Assignment 14 Concurrency Report

Multithread Haunted Carnival

Samuel Hay

This project uses three threads that each control a different class (StrongMan, Acrobat, and FortuneTeller). These classes extend runnable so that I can utilize them as threads. These threads are controlled by the GameEngine class which starts the threads then uses and organizes the threads interaction with the shared clown health resource. The threads also share a health pool that can be depleted to reach a game-over.

The major race condition that can come up is within the Clown HP. Each thread attacks the clown's health, and this could result in the clown's health going negative and may not give me an accurate "defeated by" message once the clown is defeated. I was able to control this by using an Atomic Integer for the clown's health which updates the health between each character's attack so that the threads have an accurate HP pool when beginning their attack.

This race condition was further mitigated when I later implemented an Executor Service which forces the threads to interact in an orderly fashion. The result allowed me to give the Clown class time to attack back in between each character's attack so that the fight wasn't one-sided. It also made it so that the log was organized and accurately reflected the actions taking place in order of when they took place.

The executor service allows me to keep my game organized and easy to understand for the player. It will also end the game early if the party is defeated and it will stop the round once the clown is defeated so that the party isn't beating on a dead clown.

Assignment 14 Concurrency Report

Multithread Haunted Carnival

Samuel Hay

The game engine also controls the distribution of loot to each character at the end of each round. It cycles through the loot using a stream and ensures that each character picks up something until all of the loot from the shared TreasureChest resource is depleted. This loot is stored in a list in the GameCharacter class so each thread gets it's own list.

I sacrificed some thread freedom by implementing the Executor Service but the result was greater control over the thread behavior and cleaner output.