<u>CS354R Assignment 4 Project Retrospective Report</u>

Felipe Paz, Seth Parsons, Thomas Moore

For this project, we tried to pick the most straight-forward and convenient system for our AI - a finite state machine. Having this foundational system in mind while designing our AI was really helpful at keeping our design consistent between individual people and generally giving us a path forward. We did still run into a lot of snags, though - not with our design, but just with Godot.

One of us, Seth, had a bunch of issues this time just getting the game to even run. After creating a branch to work on our particular enemy AI, he just couldn't get the game to run and couldn't even get Godot to spit out any error messages as to why. After a couple hours of troubleshooting, we figured out that the lines related to Godot's collision masking/collision layers in our ally AI was breaking the game for him. This was really confusing, because the code worked fine for everyone else (and even people on the same operating system), just not for him. Really confusing bug, but we found a way around it eventually and moved forward (note from Seth - my game was wonky for the rest of the assignment! It kept crashing when I tried to use functions like get_global_transform() or get_translation() too at seemingly random places. Super frustrating!!!).

Another problem we ran into was getting our AIs to react to the player or tokens entering a certain range of "vision". Basically, we gave our ally and enemy a big Area node that corresponded to the range in which they could either see the player or see tokens, and we wanted it to work like collision does - sending an "area_entered" signal to them and connecting a corresponding method to deal with it. We're not completely sure why, but we couldn't get this work (which is why we played around with collision layers and masks!) and instead had to resort to using the get_overlapping_areas() function in different methods to always be scanning for if a player or token was inside that area. We're aware that this is less performant, as a lot of the time we're scanning for nothing, but it's just what we had to do to get it to work.

Finally, the last problem we ran into was random number generation. We were working on a "wandering" mechanic for our enemy AI, which means the enemy would move to random different positions within a defined area when a player wasn't in range. We tried using Godot's RandomNumberGenerator class, which we were told was broken and we couldn't get to work anyway. We then tried to use the classic rand() and srand() from the C++ standard library, which was really finicky to get working as well. Ultimately, we scrapped the wandering mechanic as the RNG generation was hard to understand and our enemy just kept finding a random way to fall off the map (if you want to see what we were doing, we left the code commented out in our enemy.cpp file in the src folder).

Overall, we accomplished what we set out to do. In a more ideal world where we had a better grasp on RNG generation and Godot's collision detection system, we could've implemented more features into our AIs, but we're happy that our enemy and ally work as we expect with all the features we laid out in our milestone documentation.