# Keepaway - AI Algorithms Report

In the game, we decided to use two of the AI algorithms that were discussed in class, A\* path-finding and Finite State Machines. Both of these were incorporated into each of the non-playable characters, the Eve clones. This report will show how both of these algorithms were incorporated into the game, “keepaway.”

## A\* Path-Finding

We actually used a modified algorithm of A\* path-finding. When we first implemented normal A\*, we noticed that Eve would only run toward the nearest waypoint, and not Ralph, even if he was closer. Furthermore, if there was a clear path to Ralph, Eve would still run toward the nearest waypoint first, even if it meant a farther travel distance to Ralph. Noticing this, Jim came up with the idea of having Eve cast a ray to the next waypoint, while she runs to the current one. Because the room was surrounded by walls, the ray always collided with a wall. If the distance from the ray to the wall was less than the distance from Eve to her next waypoint, this implies that there is a wall in between her current position and her next (following) waypoint. When the distance is greater, this implies that there is a clear path to her next (following) waypoint, and that she should start running to that waypoint instead of stopping at the current one. This allowed us to add waypoints to a map that were generally much sparser than what we would need with traditional A\*.

This had the consequence of making Eve seem like she’s always chasing Ralph. The difference between this being true and our A\* algorithm was that she would not get stuck behind a wall while trying to follow Ralph. To ensure this, we listened for an event where Eve collides with a wall while in the seek state. When this event happens, Eve should call A\* again to re-evaluate her path. Below is a screenshot of the example.



In traditional A\*, she would have proceeded to the nearest waypoint, which was diagonal and to the right of her. Instead, she continues pursuit of Ralph, who has her key.

## Finite State Machine

Because of the complex nature of Eve, we needed a state machine to make sure she did what we wanted, when we wanted. While Panda3d has a built-in state machine object, we decided that it was simpler to write our own using nested if-statements. The code for this is abstracted out, so it will be easy to change it in the future. Our state machine for Eve looks like this:

The transitions are defined inside our code. Basically, once Ralph gets within range of Eve, he will be chased. If Ralph has the key, he will be chased. If Eve is behind the wall while seeking, A\* will be evaluated. If Eve retrieves the key from Ralph, she will seek back to the key nest and return the key, starting to wander again. The screenshot below shows her returning the key to its nest.