This game uses 4 threads that are each able to communicate with each other, access information, and synch up their behaviors to prevent race conditions and errors. There is a thread for the Player, two threads for the CharacterEnemies (Thief and Wizard) and one thread for the StepManager. The threads are created as direct Thread objects and stored as references in the GameManager (playerThread, stepManagerThread, and characterEnemyThreads list). When the game ends, all threads are properly synchronized using join() with a 2-second timeout to prevent indefinite blocking. Whenever the game is finished, all threads are safely joined together. The Player, Thief, and Wizard all represent characters who “step” across the game, and whenever the player encounters one of these Character Enemies a battle ensues. As for the StepManager, it manages general step logic but also spawns in smaller Enemies depending on the step. These enemies don’t move across the board, but they do engage a battle when the Player reaches them.

On top of the executor service, the threads are often controlled using a synchronized block via the “stepLock” object. This lock allows the threads to interact safely and to wait on different checks as the game goes on. There are also several volatile flags used to coordinate Game States and counters such as the globalStepCounter are done using the AtomicInteger class.

There are many instances where race conditions may arise, in which there are solutions to these conditions. One problem is that when multiple enemies are defeated

simultaneously the game may not track the gained experience properly- therefore, the synchronized keyword is used to prevent concurrent access, and the experience or other values can be made volatile to ensure changes are visible to all threads. Deadlock errors have also been accounted for. Synchronization locks are kept consistent, and threads are given timeouts so they cannot remain locked for too long. There are also many uses of giving threads delays to ensure that other threads have had a chance to catch up before continuing the game logic.