**Individual Project Proposal: Separating Axis Theorem**

**Background:**

I’ve been working on a personal project of a simple 2D game in python, but I’ve decided to try to implement it without the use of a large game-engine to make it more of a learning experience. Currently the biggest challenge in implementation has been implementing a collision detection algorithm, but I believe it would be better for myself if I learn to implement the algorithm myself rather than use an algorithm that I do not fully-understand. The game I’ve designed so far uses vector style graphics on polygons, so rather than using a simple Cartesian-based collision detection system, I’ve looked to the Separating Axis Theorem for more appropriate performance in the game. This algorithm is quite heavy on the calculation side, so I believe it would be good for requires / ensures clauses.

I’ve already started moving my game over to C#, so if I were to do this as my project topic, I would likely use either C# or C; in the case I use C# I would research precondition / postcondition checking in C#, and if I were to use C I would use Frama-C.

**Explanation:**

* Detects collision between two convex shapes in space
* Shapes will be defined as an array of points with edges between each sequential pair of points, with an additional edge between the initial point and the final point.
* Acts by projecting both shapes onto a single axis
* Collision is then detected by projecting both shapes onto the normal-vectors of each face of both shapes, and in the case that the projections of the shapes overlap across all faces, then the shapes will be said to collide.
* The normal-vectors will be checked clockwise around the shape which is considered *left.*
* Will be implemented using a space with velocity and acceleration for colliding objects; goal is to detect collision and return a value of the vector at which the objects do collide to allow for further implementations.
* With this vector we could return its perpendicular component with reference to the surface to determine a ‘bounce’ factor applied to colliding objects, as well as a parallel component to determine a ‘friction’ factor.
* With this algorithm completed we could apply masses to our shapes and simulate collisions in terms of momentum, or create the ability to reflect a shapes acceleration or velocity in terms of any vector of collision and not just the

**Current Algorithm idea:**

* Use a Simple Cartesian-based collision detection which acquires the minimum square that contains the two polygons and determines whether they are within range of each other (will find some predicate to evaluate whether the two shapes are within range of collision in terms of cartesian space), then afterwards change to the SAT method of collision detection.
* If the polygons are within range the program will use the SAT on the two shapes.
* I will assume all shapes are convex, as if they aren’t the algorithm will not work. If the project needs to be expanded, I could additionally make a model for polygons which will only take the convex-shell of a given shape.
* Additionally, to expand on the project, I would likely try to compare collision detection algorithms; apparently SAT only detects whether collision has occurred and cannot return the face on which collision occurred, or the vector at which the collision took place. Maybe I could even attempt to combine elements of another algorithm to make the SAT perform more to what I need.

**Online References I’ve Used So Far:**

* <https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169>
* <http://www.dyn4j.org/2010/01/sat/>
* <https://www.sevenson.com.au/actionscript/sat/>