

CS249b Milestone: Audit in Game Development

Johnny Zhou and Paul Salzman

Our initial project proposal underwent significant changes per our discussion with the TA. We concluded that writing our own game to incorporate the audit code would be too time consuming, detracting from the main goal of examining the applicability of audit code in games. Consequently, we found an open-source network-capable multiplayer spaceship shooting game built on the XNA framework, for which we are writing our own complete auditing facilities.

In the game, each player controls a spaceship with a regenerating shield in a designed 2D arena. There are also random asteroids floating about, which the player must avoid hitting. The objective of the game is to destroy opponent ships and gain points, which can be done by firing the ships main guns or leaving behind a mine. Players can obtain weapon power-ups along the way. Shooting an asteroid will propel the rock accordingly. To destroy an enemy ship, one can shoot asteroids into the opponent ship, or shoot the opponent directly. Shields provide limited protection for each ship, but will regenerate when giving adequate idle time. When a player destroys a ship, a point is awarded and the destroyed ship respawns shortly after. The game has a complete 2D collision model, with rotation, velocity, and acceleration all accounted for.

An unexpected change we came across was that our intent to implement repair code might have been hampered due to the nature of the new game platform in which we are developing. Our initial proposal pushed for a game with more physical simulation that would allow for more errors to arise from networked play. However, our current game has less physical simulation, but there are still opportunities for errors to occur due to lost packets or simple game engine mistakes.

In relation to our initial proposal we have added specific metrics to describe in our final report. These include lines of code for the original game vs. the amount of code added for audit. This can also be compared at a per class or object level, in addition to the global metrics. We can also measure the number of bugs found due to the added audit. Moreover, we will analyze the effects of how and when audit is called (online vs. offline) and how the scope of the audit affects performance.

We have noticed that a lot of functions in our current project receive pointers to objects and only execute the function's code if those pointers are not null. As discussed in class, this does not explicitly describe where the parameter sanity check should be placed. We would also like to explore the consequences of removing these haphazardly placed checks and track how often they are actually necessary.

In terms of progress, we have completed a large portion of the auditing code for roughly half of the classes. This auditing code currently checks for state errors ranging from null pointers, out-of-range values, management instantiation relationships, and general consistency errors. We have also planned out our division of labor and prioritized which classes contain the most important state data, and thus would warrant more in-depth audit.

What we plan to accomplish from now until the end of the quarter is as follows: add networking audit, examine the feasibility and effectiveness of repair code both locally and over the network, add some extra state tracking, and examine the unnecessary sanity checks in functions. Some of the extra state tracking includes keeping a circular buffer of the history for a projectile's path and general network statistics. Finally, of course, we will polish our existing audit code, if necessary.