# Group Steering and Visibility

Kevin Bourassa-Houle 26058709

Joshua Romoff 260532864

Comp 521 Project

Professor Verbrugge

**INTRODUCTION:**

The problem at hand is how “Group Steering” behaviours and “Visibility” react when exposed to different game objects and shapes. Games often implement enemies that have a sense of a group, also known as a squad. We experimented with goal-driven strategies for movement that were shared by a group in order to seek and destroy the human player. Visibility was then used as an obstacle for the human player in order to limit the player’s view to any non-occluded area of the map. Once combined with a complex map, designs for both Steering and Visibility run the risk of breaking entirely due to obstacles. Our method of testing their behaviours was to design different maps ranging in geometrical complexity and to both qualitatively assess performance, as well provide some quantitative feedback. At the onset of the project Joshua Romoff was designated to oversee the Visibility Polygon, and Kevin Bourassa-Houle was in charge of constructing the Group Steering. However, as the project progressed roles reversed several times, and code was constantly being shared via Github. We used Google docs for the project report and therefore the work was done as a group.

**BACKGROUND:**

2D polygonal space problems have been at the heart of many computer science and mathematical fields for quite some time. When put into the context of video games and game AI, immediate problems arise that stem from issues regarding smoothly traversing through these complex shapes.

Group steering usually applies an obstacle avoidance force (flee force), and an overall grouping force (seek force). These forces are then combined to dictate the unit’s movement, while keeping an overall sense of a group. When combining these forces on a particular map, certain “sweet spots” where forces cancel out may occur and thus result in rapid vibration. (Reynolds, 1999)

On the other hand, Visibility in a 2D space can be applied using various methods. The first and simplest is a method that raycasts to every vertex of every obstacle and then checks to see where the ray ends by again

checking intersections with every obstacle. This naive approach has a time complexity of $O(n^2)$, where n is the number of vertices.

An alternate approach would be the Angular Sweep method by Asano; which runs in O(n log(n)). Asano's approach sorts the endpoints by angle relative to the origin and constructs a ray that originates and rotates around the origin. Segments that intersect with the ray are sorted by distance and ultimately added as a vertex to the Visibility polygon. (Francisc et. al, 2014)

**METHODOLOGY:**

The three main components of the project were: the squad finite state machine, the individual steering, and the visibility polygon.

**1. Squad FSM:**

The squad's finite state machine had three states; (i) Patrolling, (ii) Attacking, and (iii) Going to the last seen position of the human player.

**(i) Patrolling:**

- This was the initial state that the Squad was put in at the start of the game. When in this state, the squad would have an objective position that was randomly chosen from the map, and would actively move its position towards the objective. As soon as the enemy (human player) would come into a member of the squad's view, the squad itself would transfer into the Attacking state.

**(ii) Attacking:**

- The Attacking state made the squad's centroid move towards the enemy's position. Once in the Attacking state the squad would remain in the state until all members lost sight of the enemy. If at a given time no member had sight of the enemy, then the squad would transfer into the Going to the Last Seen Position state.

**(iii) Going to the Last Seen Position:**

- At this point, the squad's centroid would originally change its objective to be the last known position of the enemy, and move towards it. If at any point the enemy comes into sight of any member of the squad, the squad would transition back to the Attacking state. However, should the Squad reach the last known position without ever seeing the enemy, then it would stay in this position for a set amount of time, waiting for the enemy to appear. Should that not occur within the allotted time, the squad would transfer back to the Patrolling state.

**2. Group Steering:**

The steering itself was conducted by the individuals of the squad. Squad member steering could be broken down into: (i) Sight, (ii) Flee Forces, and (iii) Seek Forces.

(i) **Sight:**

- Squad members would determine if the enemy was in their line of sight, which was defined by both distance and an angular field of view. In addition, should the enemy be within an even tighter range of angles, the member would fire at the enemy.

(ii) **Seek Forces:**

- The seek forces that we used consisted of a pulling group force toward the centroid of the Squad, and a positive force towards the nearest cover point should the squad be in the Attack state. All other forces were only considered within a certain radius of the player. As well, cover points were designated before the game had started to be on opposite sides of the Wall segments contained in the level. Then at runtime, when the Squad was in the Attack state, the closest legitimate cover point that separated the enemy from the squad member was chosen to seek to.

(iii) **Flee Forces:**

- The flee forces that were used were: a separating force between members of the group, a negative rotation force from cover points in order to portray proper cover mechanics (back

against the wall), and a force applied away from walls. All flee forces were normalized and divided by their respective distance from the member, therefore an object that was closer to the member would apply a greater force than that of a further object.

**3. Visibility Polygon:**

- The Visibility is created in the manner derived by Asana. The mesh is a set of computed triangles that emanate from the human player. The walls of the level are made up of segments with two endpoints. Each endpoint is sorted by its relative angle between it and the player. Then, we keep track of which segments are in front of another. Finally, we proceed to create our triangle fan for our visibility polygon mesh by finding the order of intersections between our origin and a given endpoint.

**RESULTS:**

In order to measure the success of both our group steering behaviours and Visibility we considered the following parameters.

**Parameters:**

**(i) Number of walls (W):**

- Increasing the overall number walls relative the map size would result in the forces of the walls on AI players to oftentimes cancel each other out and cause “sweet spots” in the map. AI movement would be stalled due to this, until eventually the force of the squad outweighed the force of the walls.
- A large number of walls *could* affect the performance of the game, but as shown on the screenshot below, hundreds of walls did not introduce any lag at all on my machine.

**(ii) Number of Rooms (R):**

- Rooms with limited entrance or exits resulted in some of our Squad Members getting temporarily or even sometimes permanently stuck.
- Had no effect on Visibility.

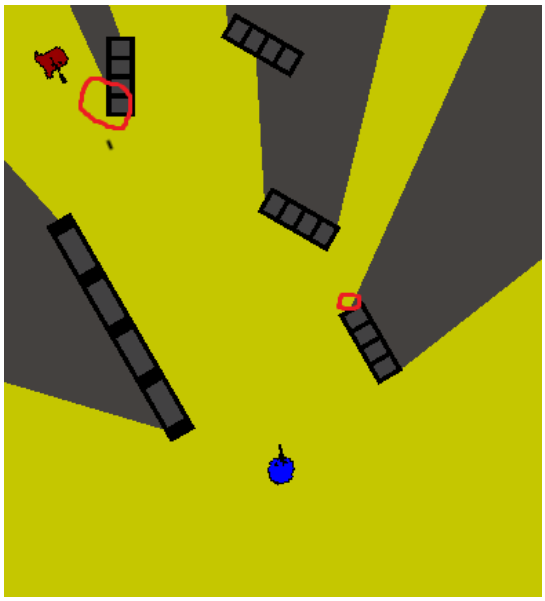
**(iii) Number of Squads (S):**

- The number of Squads would actually scale quite well. The bigger problem was actually if you had too many total squad members than the accumulated Flee force between them.
- Had no effect on Visibility.

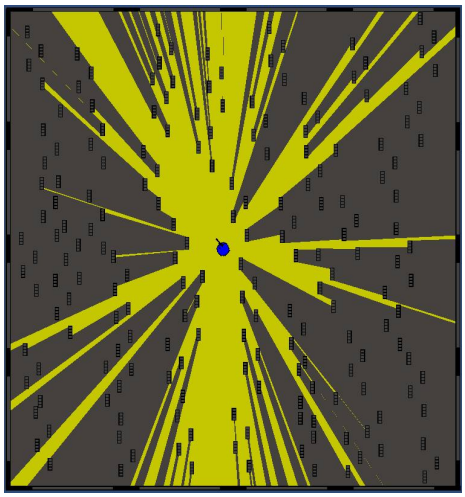
**(iii) Intersecting Walls (I)**

- These would cause glitches in the Visibility polygon due to the ordering of these two segments. As such, to build a map, one must guarantee that no two segments intersect.

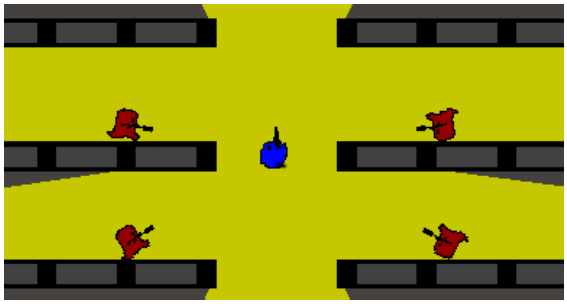
**ScreenShots:**



**(Shows imperfection of Visibility Polygon, due to walls being treated as segments)**

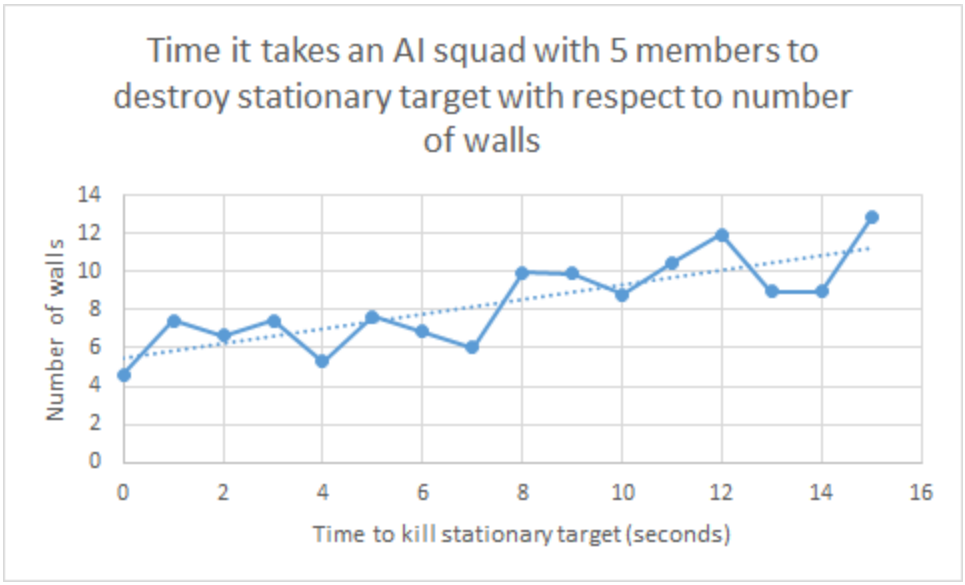


**Shows how many walls the visibility polygon can handle with absolutely no lag.**



**Highlights the AI "wall hugging" due to long parallel obstacles.**

**Chart:**




**Highlights a seemingly linear correlation between the number of uniform walls on a fixed area and the time it takes to kill the enemy. The AI is performing quite decently here, as a "bad" behaviour might have gotten exponentially worse with the number of walls.**

## CONCLUSION/FUTURE WORK:

Depending on the 2D map, one can run into severe challenges with regards to both Group Steering and Visibility. The Visibility challenges stem from there being holes in a 2D polygon, should rectangles or other 2D shapes be used as obstacles. Also, intersecting segments cause inconsistencies to arise, as segments no longer have trivial ordering. The holes can be fixed in future work by adding more segments to match the rectangular shape. Intersecting objects can also be taken care of by detecting an intersection and then subdividing one of the segments into 2 parts, thus creating more endpoints and a more stable polygon.

Group Steering possessed its own issues with regards to forces cancelling each other out, which resulted in “sweet spots”, where the squad member would get stuck for a short while. Applying a small randomized force would in fact alleviate the problem at the expense of the squad members looking “glitchy”.

**REFERENCES:**

Bungiu, Francisc, Michael Hemmer, John Hershberger, Kan Huang, and Alexander Kroller. "Ecient Computation of Visibility Polygons." Deutsche Forschungsgemeinschaft (2014). Arxiv. Web. 1 Apr. 2015.

Patel, Amit. "2d Visibility." Red Blob Games. 1 June 2012. Web. 1 Apr. 2015. .

Reynolds, Craig W. "Steering Behaviors For Autonomous Characters." Red3d. 10 Feb. 1999. Web. 1 Apr. 2015.