Group 6: Black and White Squares

Logan Philip

Ben Jacoby

Ryan Waterson

Tuan Hiep (Alex) Nguyen

# A screenshot of a game Description automatically generatedProject Summary

The symbols our project concerns are black and white squares that must be separated such that no two different colours are in the same “section” as denoted by the drawn line.

The drawn line must start on the designated start point, which is indicated by the circle in the bottom left corner of the diagram. It must also end on the designated end point, which is indicated by the marking on the top left corner of the diagram.

For our model, we will use two separate 2D grids that we will call the “tile grid” and the “point grid”.

The tile grid is a 3x3 grid that will represent the locations of the black and white squares and any other information concerning the tile area. There can be at most one empty square on the tile grid. The tile area is represented by the green-coloured area in the diagram.

Image courtesy of witnesspuzzles.com

The point grid, on the other hand, is a 4x4 grid that will represent the location of the line, the starting point of the line and the ending point. The point grid is represented by the area of the diagram containing the white line itself and the grey space surrounding the tile area.

The primary exploration of this model is determining whether a given line is a valid solution to the board; that is, there are no black squares in the same section as white squares, as denoted by the drawn line. There are some other explorations of the model that we explored too.

# Propositions

**Tile Propositions** (Propositions using the tile grid)

* wx,y: This is true when tile (x,y) on the tile grid contains a white square.
* bx,y: This is true when tile (x,y) on the tile grid contains a black square.
* tx,y: This is true when tile (x,y) on the tile grid contains neither a white nor black square (empty)
* jx,y: This is true when a black square is touching the tile at (x,y).
* kx,y: This is true when a white square is touching the tile at (x,y).
* ix,y: This is true when an empty tile at (x,y) is touching a black square and a white square.
* px,y: This is true when a white square at (x,y) is touching a black square.

**Point Propositions** (Propositions using the point grid)

* lx,y: This is true when a line is on the point grid at (x,y).
* sx,y: This is true when (x,y) on the point grid is the starting point of the line.
* ex,y: This is true when (x,y) on the point grid is the ending point of the line.
* cx,y,dir: This is true when the line at (x,y) is connected to another line above it (*dir* = 0) or to the right of it (*dir* = 1)
* dx,y,n: This is true when the line at (x,y) is *n* distance away from the starting point (travelling along the line).

**Other Propositions**

* q: This is true when the drawn line is a valid solution to the board.

# Constraints

**Tile Constraints** (Constraints concerning the tile grid)

* There can be at most one empty space.
  + ¬(t0,0∨ t1,0∨…∨ t4,4) ∨ (t0,0 ∧ ¬(t1,0∨ t2,0∨…∨ t4,4)) ∨ (t1,0∧ ¬(t0,0∨ t2,0∨…∨ t4,4)) ∨ …
* An empty tile is only touching a black square and a white square when the tile has neither white nor black on it and the previous constraints for touching black and white squares are met.
  + ix,y ↔ tx,y ∧ jx,y ∧ kx,y
* Similar idea applies for white squares touching black squares.
  + px,y ↔ wx,y ∧ jx,y
* A square can be exclusively black or white, not both.
  + (wx,y→ ¬bx,y)∧ (bx,y→ ¬wx,y)
* A solution is only solved when no empty spaces are touching black and white squares, and no white squares are touching black squares.
  + q ↔ ¬(i0,0 ∨ i1,0 ∨ i2,0 ∨… ∨ i3,3) ∧ ¬(p0,0 ∨ p1,0 ∨ p2,0 ∨… ∨ p3,3)
* A tile is empty if it doesn’t contain a white or black square.
  + tx,y ↔ ¬bx,y ∧ ¬wx,y

**Point Constraints** (Constraints concerning the point grid)

* There must always be **one** starting point.
  + (s0,0 ∧ ¬(s1,0∨ s2,0∨…∨ s4,4)) ∨ (s1,0 ∧ ¬(s0,0∨ s2,0∨…∨ s4,4)) ∨ …
  + The same applies to the ending point.
* A point can't be both the starting point and ending point.
  + ¬sx,y ∨ ¬ex,y
* There must always be a line segment at the starting point and ending point.
  + sx,y → lx,y
  + ex,y → lx,y
* A point can't be connected to a non-existent point “out-of-bounds”.
  + ¬cx,4,0
  + ¬c4,y,1
* A point being connected to another point implies that there is a line segment at both ends of the connection.
  + cx,y,0 → (lx,y ∧ lx,y+1)
  + cx,y,1 → (lx,y ∧ lx+1,y)
* Any point on the line that isn’t the start or end point must be connected to **two** other points of the line. In other words, the line must be a single continuous line from start to end without branching paths.
  + lx,y →

( ( (sx,y ∨ ex,y) ∧ ( (cx,y,0 ∧ ¬cx,y,1 ∧ ¬cx,y-1,0 ∧ ¬cx-1,y,1) ∨ (¬cx,y,0 ∧ cx,y,1 ∧ ¬ cx,y-1,0 ∧ ¬ cx-1,y,1) ∨ … ) ) ∨ ( ¬sx,y ∧ ¬ex,y ∧ ( (cx,y,0 ∧ cx,y,1 ∧ ¬cx,y-1,0 ∧ ¬cx-1,y,1) ∨ (cx,y,0 ∧ ¬cx,y,1 ∧ cx,y-1,0 ∧ ¬cx-1,y,1) ∨ … ) ) )

* The distance to the start at the starting point must be 0.
  + sx,y → dx,y,0
* At any line segment there must be *another* line segment connected to it with a distance less than the specified line segment.
  + dx,y,n → ( (cx,y,0 ∧ dx,y+1,n-1) ∨ (cx,y,1 ∧ dx+1,y,n-1) ∨ (cx,y-1,0 ∧ dx,y-1,n-1) ∨ (cx-1,y,1 ∧ dx-1,y,n-1) )
* There can be at most one line segment with any given distance to the start (distance is measured by travelling along the line).
  + ¬(d0,0,n ∨ d1,0,n ∨…∨ d4,4,n) ∨

(d0,0,n ∧ ¬(d1,0,n ∨ d2,0,n ∨…∨ d4,4,n)) ∨ (d1,0,n ∧ ¬(d0,0,n ∨ d2,0,n ∨…∨ d4,4,n)) ∨ …

* + This constraint applies separately for every value of *n* from 0 to the max possible distance any line could have to the start.
* Every point with a distance can only have one distance at that point.
  + dx,y,0 → ¬(dx,y,1 ∨ dx,y,2 ∨ … ∨ dx,y,max)
* Every point with a line segment must also have a distance to the start.
  + lx,y → (dx,y,0 ∨ dx,y,1 ∨ … ∨ dx,y,max)

**Point and Tile Constraint** (Constraint concerning both grids at once)

* A square is “touching” a black square only when a black square is a single x *or* y coordinate away (no diagonals) and there is no connection between line segments between them.
  + jx,y ↔ (bx,y+1 ∧ ¬cx,y+1,1) ∨ (bx+1,y ∧ ¬cx+1,y,1) ∨ (bx,y-1 ∧ ¬cx,y,1) ∨ (bx-1,y ∧ ¬cx,y,1)
  + Propositions jx,y and bx,y are constraining the tile grid while proposition cx,y,dir is constraining the point grid using the same x and y coordinates. The set of x and y coordinates used for this constraint is the set of x and y coordinates used for the tile grid. This is needed to allow the point and tile grids to function together.
  + Same idea applies for white squares (kx,y).

# Model Exploration

**Primary modes:**

The first way we began exploring our model was by trying to generate any configuration of black and white squares, and any completed line and determining if the model is in a “solved” state. A solved state is any combination of propositions that do not have any white squares touching any black squares, or and empty tiles touching both a black square and a white square. This is the default mode of the python program, or mode “0”. The program will find any valid set of propositions, determine if the board is solved, and print the results along with a nice visual representation of the game board. Using this mode, you can see that there are clearly many more “unsolved” board arrangements than there are “solved” ones. For a board with a grid size of 2x2, roughly 15% of all the possible valid board configurations are solved, but this number could be different for the standard 3x3 size; finding the total number of valid board configurations for a 3x3 is far too complex of a calculation for the program to run.

Since finding a solved board is fairly rare, we’ve also included another mode to the program that only finds solved board configurations: mode “1”. This mode started off as the only mode during the majority of the development process. It made it much easier to determine if the constraints were properly working, and if what the program deemed as “solved” was actually what it should be.

**Extra modes:**

After getting our model to a spot we were happy to call complete, we started looking into more ways to explore it. This was when we chose to explore what happens when the grid size changes. This was fairly east to implement, apart from some minor issues, since the program was written from the start with a variable board size in mind. To change the grid size, one can simply change the number of columns and rows inside the desired mode at the bottom of the run.py file. Expanding the board size produces the results one might expect: more black and white squares, and generally a much longer line. Expanding the grid makes it even more difficult to find a solved board, which further indicates that the larger the grid, the smaller the percentage of valid models that are solved.

Next, we designed two new modes to the program that accept a new input of either a static configuration of black and white squares, or a static line configuration, respectively.

The first of these, the static tile configuration mode, or mode “2”, runs the program with a static arrangement of black squares, white squares, and the starting and ending points. This mode takes the static configuration and finds all possible lines that solve the board. The program will then print all of these solved boards with a visual representation of each board along with the number of possible solutions to the static board. The static configuration itself can be changed by modifying the matrix in mode 2 at the bottom of the run.py file. The matrix is laid out such that the items’ locations in code directly represent the locations they will be in the model. Because of this particular way of implementing the matrix, and because it would be too complex to calculate anything larger, this mode will only properly work for 3x3 grids.

And finally, the static line configuration mode, or mode “3”, runs the program with a static line from start to end. This mode takes the static line and finds all combinations of black and white squares that the line would be a solution to. The program will once again print of all these solved boards with a visual representation of each board and the number of possible arrangements. The static line configuration can be changed by modifying the list of points in mode 3 at the bottom of the run.py file. The list of points should be created such that the first point is the start of the line, the last point is the ending of the line, and each point in between is only one x **or** y position away from the previous point. This mode should only be used with grids sized at most 4x4, because after that, the model gets far too complex to calculate and print all the solutions.

**Results:**

Overall, our project has successfully completed what we originally set out to do in trying to determine whether a given game board is solved or not. The process of figuring out if a given board was solved or not was not the most difficult part, however. Rather, we found it much more difficult to come up with a set of constraints that properly restricted the line to a single continuous line from the starting point to the ending point. But after a lot of trial and error, and of course, help, we finally got there in the end.

We were able to find out many things about our model once we finally got it working. Like already mentioned, we discovered that there are significantly more unsolved board states than there are solved states, and that the percentage of solved states is likely inversely related to the size of the board.

Using the static tile mode allows us to see all the solutions to any given arrangement of squares, which can be interesting to see. Some arrangements have very few solutions, some have many, and some even have none at all. Notably, we can see that if we enter a tile configuration with a black or white square that is impossible to isolate, then that configuration will have no solutions. And generally, configurations with very little of either colour tend to have many solutions. But all of that also depends on where the start and end point is, because one configuration of black and white squares may have a ton of solutions with one set start and end point, but none, or hardly any, with another.

As for the static line mode, we have found that with most line configurations, as long as the line creates some dividing line that splits the board into at least two distinct sections, there will always be at least two arrangements of black and white squares that would cause the board to be solved. We can see this is the case because 1: our model requires that there be at least one black square and at least one white square, so there must always be somewhere for either colour be become isolated from the other; and 2: any possible arrangement of black and white squares has an inverse, such that in any given arrangement you can swap all black squares with white squares and all white squares with black squares and be left with another valid solution. We have also found that for any given static line, the larger the number of “sections” created by the line, the larger the number of solutions. Each section can have either black squares or white squares in it, so if a section has one white square in it, all the squares in that section must also be white. If you have more sections, less of the tiles on the board will be tied to the colours of other tiles. Therefore, the more sections that there are, the more freedom there is to place either colour in any given tile, resulting in more solutions overall.

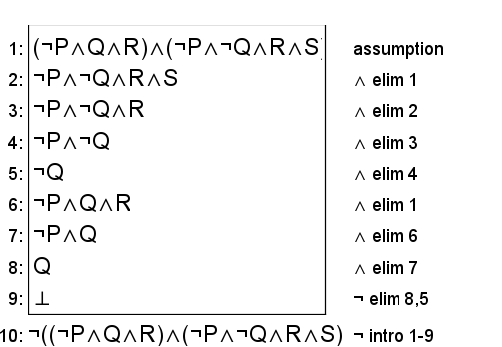
# Jape Proof Ideas

First proof:

Relevant constraints:

* An empty space is only touching a black square and a white square when the point has neither white nor black on it and the previous constraints for touching black and white squares are met.
  + ex,y ↔ ¬ bx,y ∧ ¬ wx,y ∧ jx,y ∧ kx,y
* Similar idea applies for white squares touching black squares.
  + px,y ↔ ¬ bx,y ∧ wx,y ∧ jx,y

Proof that it is impossible to have an empty space touching a black square and white square while having that same space be a white square:

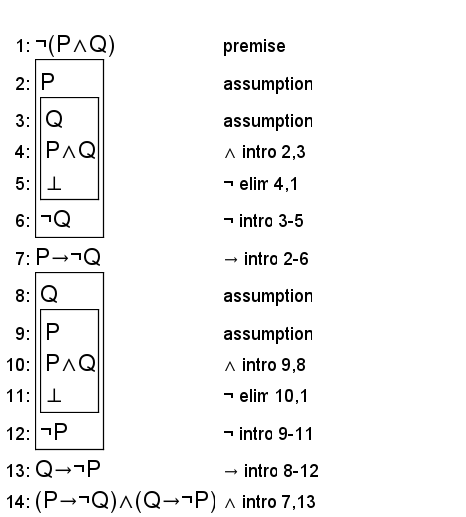


Second proof:

Relevant constraint:

* A grid space can’t be both the starting point and ending point.
  + ¬(sx,y ∧ ex,y)

Proof that a given grid space being the starting point implies that it is not the ending point, and vice versa:

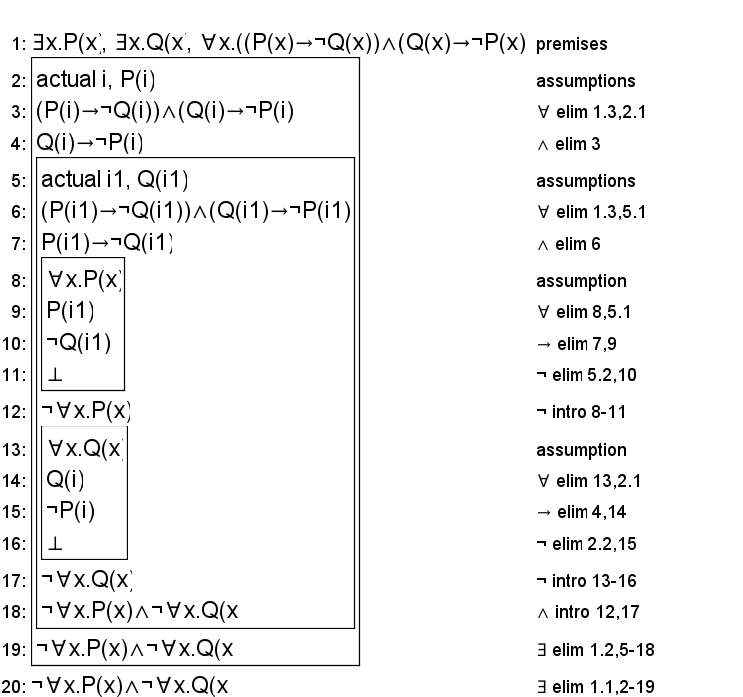


Third proof:

Relevant constraint:

* A square can be exclusively black or white, not both.
  + (wx,y→ ¬bx,y)∧ (bx,y→ ¬wx,y)

Proof that if there exists a white square and a black square (P(x) and Q(x) in this proof)) and that a square being white implies it is not black, and vice versa (shown through the constraint), that it is not possible for all squares to be black and it is not possible for all squares to be white.



# First-Order Extension

Many of the constraints of the model involve defining that a given number of coordinates on a grid must have (or lack) some property. For example, it is defined that the tile grid cannot have more than one empty square, and it is defined that there must be exactly one starting point on the point grid. These constraints are quite long and unwieldy when using propositional logic and are greatly simplified when using predicate logic.

A common tactic we used to implement the first-order extension was to state that there cannot exist two unique points such that those two points share some property. For example, that no two points can be the starting point. Many of the updated constraints are of the form ¬∃(x1,y1),(x2,y2).(A(x1,y1) ∧ A(x2,y2)), where A is some proposition and (x1,y1) ≠ (x2,y2).

When changing the overall structure of the constraint did not provide any meaningful benefit, a common tactic we used to transition them from propositional logic to predicate logic was to use the universal quantifier. This involved updating the constraint to the form ∀(x,y).(A(x,y) ↔ (…)), where A is some proposition and … represents the only conditions under which A is true.

The propositions have not been changed in any meaningful way except for appearance, as we did not find any significant benefit to changing them for predicate logic. The main power of using predicate logic was in applying these propositions.

## Updated Propositions

**Tile Propositions** (Propositions using the tile grid)

* W(x,y): This is true when tile (x,y) on the tile grid contains a white square.
* B(x,y): This is true when tile (x,y) on the tile grid contains a black square.
* T(x,y): This is true when tile (x,y) on the tile grid contains neither a white nor black square (empty)
* J(x,y): This is true when a black square is touching the tile at (x,y).
* K(x,y): This is true when a white square is touching the tile at(x,y).
* I(x,y): This is true when an empty tile at (x,y) is touching a black square and a white square.
* P(x,y): This is true when a white square at (x,y) is touching a black square.

**Point Propositions** (Propositions using the point grid)

* L(x,y): This is true when a line is on the point grid at (x,y).
* S(x,y): This is true when (x,y) on the point grid is the starting point of the line.
* E(x,y): This is true when (x,y) on the point grid is the ending point of the line.
* C(x,y,*dir*): This is true when the line at (x,y) is connected to another line above it (*dir* = 0) or to the right of it (*dir* = 1)
* D(x,y,n): This is true when the line at (x,y) is *n* distance away from the starting point (distance is measured by travelling along the line).

**Other Propositions**

* Q: This is true when the drawn line is a valid solution to the board.

## Updated Constraints

**Tile Constraints** (Constraints concerning the tile grid)

* There can be at most one empty space.
  + ¬∃(x1,y1),(x2,y2).(T(x1,y1) ∧ T(x2,y2))
  + Note that (x1,y1) ≠ (x2,y2)
* An empty tile is only touching a black square and a white square when the tile has neither white nor black on it and the previous constraints for touching black and white squares are met.
  + ∀(x,y).(I(x,y) ↔ (T(x,y) ∧ J(x,y) ∧ K(x,y)))
* Similar idea applies for white squares touching black squares.
  + ∀(x,y).(P(x,y) ↔ (W(x,y) ∧ J(x,y)))
* A square can be exclusively black or white, not both.
  + ¬∃(x,y).(W(x,y) ∧ B(x,y))
* A solution is only solved when no empty spaces are touching black and white squares, and no white squares are touching black squares.
  + Q ↔ ¬∃(x,y).I(x,y) ∧ ¬∃(x,y).P(x,y)
* A tile is empty if it doesn’t contain a white or black square.
  + ∀(x,y).(T(x,y) ↔ (¬W(x,y) ∧ ¬B(x,y)))

**Point Constraints** (Constraints concerning the point grid)

* There must always be **one** starting point.
  + ¬∃(x1,y1),(x2,y2).(S(x1,y1) ∧ S(x2,y2)) ∧ ∃(x,y).S(x,y)
  + Note that (x1,y1) ≠ (x2,y2)
  + The same applies to the ending point.
* A point can't be both the starting point and ending point.
  + ¬∃(x,y).(S(x,y) ∧ E(x,y))
* There must always be a line segment at the starting point and ending point.
  + ¬∃(x,y).(S(x,y) ∧ ¬L(x,y))
  + ¬∃(x,y).(E(x,y) ∧ ¬ L(x,y))
* A point can't be connected to a non-existent point “out-of-bounds”.
  + ¬∃x.(C(x,4,0)∧ ¬∃y.(C(4,y,1))
* A point being connected to another point implies that there is a line segment at both ends of the connection.
  + ¬∃(x,y).(C(x,y,0) ∧ (¬L(x,y) ∨ ¬L(x,y+1)))
  + ¬∃(x,y).(C(x,y,1) ∧ (¬L(x,y) ∨ ¬L(x+1,y)))
* Any point on the line that isn’t the start or end point must be connected to **two** other points of the line. In other words, the line must be a single continuous line from start to end without branching paths.
  + ∀(x,y).( L(x,y) →

( ( (S(x,y) ∨ E(x,y)) ∧ ( (C(x,y,0) ∧ ¬C(x,y,1) ∧ ¬C(x,y-1,0) ∧ ¬C(x-1,y,1)) ∨ (¬C(x,y,0) ∧ C(x,y,1) ∧ ¬ C(x,y-1,0) ∧ ¬ C(x-1,y-1,1)) ∨ … ) ) ∨ ( ¬S(x,y) ∧ ¬E(x,y) ∧ ( (C(x,y,0) ∧ C(x,y,1) ∧ ¬ C(x,y-1,0) ∧ ¬ C(x-1,y-1,1)) ∨ (C(x,y,0) ∧ ¬ C(x,y,1) ∧ C(x,y-1,0) ∧ ¬ C(x-1,y-1,1)) ∨ … ) ) )

* The distance to the start at the starting point must be 0.
  + ¬∃(x,y).(S(x,y) ∧ ¬D(x,y,0))
* At any line segment there must be *another* line segment connected to it with a distance less than the specified line segment.
  + ¬∃(x,y).(D(x,y,n) ∧ ¬(C(x,y,0) ∧ D(x,y+1,n-1)) ∧ ¬(C(x,y,1)∧D(x+1,y,n-1)) ∧ ¬(C(x,y-1,0) ∧ D(x,y-1,n-1)) ∧ ¬(C(x-1,y,1)∧(D(x-1,y,n-1))))
* There can be at most one line segment with any given distance to the start (distance is measured by travelling along the line).
  + ¬∃(x1,y1),(x2,y2),n.(D(x1,y1,n) ∧ D(x2,y2,n)
  + Note that (x1,y1) ≠ (x2,y2)
  + This constraint applies separately for every value of *n* from 0 to the max possible distance any line could have to the start.
* Every point with a line segment must also have a distance to the start.
  + ¬∃(x,y).(L(x,y) ∧ ¬∃n.(D(x,y,n))

**Point and Tile Constraint** (Constraint concerning both grids at once)

* A square is “touching” a black square only when a black square is a single x *or* y coordinate away (no diagonals) and there is no connection between line segments between them.
  + ∀(x,y).(J(x,y) ↔(B(x,y+1) ∧ ¬C(x+1,y,1)) ∨ (B(x+1,y) ∧ ¬C(x+1,y,1)) ∨ (B(x,y-1) ∧ ¬C(x,y,1)) ∨ (B(x-1,y) ∧ ¬C(x,y,1)))
  + Propositions J(x,y) and B(x,y) are constraining the tile grid while proposition C(x,y,dir) is constraining the point grid using the same x and y coordinates. The set of x and y coordinates used for this constraint is the set of x and y coordinates used for the tile grid. This is needed to allow the point and tile grids to function together.
  + Same idea applies for white squares (K(x,y)).