# Project Summary

Laser chess is a game like chess with two sides where both teams have a Laser, four triangular pieces with one mirrored side and a King piece. The goal of the game is to reflect the Laser using the mirrored pieces in the direction of the King. If the King is hit by the Laser, the game is over and the player who hit their opponent’s king wins.

We decided to use a slightly altered version of Laser Chess for simplicity’s sake in this project. In our version, for a move you may either move a piece one space in any direction or rotate a piece 90 degrees in any direction. The Laser always starts at the same position at (0,0) on the board. At the end of the turn, the Laser will be shot. If it goes off the map nothing happens and it is your opponent’s turn, if it hits the non-mirrored side of a piece no matter what team that piece is on that piece is removed from the board and if it hits a king the game is over and the team with the king left standing wins.

For our modeling projecting we will be checking if it is possible for the Laser to reach the King after one move. The Laser is in a fixed position at (0,0) and the King is in a fixed position at (4,3) on our 5x4 board. The four mirrored pieces will all be on our team will be randomly placed on the board.

|  |  |
| --- | --- |
| Chart  Description automatically generated | Chart  Description automatically generated with medium confidence |
| This is an example situation on our board where the Laser does not hit the King, therefore we do not win. | This is an example situation on our board where the Laser hits the King after using some of the pieces to reflect in its direction. |

# Propositions

**The Pieces:** P(x,y),(o) , every piece has an (x, y) position, (o) orientation. For examples, P(1,1) (NW) would be True for a piece in the position (1,1) with an orientation of (NW). The options for orientation are Northwest (NW), Southwest (SW), Southeast (SE) and Northeast (NE). These describe the side of the piece that has the mirror.

**The Laser:** L(x,y), (d) is the laser that has a position (x, y) and (d) for the direction of the laser. For example, if the Laser is at the position (2,3) and the direction (N), the proposition L(2, 3), (N) evaluates to True. There are four options that the direction may have which are North (N), South (S), East (E), and West (W)

**The King:** K(x,y) is the opponents King that has a position (x, y). In our version of this Laser Chess, the King always has a fixed position of (4,3).

**Game Over:** G is the proposition for game over. G is True if the opponent’s king is hit by the Laser.

# Constraints

G (Game Over) only holds when the Laser and the King have the same position. For example:

**(K2,2 ∧ L2,2,N) G.**

A Piece cannot move to a position where there is already another piece.

The Laser always starts at the same position with the same orientation:

**L(0,0) (E)**

The King always starts on the same x and y coordinates:

**K(4, 3)**

The Laser continues in the same direction until it hits a Piece. If it hits the mirrored side of a Piece, it will change directions and continue. If the Laser hits the King, the game is over. If the Laser hits a non-mirrored side of a Piece or goes out of bounds, it stops. For example:

**(L2,3,E) ∧ P2,3,NW)) L2,3,N**

A Piece may only move to one adjacent square or rotate 90 degrees.

The 4 pieces must be on the board.

Only one piece can be at a certain position at a time.

# Model Exploration

In our model, we made a 5x4 board by giving each piece and King their own grid of Booleans. The board returns true if there is a Piece at a specified position. Our goal was to try to make code that checked if there was a way to win from a series of test cases, by moving one of the pieces.

We wrote code for a handful of classes: one for the Pieces, one for the Laser, and one for the King components. These all held the arrays of true or false statements for each piece, laser, and king. This was important because it was an easy way to represent the position of each component, and we could manipulate them with functions we made for each class (move up, down, etc). Then, we added all the constraints to play by the rules of the Laser Chess. We made more functions that worked to check every spot around each piece, and whether winning the game was possible after either moving a Piece one space or rotating it by 90 degrees.

Upon further exploration, the game original rules of Laser Chess created more and more issues due to its complexity. This meant that we had to limit our variables. Originally, we wanted to simulate a game against an artificial intelligence. Upon further exploration, this path became extremely difficult to implement. Because of this, we decided to eliminate an opponent entirely. Furthermore, we had to lock the King position, because it was became very tedious to write out every possible move.

# Jape Proofs

First:

Graphical user interface, application, table

Description automatically generated

In this first proof, the propositions represent different spots in one piece’s domain (it’s grid). P1 would be one x and y coordinate, and P2 would be another. They both can’t be true, but one or the other can and they can both be false. It’s a very simple concept but important for the basis of how the pieces work in the game, because one piece can’t be at multiple spots on the board.

Second:

Table

Description automatically generated

In the second proof, P is true if a piece is at a certain spot on the board, E is true if that spot is empty, Q is true if the king is at that spot, A is true if the laser is at the current spot, B is true if the game is won, D is true if the laser is changing direction at the current position. The propositions are: there must be either a piece, a king, or the spot is empty, there is a laser at the spot, if the king and the laser are at the spot the game ends, the direction does not change, if there’s a piece at the spot there is a direction change, the spot is not empty. This is important because it is the win condition: it’s trying to prove a game over.

Third:

Text

Description automatically generated

A is true if there is a laser at a specific spot on the grid, T is true if this spot is actually the edge of the board(hits the edge as if to go beyond, after any obstacles on the actual board), Q is true if the king is at the position, C is true if the laser will continue after this position and B is true if the game is won. The propositions are as follows: there is a laser at this spot, it is hitting the edge of the board, if this is the edge, then it can’t be a king piece (because it’s technically not on the board), if there’s a laser hitting the edge, then the laser must stop, if the laser must stop and it’s not hitting a king this means there is no win. All of these propositions prove that there is a loss.

# First Order Extension

One thing we can use in predicate logic is coordinates, and it make a lot more sense where it’s a grid of true and false values.

For all pieces, they should have a x and y and a direction, E.g. P(x,y,d), true where that piece is and what direction it’s facing

The king should have only and x and y, K(x,y) representing where the piece is when it’s true

The laser, similar to the piece, has a position and a direction L(x,y,d)

All of the x, y and d values have different domains, and the values in the propositions must stay within these domains.

Game ends at if L(x,y,d) and K(x,y) is true (share same x and y values) or if the laser is going in the direction of the border as it’s in the outside ring of the grid(facing the edge)

For example, trying to express that a piece can’t be at any other spot can be expressed as xy.i.zr.j.((P(x,y,i) P(z,r,j)), but this is only true for accessing DIFFERENT positions, where (x,y,i) does not = (z,r,j). There isn’t really a way to write the exception in logic. For example, if there’s a piece P(1,2,3), then that piece can’t exist at P(3,1,2), but of course there is a piece at P(1,2,3) because we just stated there was.

Another extension that could be done would be writing that there always exists a piece position: . Each piece has a spot that it is true at, and as we saw in the last example no more than one.

Explaining that there is either a piece, a king, or a spot is empty can be shown by: xy.((P(x,y) ) ), E representing a set of empty spots represented by true when it’s empty. This is logically equivalent to an exclusive or, where only one can be true for each x and y value.