Pathing Barriers

The purpose of pathing barriers is to erect invisible barriers in the game world that prevent pathing through them.

If a path, created for example when a player casts blink, intersects a barrier that path will be cut short in front of the barrier. The barriers can be applied in several ways although the main purpose when they were added was to erect barriers inside of doors to prevent players from getting through them.

What the barriers won't do

A barrier will only block movement controlled by pathing i.e. if the player manually runs, using keys like wasd, through the barrier that movement will not be blocked.

To prevent a player from getting through a door it is therefore crucial that the barrier is placed inside of the door or just behind it. If the barrier is placed inside the door the player's client's collision detection will prevent the player from running through the object manually and the barrier will prevent for example blink or charge.

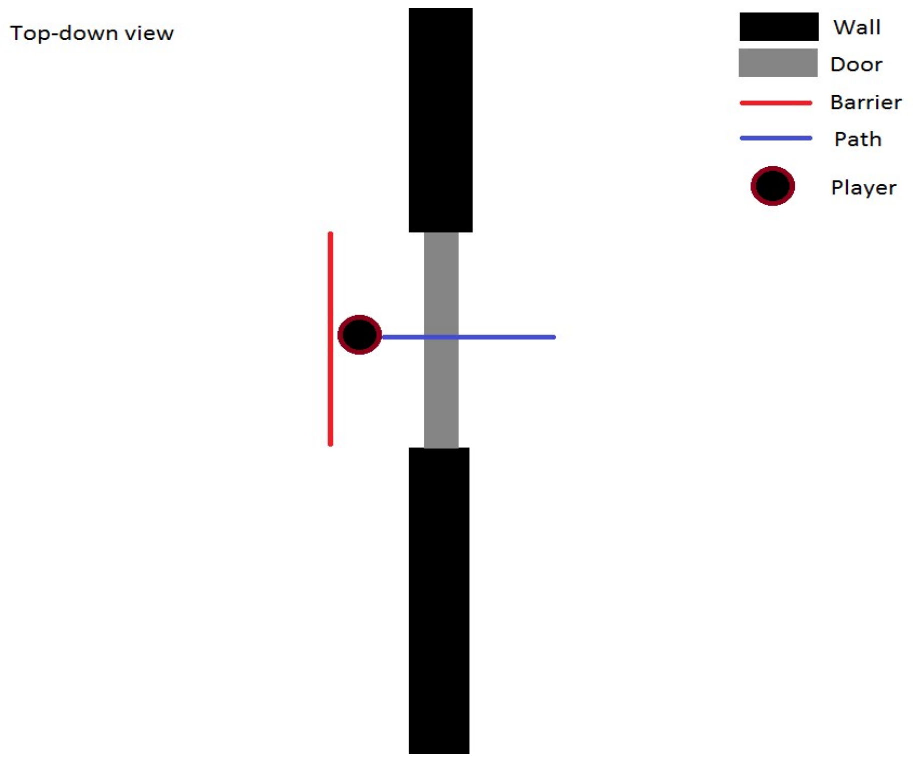

Illustration 1: The player can manually pass through the wall.

In Illustration 1 the player has walked through the barrier manually and the path created by his blink passes through the door unhindered. Since the wall is infinitely thin, it's in 2D, the player can pass the barrier with the least significant decimal in his coordinates and it'll count as being through it.

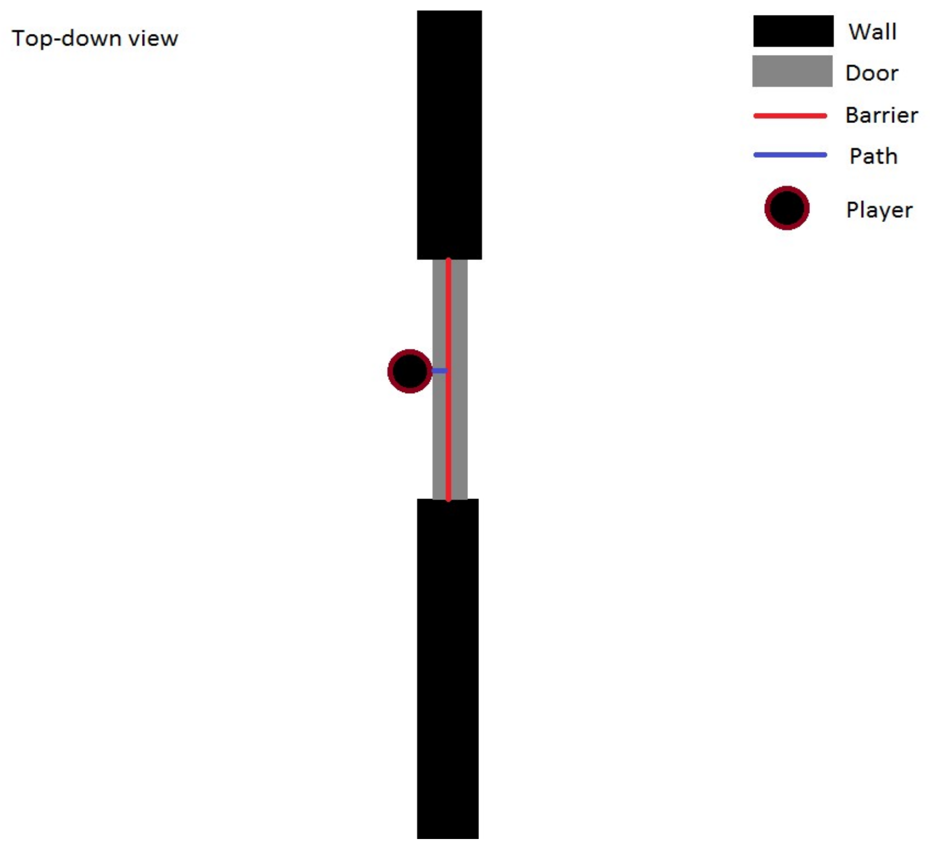

Illustration 2: The client's collision detection prevents the player from passing through the barrier.

In the case of Illustration 2 the player is running against the door but thanks to the client's built in collision detection the player is unable to cross the barrier and hence the path created by for example blink is cut short and the player won't get through.

If the player were to stop inside of the door, the path being cut short there, the client's collision detection would break and the player would be able to just run from the inside of the door to any side he chooses. To prevent this from occurring a player that hits a barrier will bounce back about 3 yd and hopefully land outside the object it's protecting.

Erecting a barrier

Erecting a barrier is a simple process of adding three sets of x-, y- and z-coordinates and the current mapid to the pathing_barriers table in the mangosd database. A guid may also be added to the entry to connect a barrier to a door, thus making sure the barrier is disabled when the door is opened.

Each barrier is made up of two vectors forming it's sides and the area covered will be, in mathematical terms, a positive linear combination of the two vectors up to the vectors' length (See Equation 1).

$$P=\lambda_1 \vec{\boldsymbol{v}}_1+\lambda_2 \vec{\boldsymbol{v}}_2, 0\leq\lambda_1\leq 1, 0\leq\lambda_2\leq 1$$

Equation 1: The linear combination of the barrier's sides.

In Illustration 3 the three sets of coordinates (x_0, y_0, z_0), (x_1, y_1, z_1) and (x_2, y_2, z_2) create the two red vectors $\mathbf{v_1}$ and $\mathbf{v_2}$. The limits of the covered area is indicated by the two blue vectors that are the two created vectors moved along themselves to the opposing end of their neighbour.

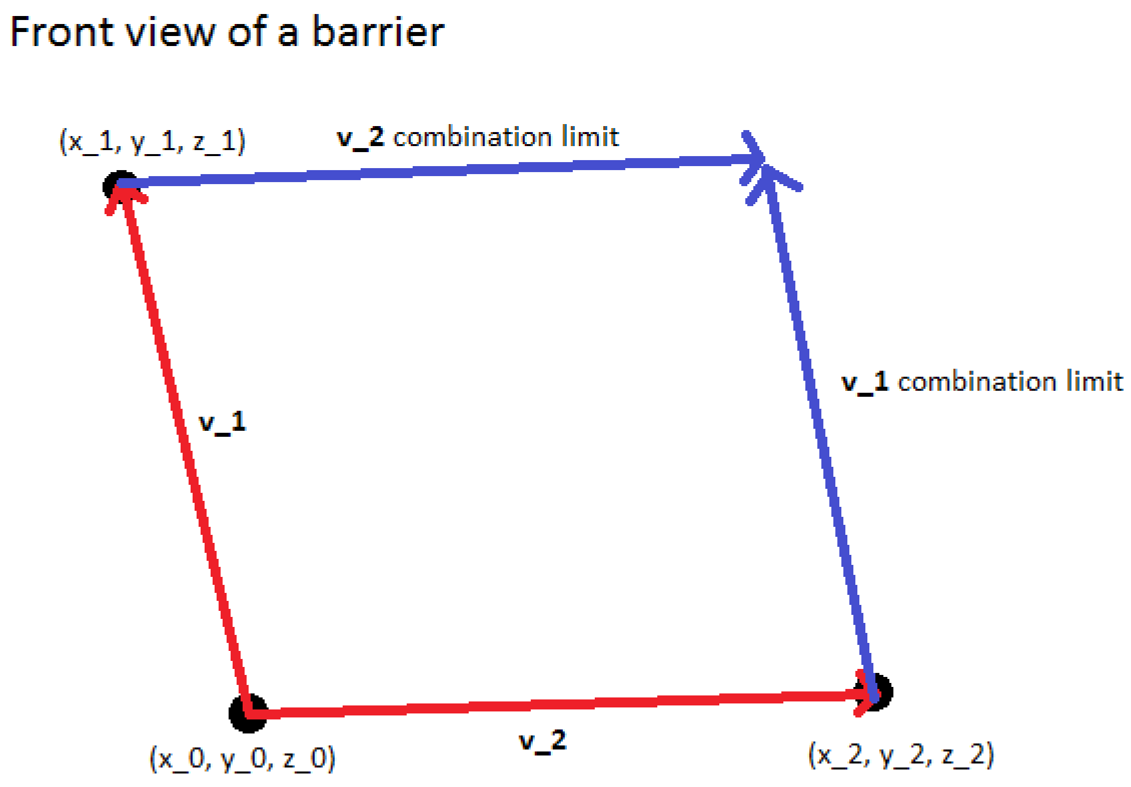

Illustration 3: The created vectors v_1 and v_2 (red) and the resulting barrier limits (blue).

When a path is drawn a check will be performed to see if it passes inside the area that is framed by the red and blue vectors that make up a 2D plane in the 3D game space.

Adding the barrier to the database

To add a barrier to the DB choose a common point for the two vectors, the (x_0, y_0, z_0), and the two end-points, (x_1, y_1, z_1) and (x_2, y_2, z_2), by for example using the global coordinates given by the .gps command. Also note down the mapid to make sure the barrier ends up in the correct map and if there is a door that should disable the barrier be sure to add its guid, which can be ascertained with the command .gobject near. An example of an insert command for a barrier at the first door to the left in BRD follows below:

```
INSERT INTO `pathing_barriers` (`id`, `guid`, `map`, `x0`, `y0`, `z0`, `x1`, `y1`,
`z1`, `x2`, `y2`, `z2`, `comment`) VALUES (8, 15617, 230, 496.25, 16.1, -75,
496.25, 16.1, -50, 496.29, 12.39, -75, 'The first door to the left in BRD.');
```

The points chosen for the door is approximately shown in Illustration 4, although the height coordinate (z-coordinate) was manually adjusted up/down for the different points to be certain that the entire height of the door was covered.

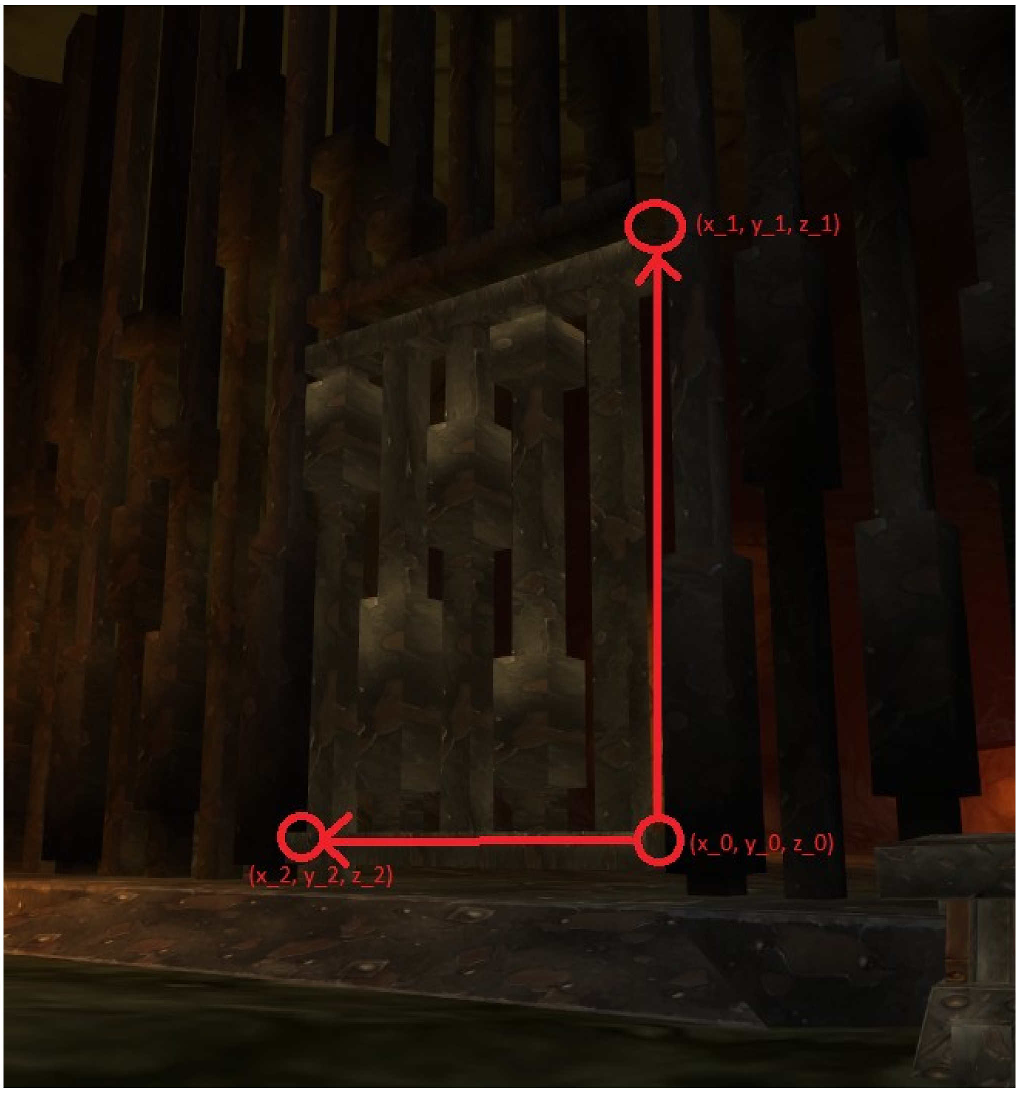

Illustration 4: Approximately chosen points at the barrier in BRD.

The standard method for setting up a barrier for a door is the following: Choose one point at a corner that is the common point, then place the other two points at the opposing corners and adjust the z-coordinates to make sure the entire door is covered. In the case of Illustration 4, and most doors, it is wise to set z_0 and z_2 to the same value to get a levelled limit at the door's bottom.

The code

The math behind the code

The main workhorse of the pathing barriers is the function `IsLineIntersecting2DPoly` in pathfinder.cpp. The function takes all the three points for the barrier in an array as well as the previous path point and the current path point and creates the two vectors for the barrier as well as a vector describing the direction of the current path. The points are then plugged in to the general solution to the parametric equation describing an intersection between the current pathing vector and the barrier, see equation below.

$$\boldsymbol{P}_{path} = p\begin{pmatrix} x_4 - x_3 \\ y_4 - y_3 \\ z_4 - z_3 \end{pmatrix} + \begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix}$$

$$\boldsymbol{P}_{barrier} = t\begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \\ z_1 - z_0 \end{pmatrix} + s\begin{pmatrix} x_2 - x_0 \\ y_2 - y_0 \\ z_2 - z_0 \end{pmatrix} + \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

$$\boldsymbol{P}_{barrier} - \boldsymbol{P}_{path} = 0$$

From the solution to the parametric equation we get the values of the parameters p, t and s. If the value of the parameters are all $0 \leq p, t, s \leq 1$ then the path is intersecting the barrier. If the parameter is outside of the interval or not defined due to a singular equation there is no intersection.

The implementation

In the `PathInfo::BuildPointPath` in the file PathFinder.cpp and in the `MmapManager::DrawRay` in the file MoveMap.cpp a loop goes through all the barriers in the current map and checks if there are any intersections by calling `bool IsLineIntersecting2DPoly`. If the associated door is not open and there is an intersection found the path will be cut short and a new bounce-back point will be calculated and added to the path instead.

The function `IsLineIntersecting2DPoly` simply inserts all the different coordinates into the general solution to the parametric equation given in **"The math behind the code"** that was calculated by plugging the equation into Maple with the syntax below and returns a boolean indicating if an intersection was found.

```
solve({t*(x[1]-x[0])+s*(x[2]-x[0])+x[0]-p*(x[4]-x[3])-x[3], t*(y[1]-y[0])
+s*(y[2]-y[0])+y[0]-p*(y[4]-y[3])-y[3], t*(z[1]-z[0])+s*(z[2]-z[0])+z[0]-
p*(z[4]-z[3])-z[3]}, {p, s, t})
```

Written by: Andreas Kempe, andreas_kempe@msn.com
Last updated: 18 may 2013