Go Stone vs. Go Board

Collision detection between stones and the board

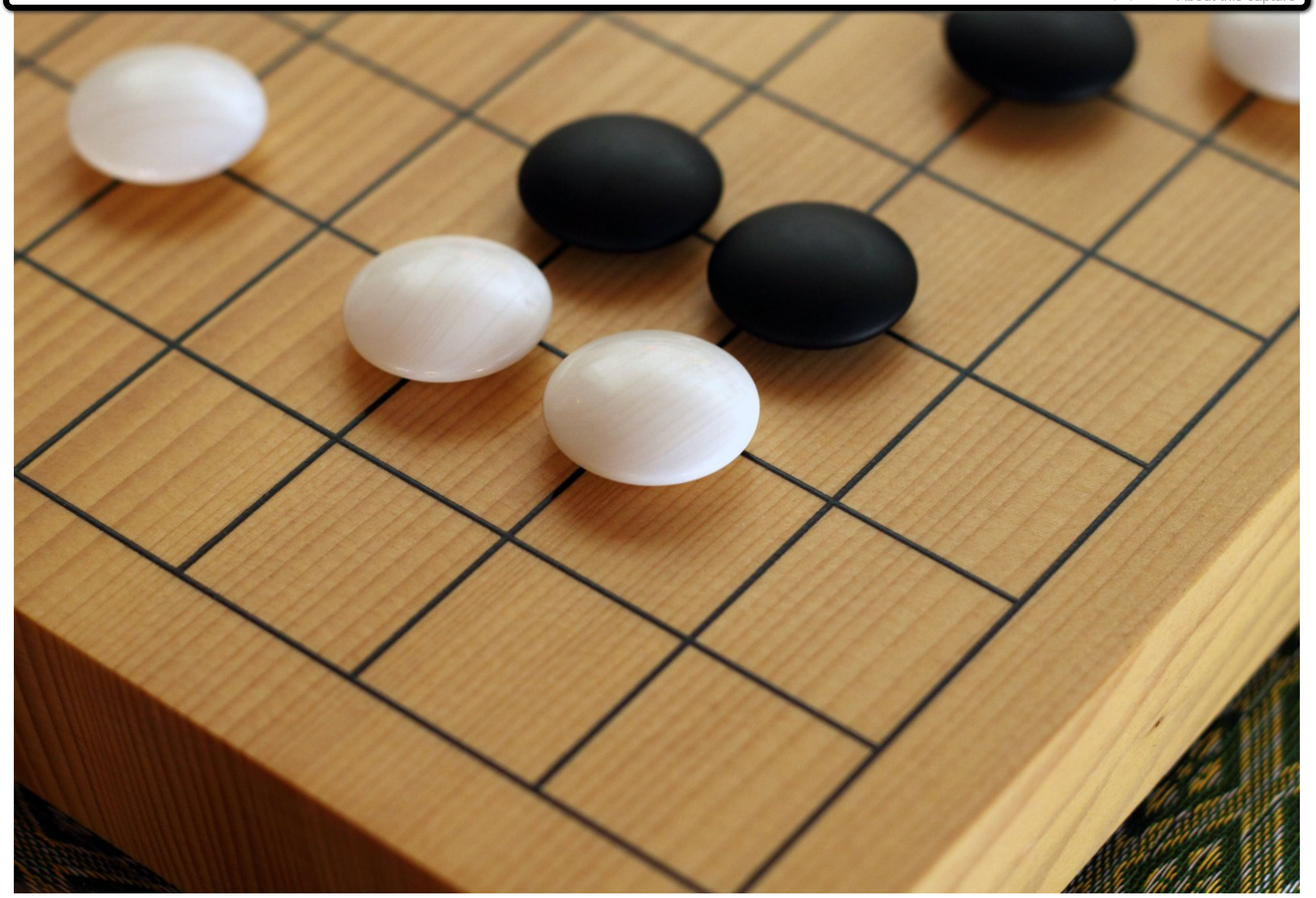
Posted by Glenn Fiedler (http://web.archive.org/web/20181107181450/https://gafferongames.com/about) on Friday, February 22, 2013

Introduction

Hi, I'm Glenn Fiedler. Welcome to **Virtual Go** (http://web.archive.org/web/20181107181450/https://gafferongames.com/categories/virtual-go/), my project to create a physically accurate computer simulation of a Go board and stones.

In this series so far we've defined the shape of a go stone, rendered it using 3D graphics hardware and simulated how it moves in three dimensions.

Our next goal is for the go stone to bounce and come to rest on the go board.



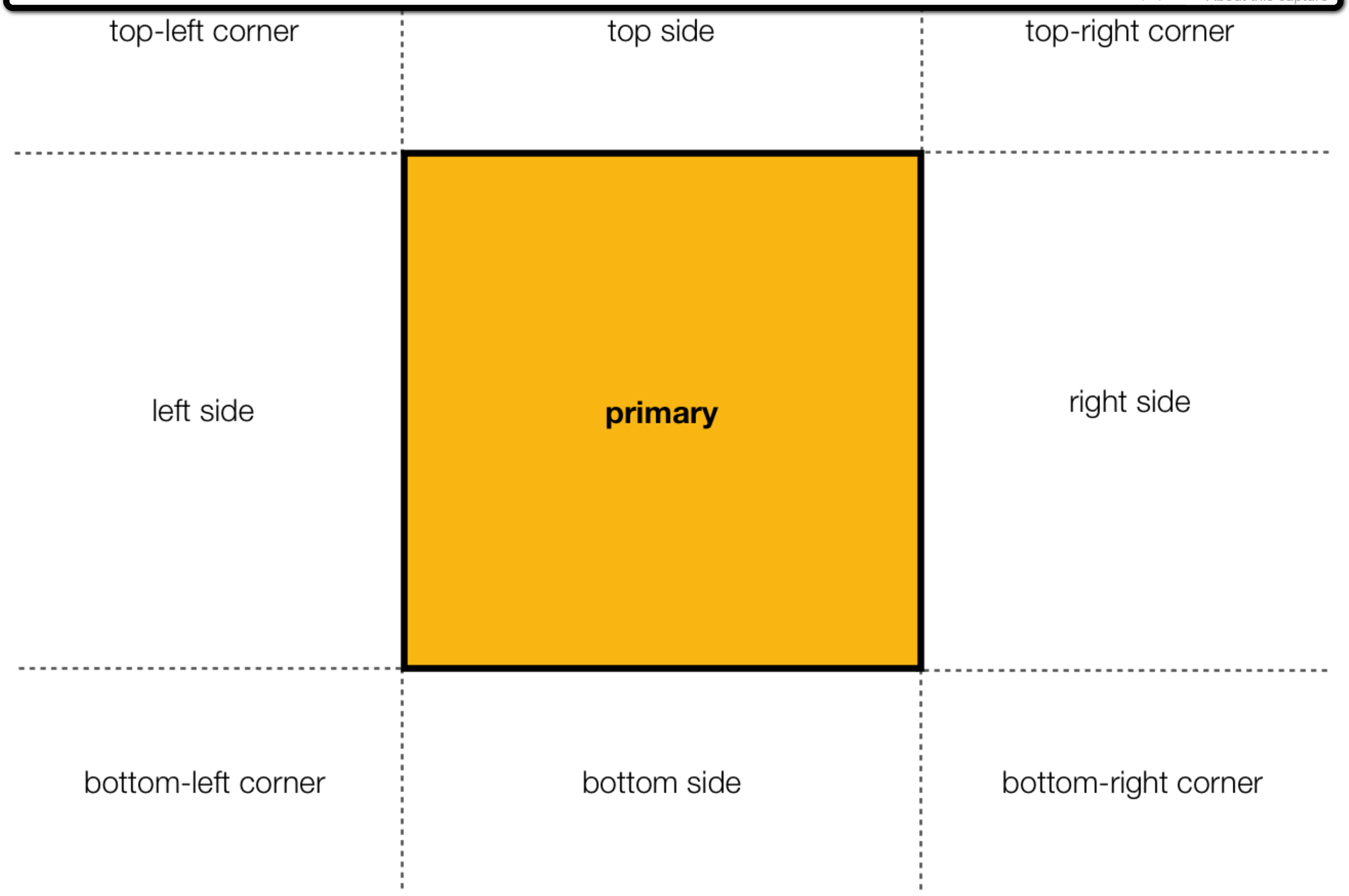
Understandably, this is quite complicated, so in this article we'll focus on the first step: detecting collisions between a go stone and the go board.

Voronoi Regions and The Minkowski Difference

First, lets assume that the go board is axis aligned and does not move.

Next, because go stones are small relative to the go board, we can break down collision detection into regions which are treated differently.

The common case is with the primary surface, the actual playing surface of the go board, so lets start by looking top-down at the go board and breaking it up into 2D voronoi regions.

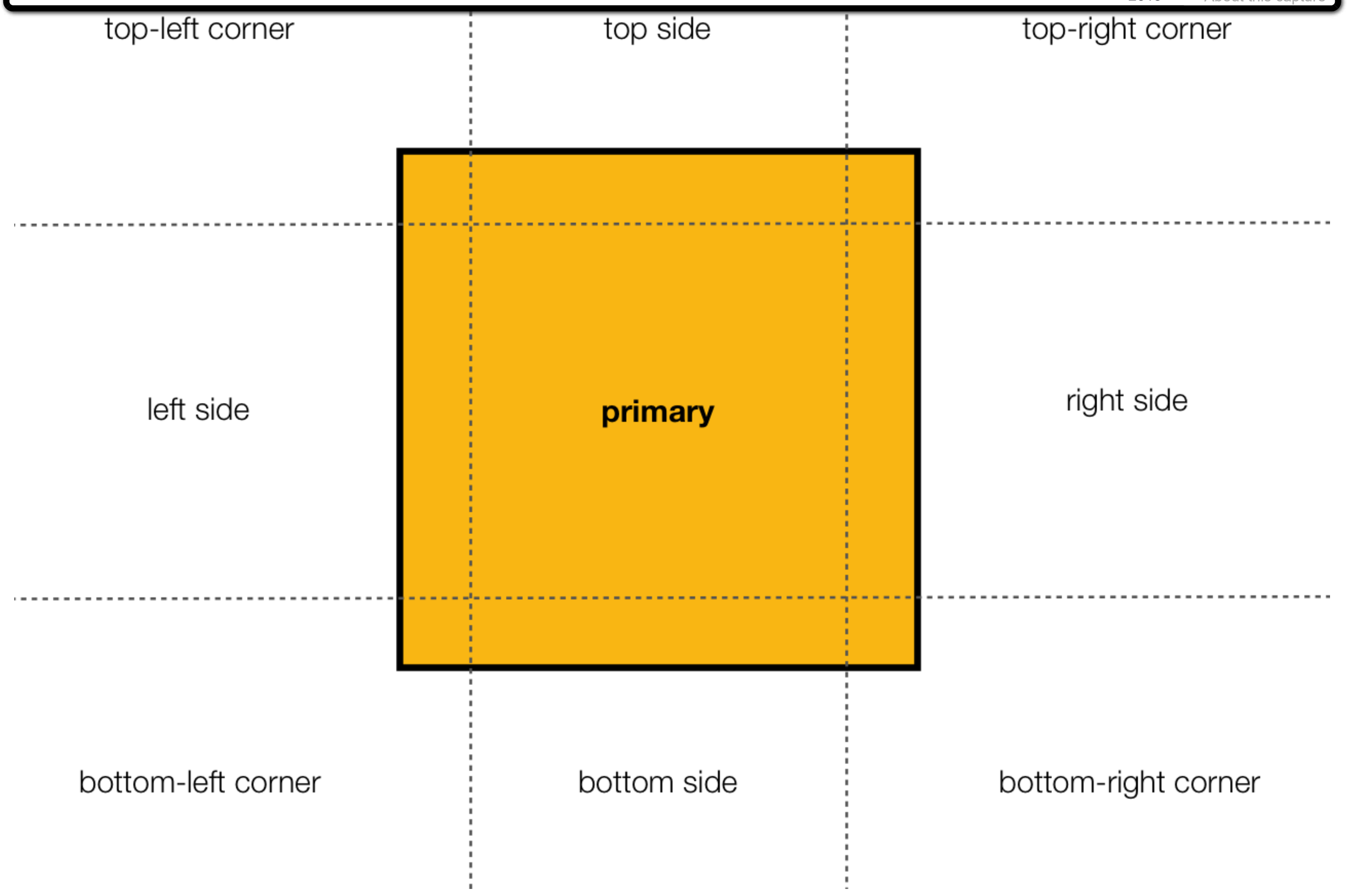


Each voronoi region corresponds to a subspace where all points (x,z) in that region map to the same nearest feature on the go board. This gives us one region that maps points to the top surface of the go board, four regions that map to the sides, and four corner regions.

If we were testing an infinitely small point against the go board, this would be enough, but we are colliding a go stone of a certain width and height.

One simple way to incorporate the dimensions of the go stone is to offset the regions from the edge of the go board by the the go stone's bounding sphere radius.

This creates something like a poor man's version of a minkowski difference:



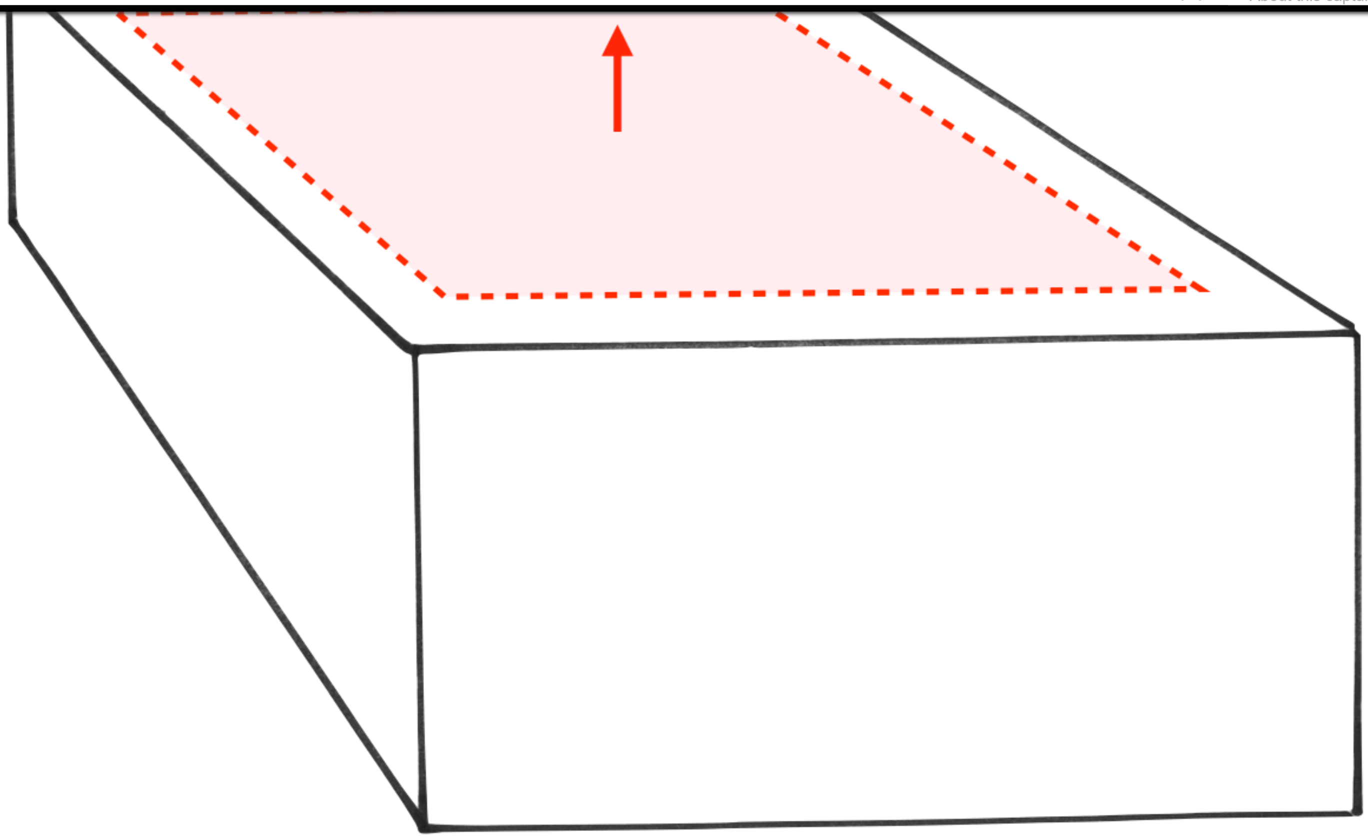
We can now test the center of the go stone against these regions to quickly to categorize the type of *potential* collision.

Go Board Collision Cases

Although the go board has nine different regions there only three unique types:

- Primary
- Edge
- Corner

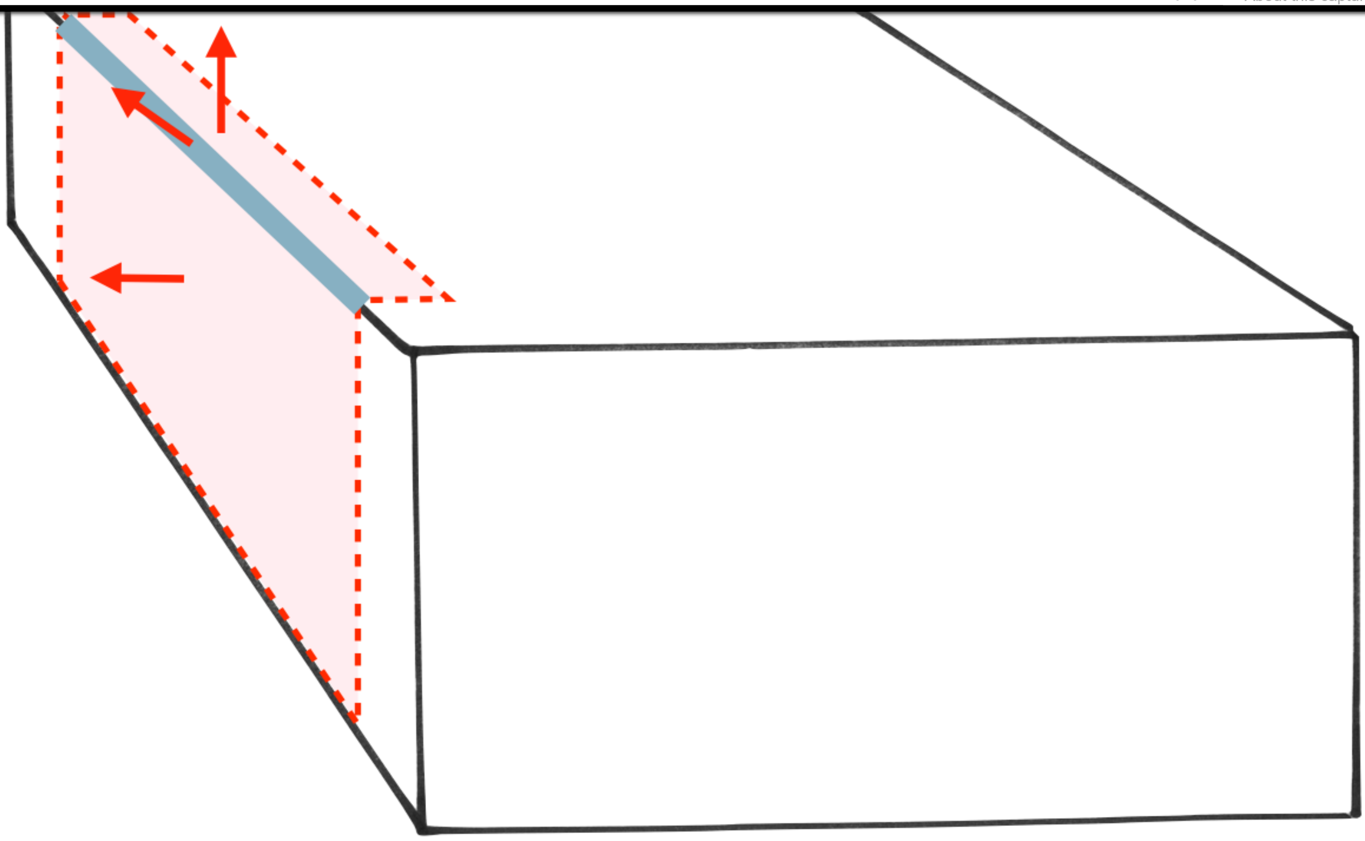
Primary is the common case.



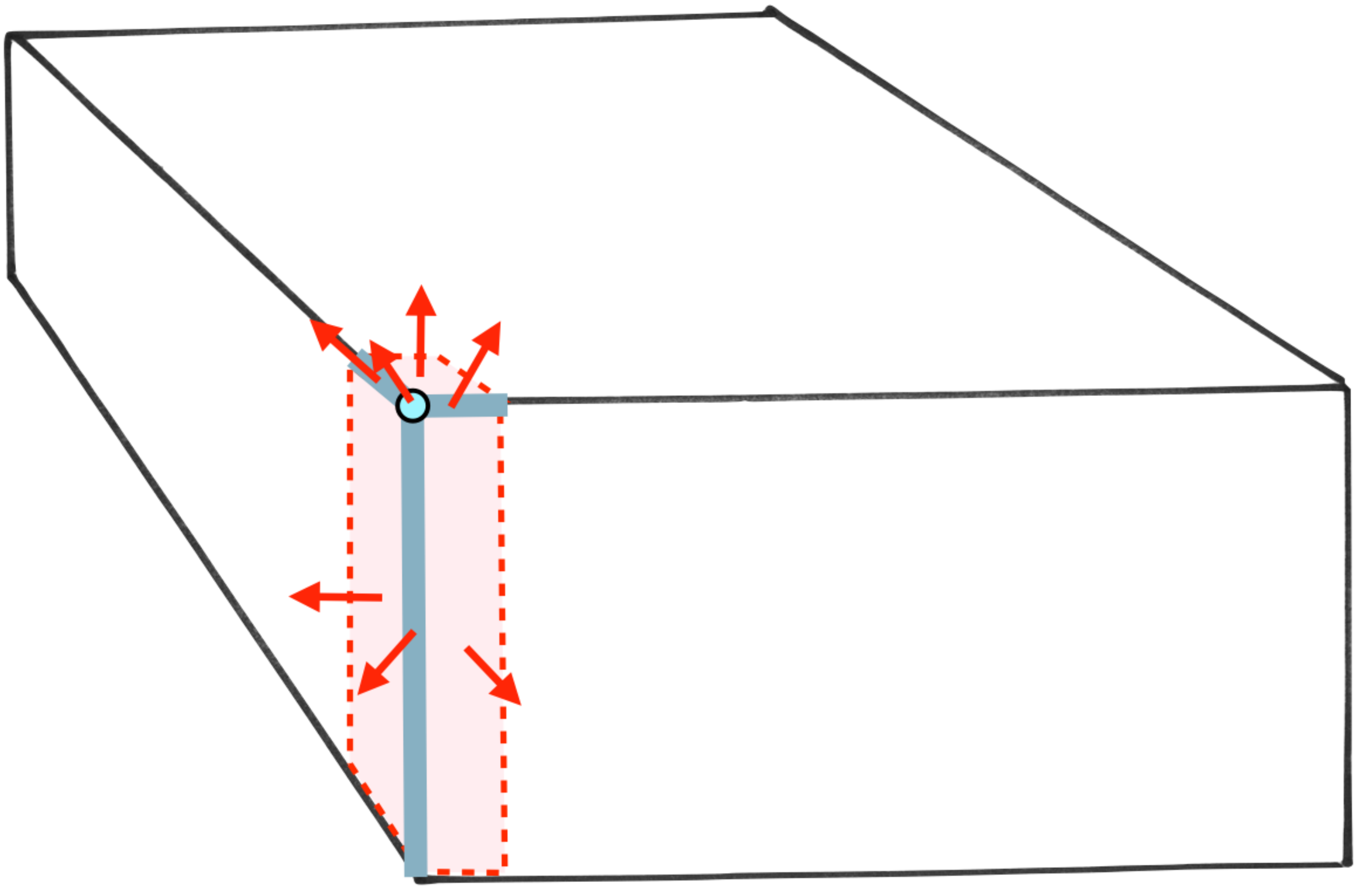
It’s also the easiest to handle. The only possible collision is between the stone and the playing surface of the go board.

Since the go board rests on the floor and cannot move we do not need to worry about collisions with the bottom surface. This means that we can consider the go board to be infinitely thick. This is extremely useful because it removes the possibility of fast moving go stones tunneling vertically through the board.

Next is the edge case. This is more complicated because there is more than one way to collide in edge regions. Tests must be done between the go stone and the top plane, the side plane, and the side edge.

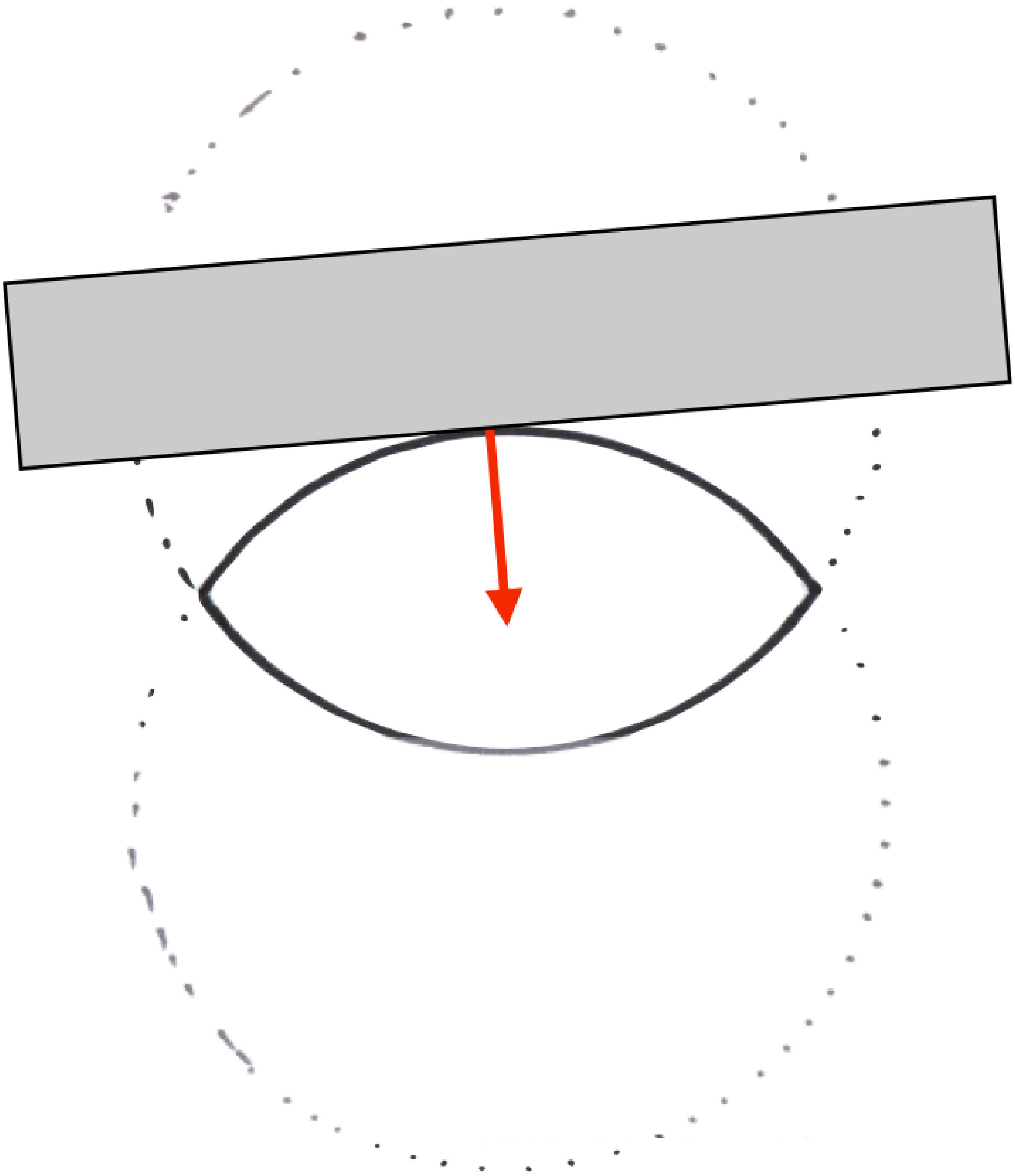


The corner case is more complicated still. Potential collisions include the top plane, the two side planes, the side edges adjacent to the corner, the vertical corner edge, and the corner point.

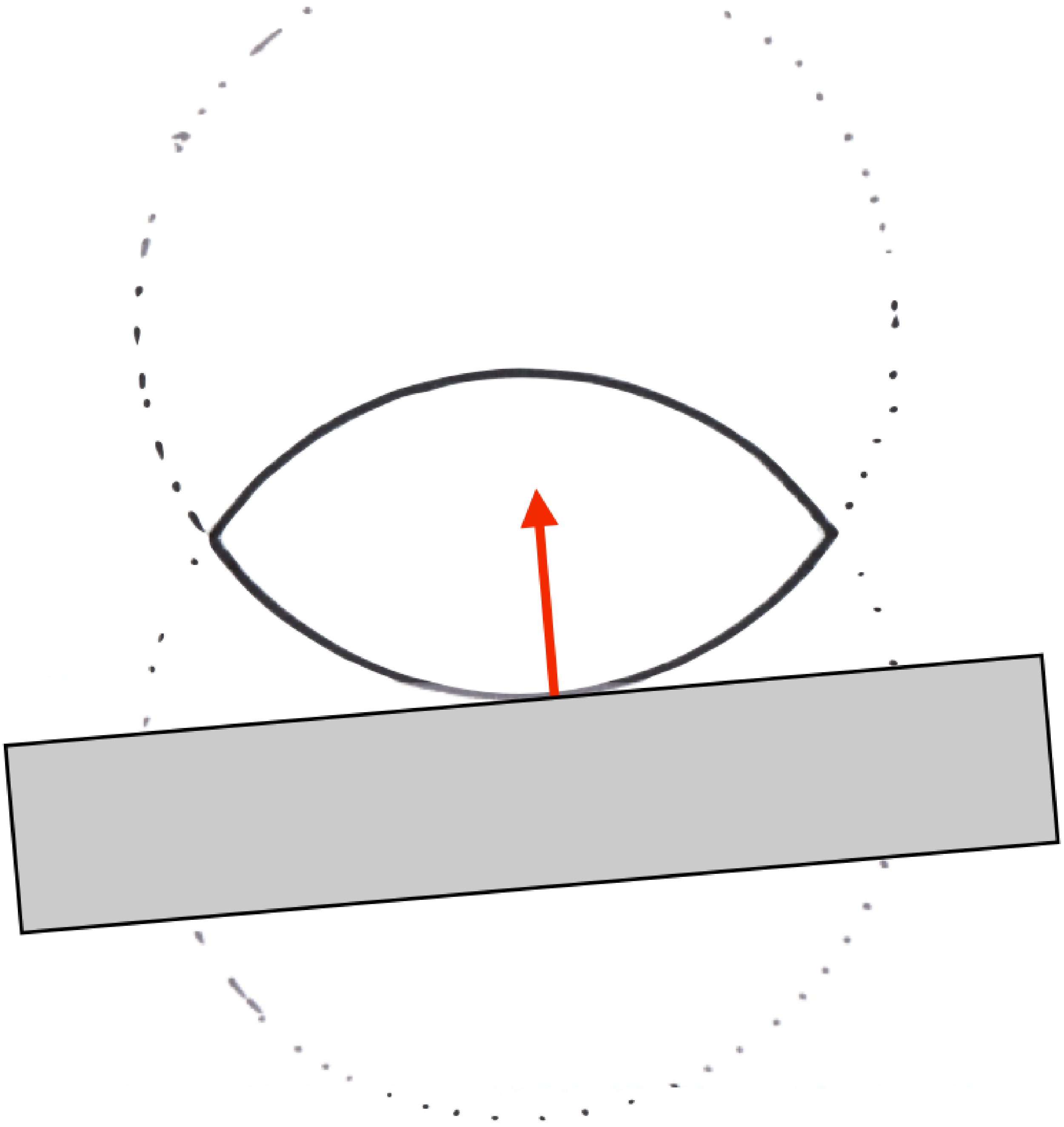


Go Stone Collision Cases

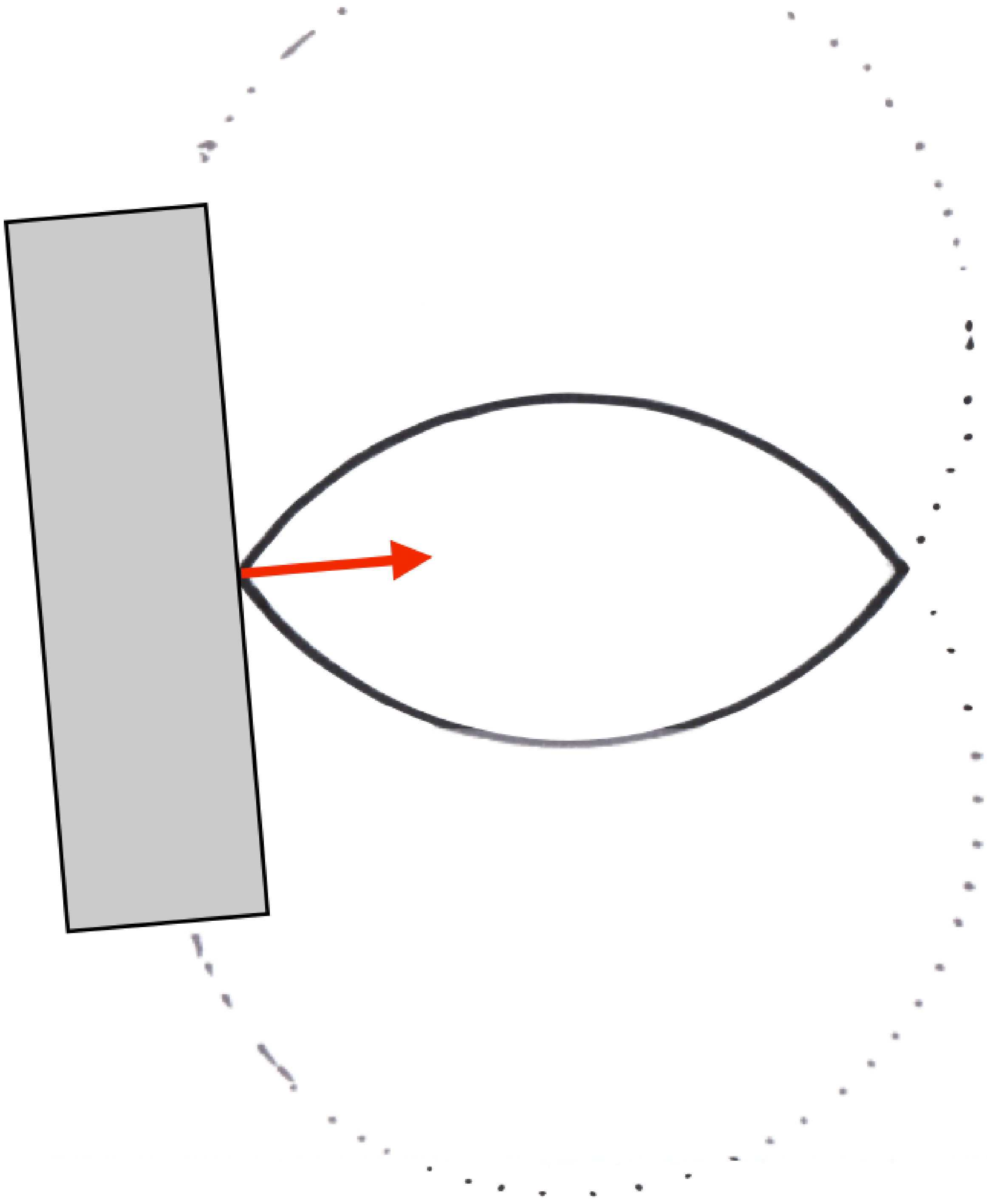
The first is a collision on the top surface of the biconvex. This corresponds to a collision with a portion of the bottom sphere that generated the go stone.



Next is the bottom surface of the biconvex. This corresponds to the top sphere.



Finally, the collision point can be on the circle ring at the intersection of the two sphere surfaces.



Separating Axis Test (SAT)

We have 3 ways a stone can collide with any convex object, and 9 different regions that must be treated differently when testing vs. the go board. Within each region we have up to 7 different features on the go board that must be tested against 3 different features on the go stone.

This is all rather complicated. How can we simplify it?

The solution is to use the separating axis test (http://web.archive.org/web/20181107181450/https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169).

between objects more general and less prone to combinatorial explosion.

Calculating The Support

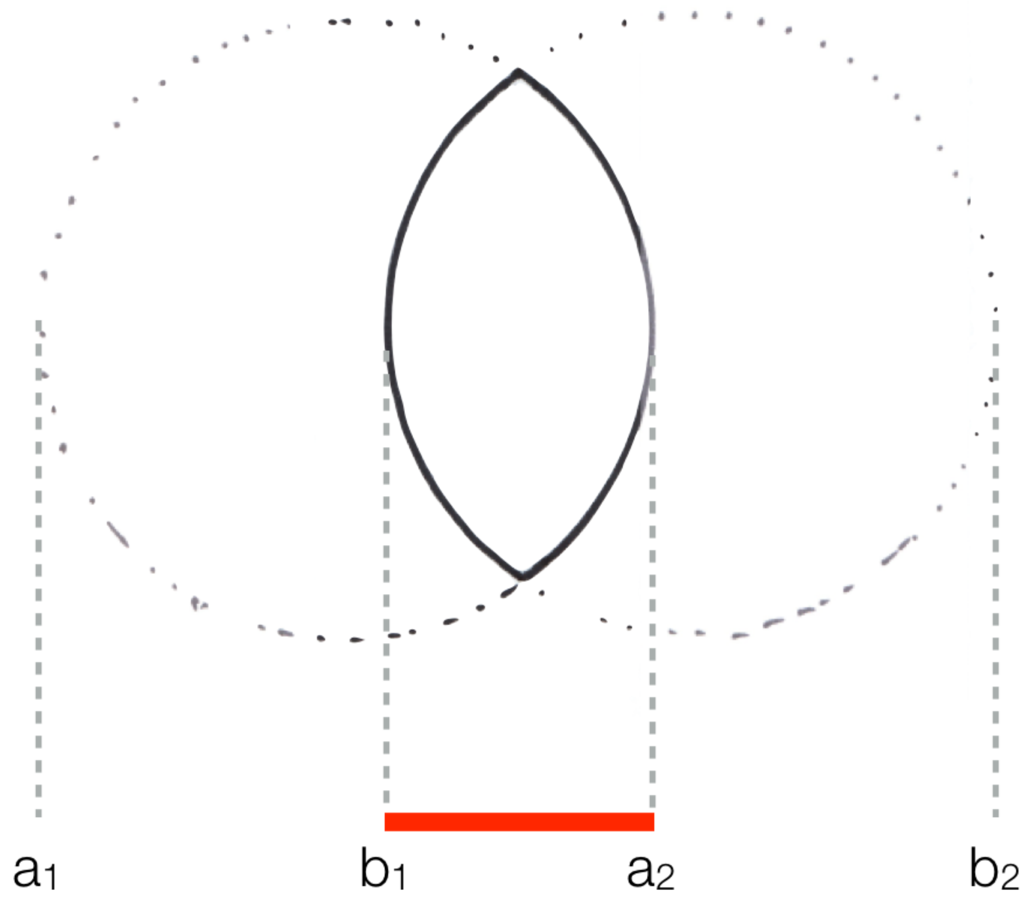
In order to use the separating axis test we must first write a function that determines the support of the go stone.

The support is the projection of an object on to an axis. This can be difficult to think about in 3D, but for me it makes it easier to think of the axis not as a line, but as the normal of a plane.

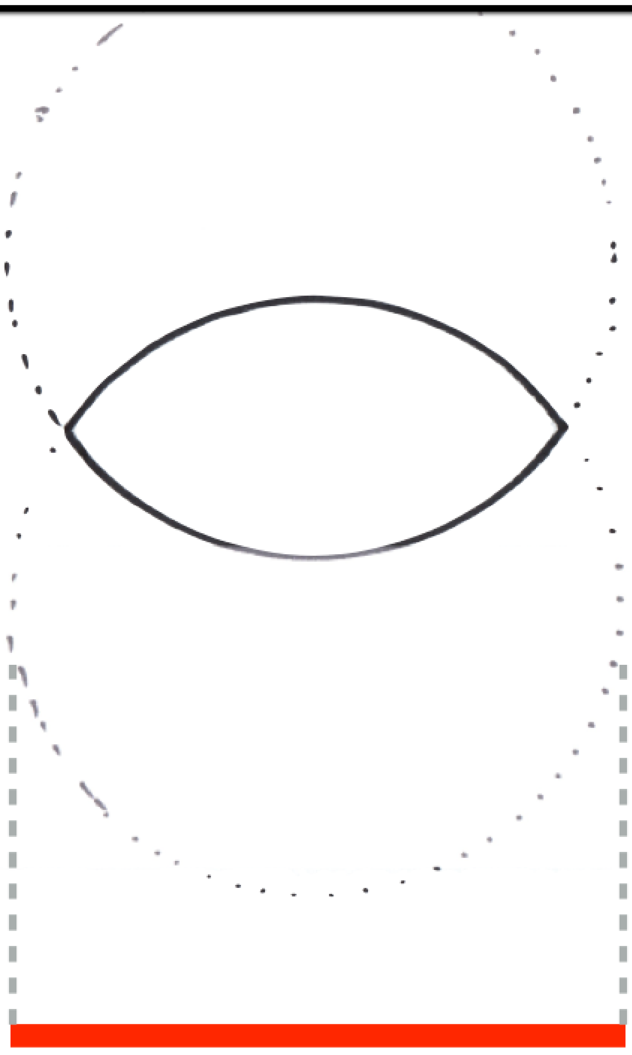
Then what we are really asking is: given this plane normal, what two planes from either side tightly bound the object like book-ends on a shelf?

To calculate the support of a biconvex solid we must consider two cases.

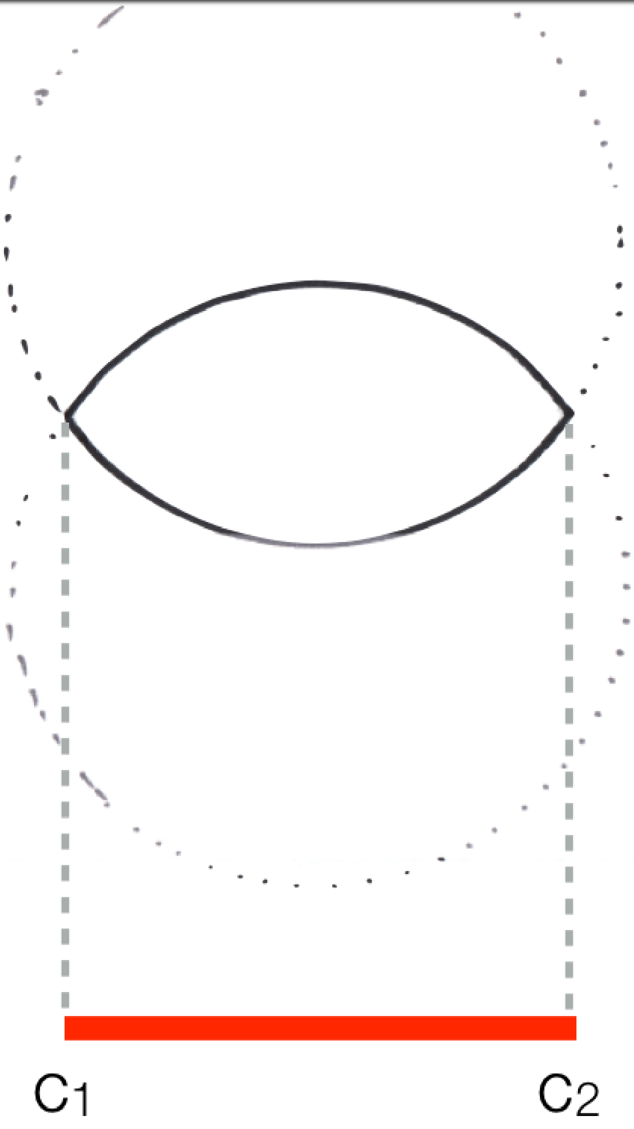
The first is when the go stone is vertical relative to the axis. Here it is reasonably easy. To calculate the support you simply calculate the intersection of the supports of the spheres used to generate the go stone. This makes a nice sort of intuitive sense seeing as the go stone is itself the intersection of two spheres.



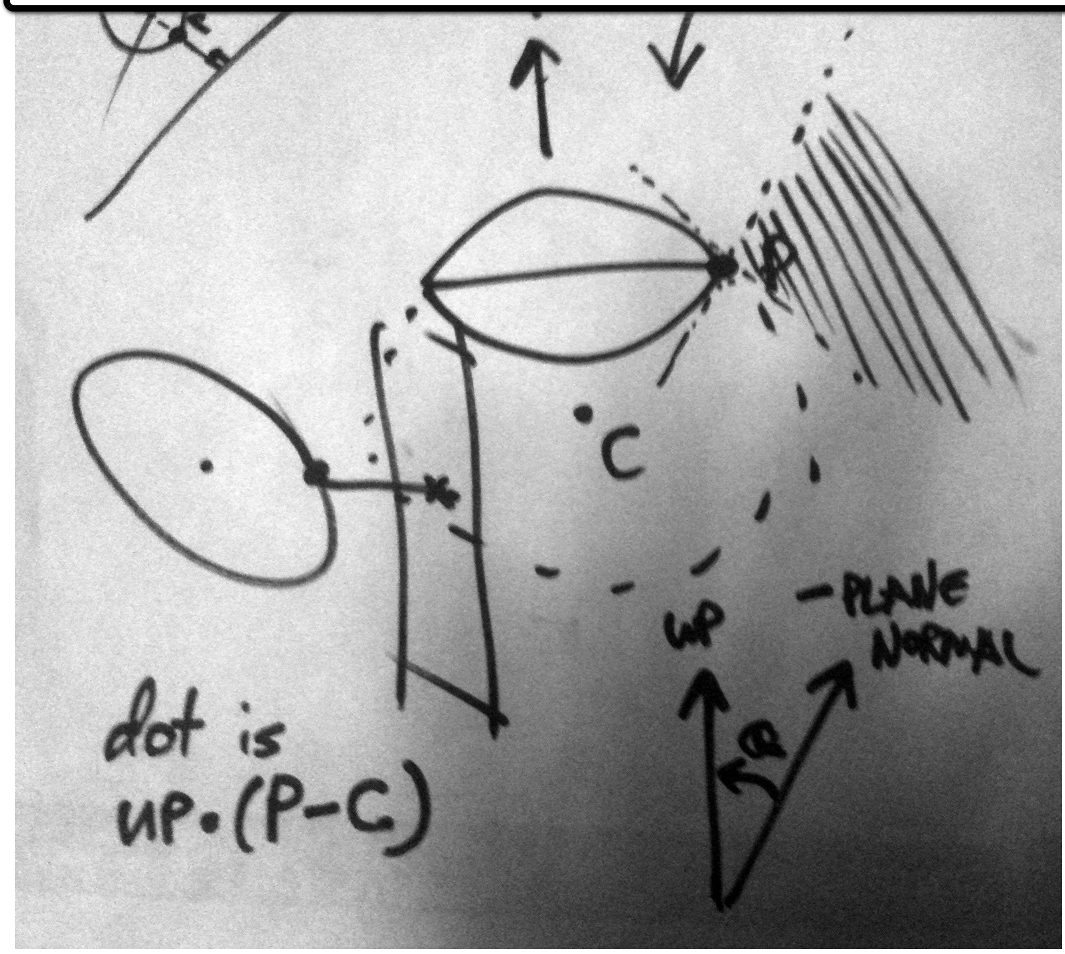
Unfortunately, this technique breaks down when the stone is horizontal relative to the axis because it fails to exclude the portion of the spheres that don't contribute to the biconvex solid.



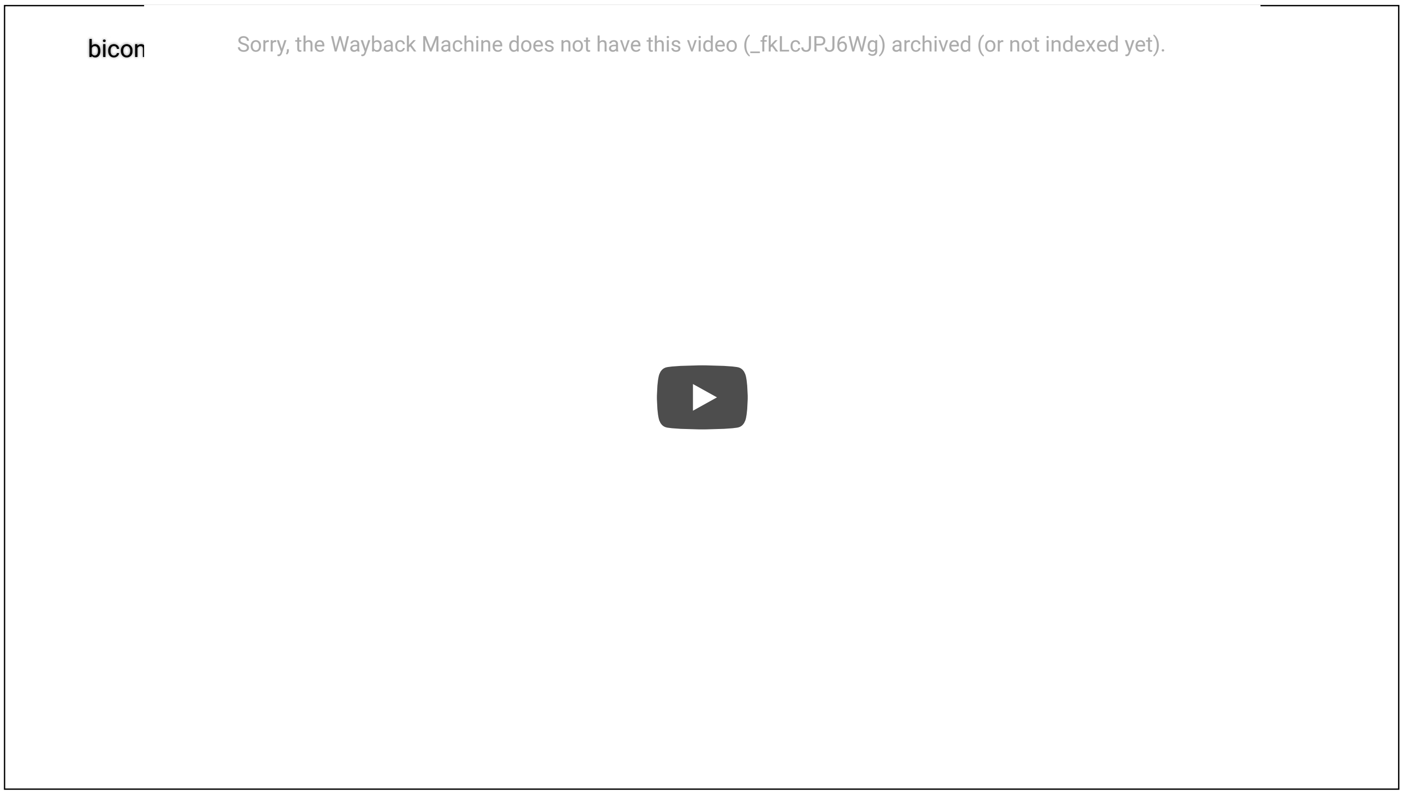
What you need to do instead is to calculate the support of the circle edge.



The tricky part is detecting when the transition between these two cases occur. Here's a diagram I created a while back when I first tried to work this out. If you look closely you can see the exact point where my head exploded:



And here's a visualization of the end result:


bicon
Sorry, the Wayback Machine does not have this video (_fkLcJPJ6Wg) archived (or not indexed yet).

Now we are ready to continue with the SAT for detecting collisions.

With this support we can use a one-sided variant of the SAT to detect collision with the primary surface. We're doing one-sided because we're treating the go board as 'infinitely thick' to avoid tunneling in the common case.

First, we take the normal of the primary surface which is (0,1,0) and find the support for the go stone using this normal as the axis: s_1 and s_2.

Next, we calculate the projection of the board surface along the normal: t

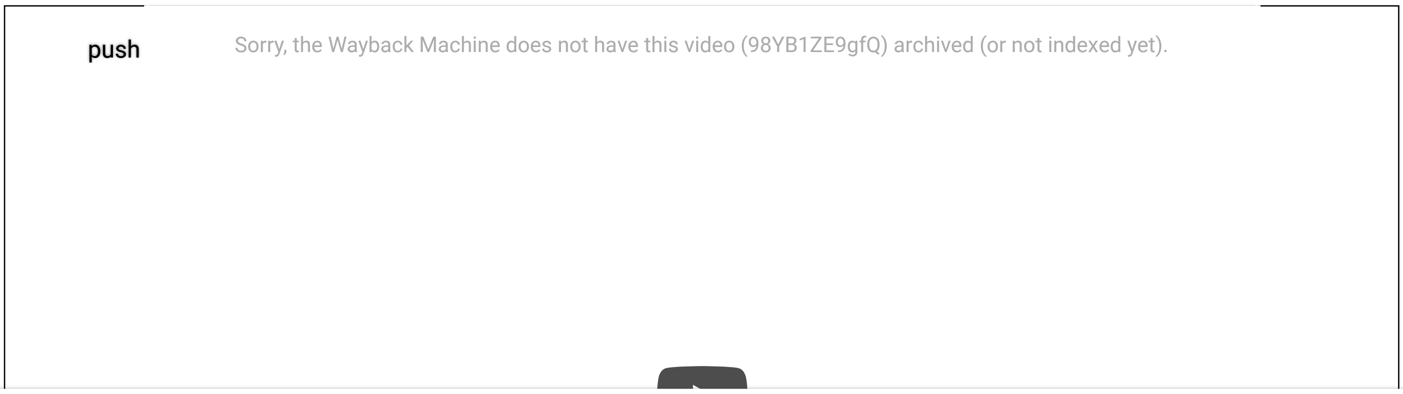
Then, if $s_1 <= t$ then the go stone is colliding with the go board:

go st
Sorry, the Wayback Machine does not have this video (br3wVa0CIis) archived (or not indexed yet).

Unfortunately, we detect the collision after the go stone has already penetrated the go board. There are many solutions for this problem: continuous collision detection (http://web.archive.org/web/20181107181450/http://jitter-physics.com/wordpress/?tag=continuous-collision-detection), and speculative contacts (http://web.archive.org/web/20181107181450/http://jitter-physics.com/wordpress/?tag=continuous-collision-detection) being interesting avenues I may explore later on.

But for now I just do the simplest and most pragmatic thing I can think of.

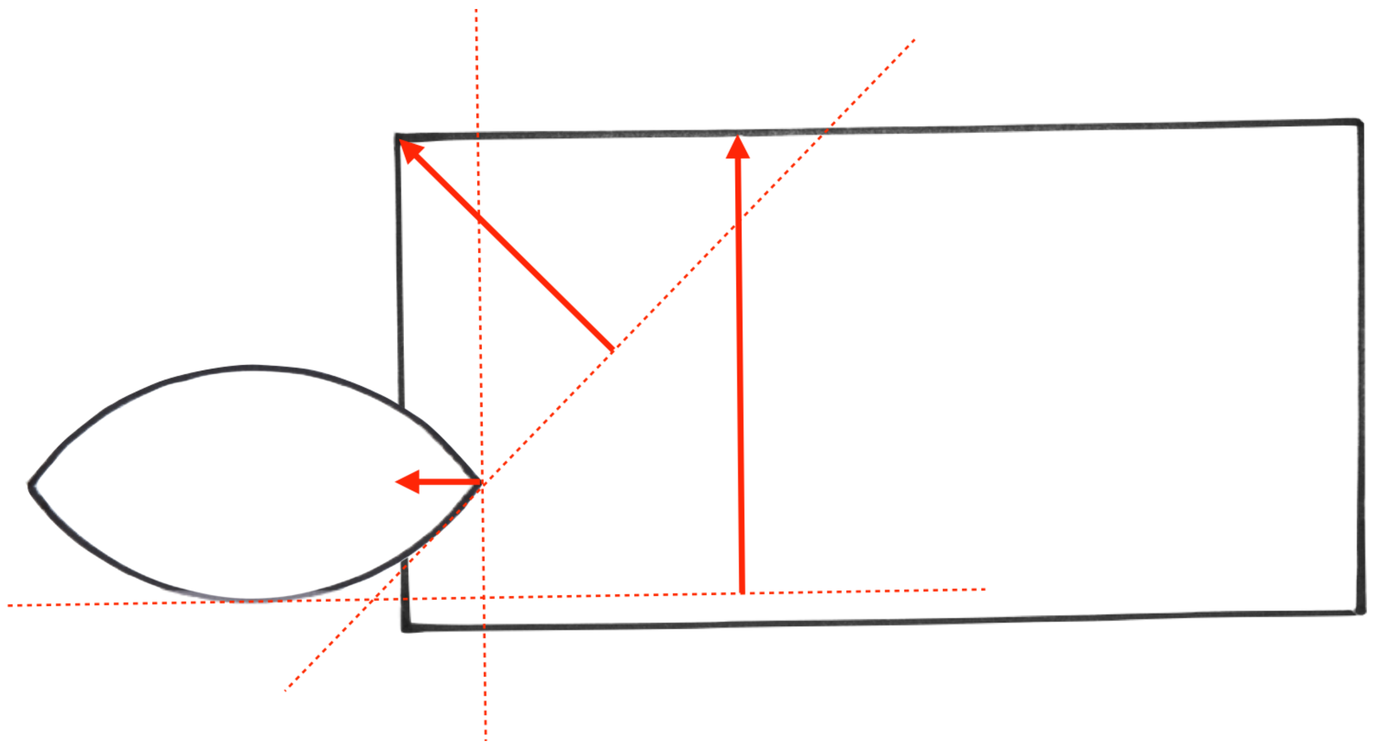
I just push the stone out of the board along the axis.


push
Sorry, the Wayback Machine does not have this video (98YB1ZE9gfQ) archived (or not indexed yet).

After I push the stone out, I recalculate the nearest point between the stone and board and use this as the contact point.

Edge and Corner Cases

The primary surface case is easy because only one axis needs to be tested, but in corner and edge regions multiple axes must be tested for collision.

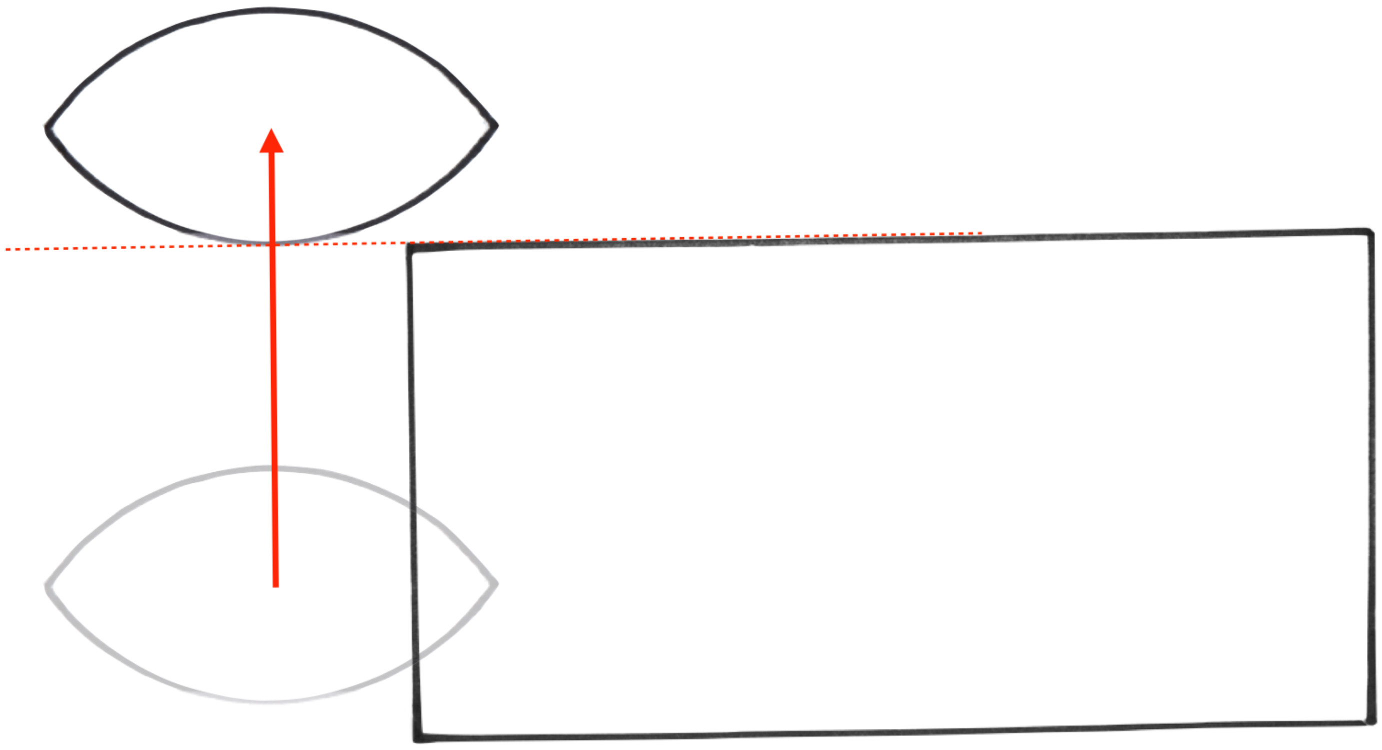


This is where the SAT really starts to shine. Now instead of combinatorial explosion testing each of the features of the go stone vs. each of the features on the go board, we flatten both the go stone and the go board into support and test for collision one axis at a time.

The separating axis test as applied as follows:

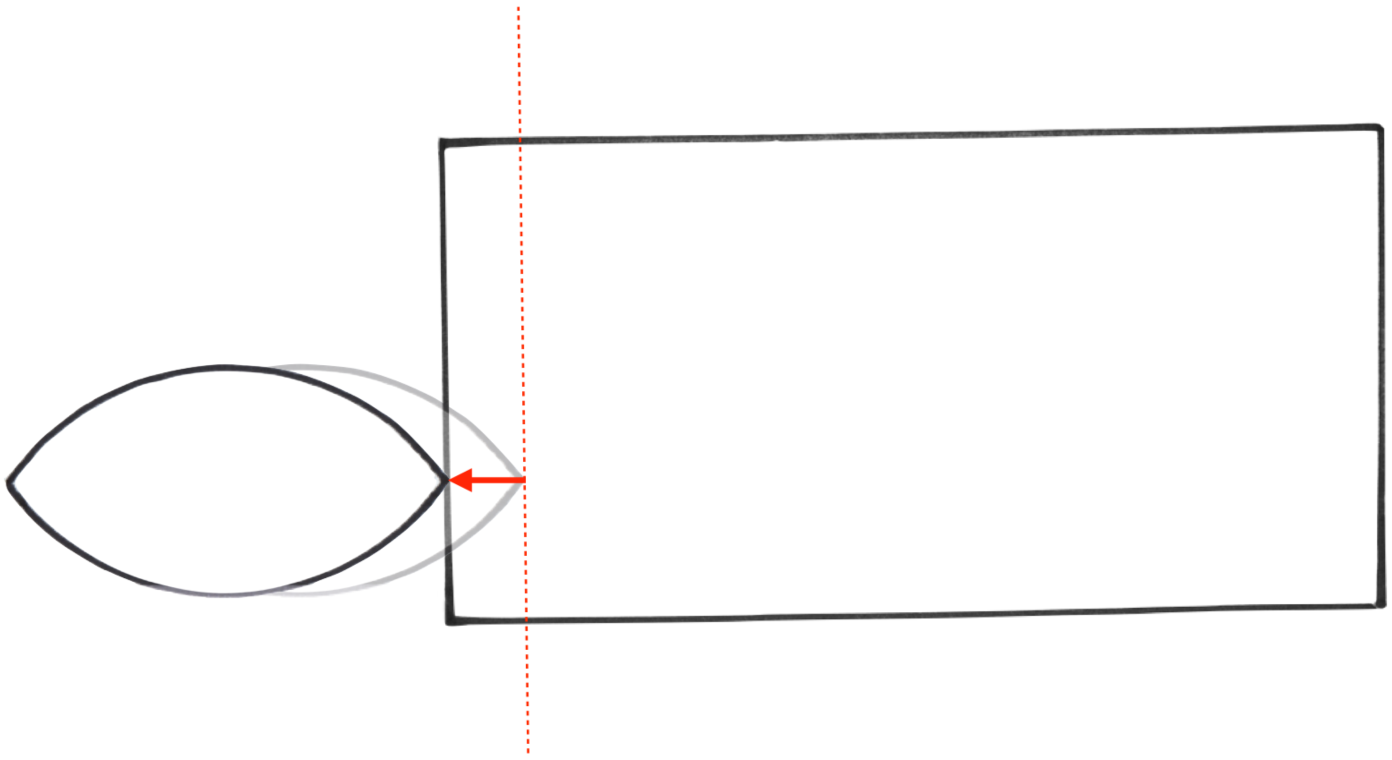
- Test all features in the region and determine if there is any separating axis
- If a separating axis exists then the go stone is not colliding with the board
- Otherwise the stone must be colliding with the board

If the stone is colliding we must now work out what direction to push the stone out. I thought about this for a while and tried to come up with a simple pattern that worked.



Next, I thought that perhaps I could use the previous position of the go stone and try to determine the direction that the stone is approaching from. But then I thought about go stones that were rotating rapidly and how this wouldn't always be correct. Then I started thinking about corner and edge cases, and the longer I thought the more this approach seemed too complicated, like I was trying to invent my own half-assed continuous collision detection method that would probably only work half the time and be almost impossible to test.

In the end I settled on the simplest solution I could come up with: push the go stone out along the axis with the least amount of penetration.



We should probably do the same :)

NEXT ARTICLE: Rotation and Inertia Tensors (http://web.archive.org/web/20181107181450/https://gafferongames.com/post/rotation_and_inertia_tensors/)

(http://web.archive.org/web/20181107181450/https://www.linkedin.com/in/glennfiedler/)

(http://web.archive.org/web/20181107181450/https://twitter.com/gafferongames)

(http://web.archive.org/web/20181107181450/https://github.com/gafferongames)