# **Overall Theme/Impact**

1. Does the app have an coherent and identifiable theme?
   * From What I had learned about Autonomous Agents, I feel as though this is a good place for an Autonomous Agent in Proximity.
2. Does the app work as intended and it is reasonably engaging (both visually and otherwise)?
   * I think the app is reasonably engaging. Visually you are able to see the agent’s vision by color, along with their sound radius. All these things rotate and move as one entity which the user can see and even interact with when clicking on the screen.
3. Does the app functionality and programming go beyond what we did in class?
   * This project was made from the es6 refactoring homework and was significantly improved upon for independent movement. The class feeds data to sub agents and comes equipped with functions for line interception and collision testing and distinguishing between search targets.
4. Is the app at least approaching/approximating "portfolio quality" that you would not hesitate to show a potential employer?
   * I think so, however I do believe there are improvements that need to be made. I even started an optimized version of the classes.js where I started a cleaner version of the code. I want to add more interactions between different polygonal shapes, but that requires more complicated collisions and SAT testing.

# **User Experience**

1. The purpose of the app and how to use it are obvious.
   * Yes, everything is labelled appropriately along with directions provided at the bottom. Especially the primary controls which are the sliders for vision and sound.
2. Users should be able to figure out how to use the app with minimal instruction. The app runs without errors
   * Users can quickly interact and see how the app works. Especially since the agents are autonomous. The mouse click they can even do, along with instructions. The app runs without errors.
3. Has required text content
   * The required text content is present
4. Has required controls. Widgets are well labeled and follow interface conventions
   * Control widgets are labeled and follow interface conventions.
5. Runs without errors
   * Runs without errors
6. Visual design is pleasing (or at a minimum, "not ugly")
   * Visual design is good. The embed font I might change.

# **Media**

1. CSS does not pass validation
   * Passed
2. HTML does not pass validation
   * Passed
3. Missing required semantic HTML elements
   * Header, Section, Article
4. Majority of CSS is not in an external stylesheet
   * All CSS is external
5. Missing an embedded font
   * Embed Font – BebasNeue
6. Images not properly optimized
   * Images aren’t used
7. Did not draw rectangles, arcs, and lines
   * Agents can be made up of circles or rectangles but at minimum, they have a circle for sound radius and a triangle made up of lines for vision.
8. Did not use canvas API
   * Used canvas

# **Code**

1. File Naming standards NOT followed
   * Files named appropriately.
2. Code standards NOT followed
   * Code standards are followed as needed.
     1. Let, const, no var
     2. Functions are lowercase
3. Inline event handlers used
   * No inline event handlers used
4. Missing/improperly implemented ES6 Modules
   * ES6 used appropriately.

# **What Went Right/Wrong**

# Right

* + Movement, rotation, collision detection by shape and vision line are working.
  + Can safely switch between request and fps loop
  + Can add and remove number of agents
* Wrong
  + Code in certain areas does not follow DRY principle due to bug fixing.
  + Had to account for undefined properties during execution
  + Using ctx.isPointInPath is very limiting depending on the shape you make.

# **Non Course Resources:**

* [Collision Detection By Line](//https:/stackoverflow.com/questions/39670599/canvas-triangle-pentagon-rectangle-collision-detection-with-eachother)

Code:

lineSegmentsIntercept = (function(){ // function as singleton so that closure can be used

var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not need creation

// each time the function is called. This gives a significant performance boost.

v1 = {x : null, y : null}; // line p0, p1 as vector

v2 = {x : null, y : null}; // line p2, p3 as vector

v3 = {x : null, y : null}; // the line from p0 to p2 as vector

function lineSegmentsIntercept (p0, p1, p2, p3) {

v1.x = p1.x - p0.x; // line p0, p1 as vector

v1.y = p1.y - p0.y;

v2.x = p3.x - p2.x; // line p2, p3 as vector

v2.y = p3.y - p2.y;

if((cross = v1.x \* v2.y - v1.y \* v2.x) === 0){ // cross prod 0 if lines parallel

return false; // no intercept

}

v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector

u2 = (v1.x \* v3.y - v1.y \* v3.x) / cross; // get unit distance along line p2 p3

// code point B

if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3

u1 = (v2.x \* v3.y - v2.y \* v3.x) / cross; // get unit distance on line p0, p1;

// code point A

return (u1 >= 0 && u1 <= 1); // return true if on line else false.

// code point A end

}

return false; // no intercept;

// code point B end

}

return lineSegmentsIntercept; // return function with closure for optimisation.

})();

# **Expected Grade:** 85%

I think this project has a lot of potential for creating an autonomous agent based off human senses. I think taking the time to create a well structured class for collision testing would make a huge difference in determining what an agent should do. I already have an optimized class I have decided to work on for better results.