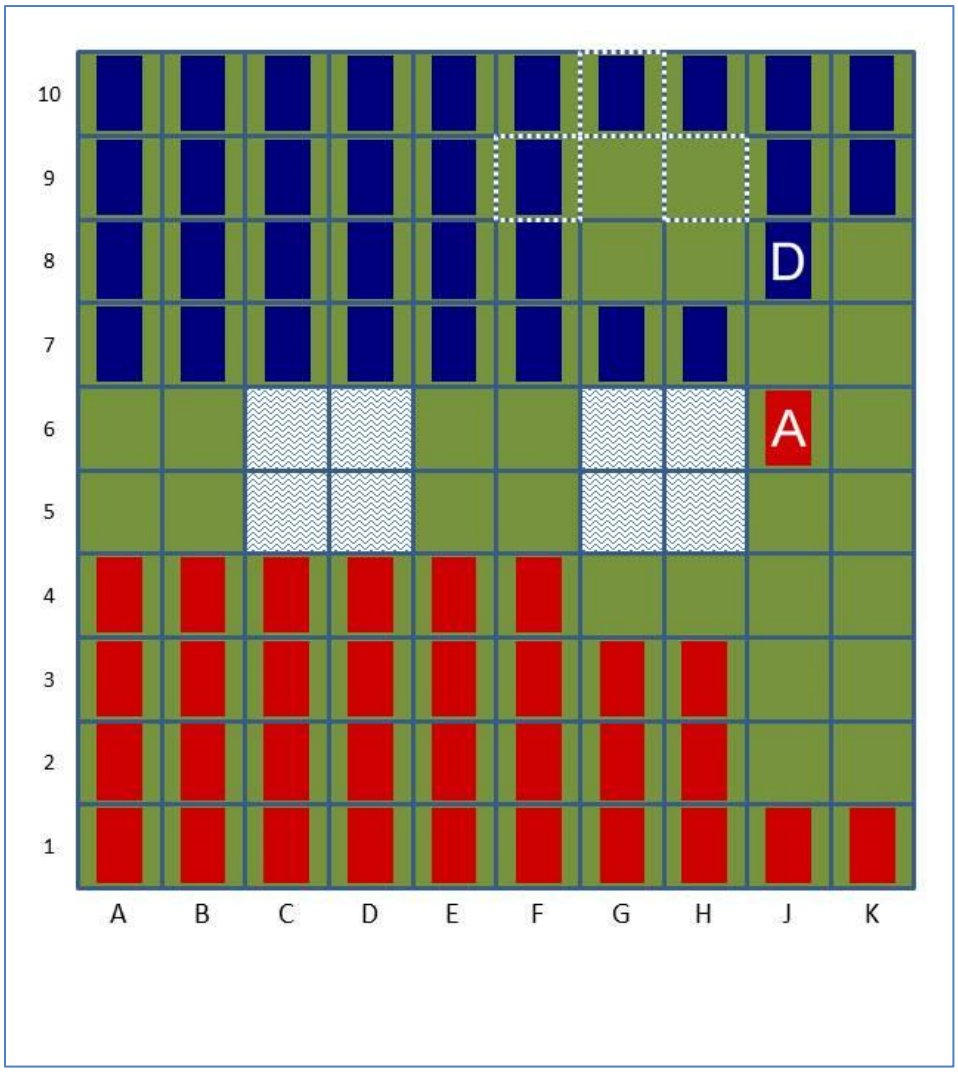


Square H8 is empty. This square may enable to detect a safe 2 x 2 area of empty squares or an open circuit. Every detection of a new empty squares triggers a new scan from the square.



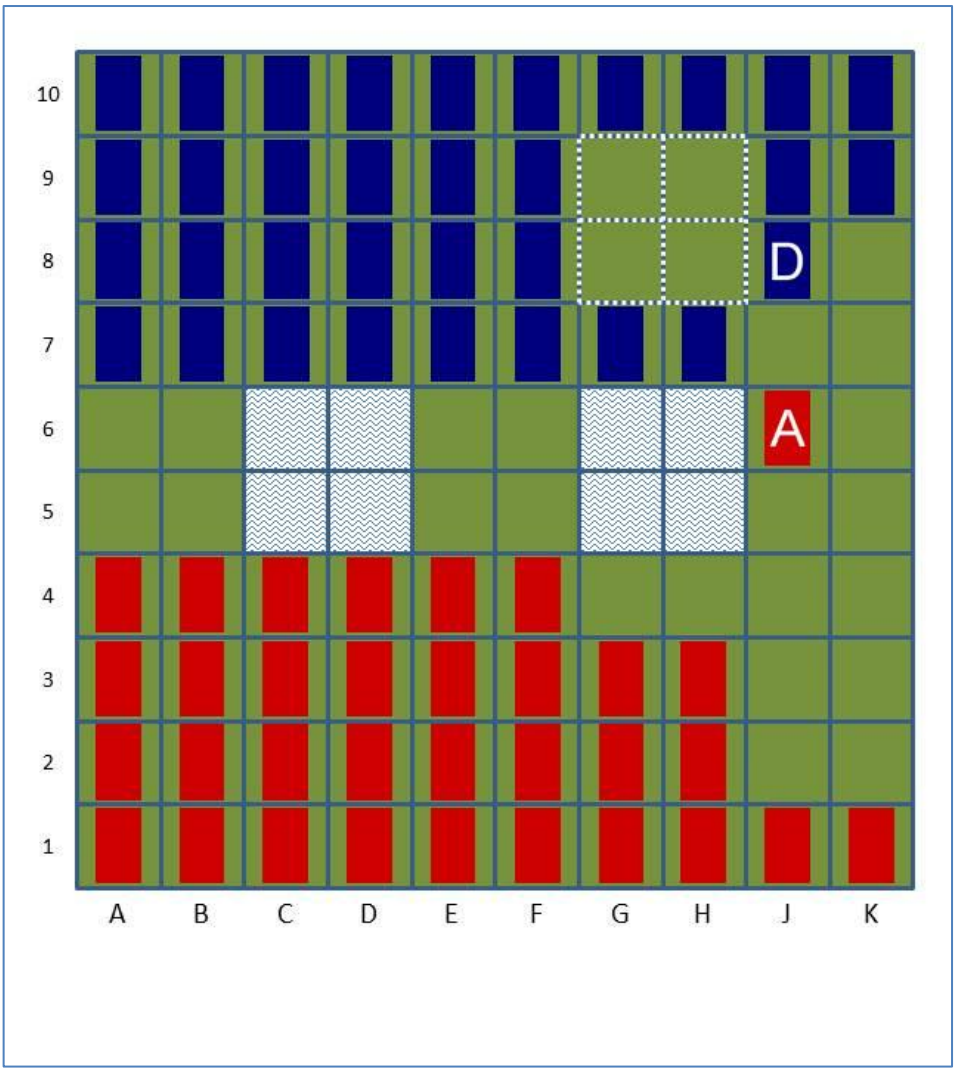
The currently detected empty square G8 is the starting point of a new scan of the squares that surround G8. Only new squares will be included in the scan. Square H8 was the starting point of a former scan and therefore should be excluded. Only the squares G7, F8 and G9 will be relevant in the scan that originates from G8.

G7 is occupied by an unknown blue piece and will not offer a resort for the defender.
F8 too is occupied, therefore this leads to the same result.
G9 is empty. Just like G8 this field therefore is the starting point of a new scan.



The squares G8, F9, G10 and H9 surround the starting point G8.
Square G8 has been the starting point of a former scan and will be excluded from the present scan.

F8 is occupied by an unknown blue piece and will not offer a resort for the defender.
H10 too is occupied with the same result.
H9 is empty.



The consecutive scans have led to the detection of empty squares H8, G8, G9 and H9.

At this point of the processing it is possible to detect that all scanned and empty squares constitute a 2 x 2 area of empty squares.

Because attacker and defender are not next to each other this leads to the conclusion that the defender is able to find a safe spot and will not be captured in the candidate trap area.

A.5.3 Efficiency

This workout of an example is meant to demonstrate one of the many possible solutions for a look ahead function that handles the two-squares rule. Probably much better and more efficient ways can be used to implement a look ahead function for game positions where the two squares rule may or may not lead to a capture.

Probably it is possible to combine data processing for this look ahead function with similar data processing for other functions in the Stratego program. A study of other look ahead functions probably will show that it is possible to detect overlap in the functionality and use a common starting point the for different look ahead functions.

This document is meant for showing functional logic and is not the place to determine what is the most efficient way to implement path or line search in a specific programming language. So this description can only serve as a starting point for the real implementation of the current look ahead function. There remains challenge enough for who feels now the drive to turn this functional view into reality.