Video game problem

After thinking for awhile ,for this problem my solution is to describe all walls as vector (A,B) and players' position as point (x,y). Hence the binary file which stores game data should have below format/structure

width	height	nOp	nOw	checksum	data

where :

width : the width of 2D game scene with resolution can be ,let say 1 pixel.
Height : the height of 2D game scene
nOw : number of walls
nOp :numbers of players
since a wall is a line segment as mentioned in the problem, I use a vector (A,B) where A ,B are a points.
A point is simply an array with (x,y) similar to Cartesian coordinate, x,y are integer.
Checksum : this is an option and wasn't implemented yet, it should contain the polynomial checksum of data .
Data : this field contain an arbitrary number of walls and players position which were described above.

The source code contain a load game function, a save game function, a add/remove wall function and finally a light of sight function to check whether 2 players can see each other with a given wall.
The math behind Light of sight algorithm is derived from line-line intersection I read from wikipedia [1] and it should work well. One can summarize it as simple as below :
given 2 line segment AB and CD where A(Xa,Ya) , B(Xb,Yb),C(Xc,Yc),D(Xd,Yd).
A line equation is y=ax+b, and with 2 point A,B we have a linear equation
Ya=a*Xa+b, Yb=a*Xb+b
this equation can be easily solved using high school math or matrix manipulation.
Similar to CD : Yc=c*Xc+d, Yd=c*Xd+d
An intersection (if have) between AB, and CD will satisfied those 4 equations.
And 1 more thing , the intersection should belong to both line segments.
In case 'a' equals to c (the slope), both line should be parallel or overlaps hence there's no intersection.
In conclusion I will calculate t and u, if 0<=t<=1 and 0<=u<=1, then AB and CD have 1 intersection, otherwise A can see B.
Please check my source code which was written in C (for easier to handle binary file and data) for more detail.
To compile the source code on Linux, open the terminal and run following command :

$gcc test3.c -o test3

[1] https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection

Note that the intersection point above is for the infinitely long lines defined by the points, rather than the line segments between the points, and can produce an intersection point not contained in either of the two line segments. In order to find the position of the intersection in respect to the line segments, we can define lines L_1 and L_2 in terms of first degree Bézier parameters:

$$L_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + t \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix}, \qquad L_2 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + u \begin{bmatrix} x_4 - x_3 \\ y_4 - y_3 \end{bmatrix}$$

(where t and u are real numbers). The intersection point of the lines is found with one of the following values of t or u, where

$$t = \frac{\begin{vmatrix} x_1 - x_3 & x_3 - x_4 \\ y_1 - y_3 & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_4 \\ y_1 - y_2 & y_3 - y_4 \end{vmatrix}} = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

and

$$u = \frac{\begin{vmatrix} x_2 - x_1 & x_1 - x_3 \\ y_2 - y_1 & y_1 - y_3 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_4 \\ y_1 - y_2 & y_3 - y_4 \end{vmatrix}} = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)},$$

with:

$$(P_x, P_y) = (x_1 + t(x_2 - x_1),\ y_1 + t(y_2 - y_1)) \quad \text{or} \quad (P_x, P_y) = (x_3 + u(x_4 - x_3),\ y_3 + u(y_4 - y_3))$$

There will be an intersection if $0.0 \le t \le 1.0$ and $0.0 \le u \le 1.0$. The intersection point falls within the first line segment if $0.0 \le t \le 1.0$, and it falls within the second line segment if $0.0 \le u \le 1.0$. These inequalities can be tested without need for division, allowing rapid determination of the existence of any line segment intersection before calculating its exact point.[2]