Baermelk Discord Bot
Final Presentation

C++ Team 3: Rocco Pearce, Lucas Advent, Roland Allaire, and Maggie Morgan

Responsibilities

Rocco - Filework, bot testing, development, main code execution, profiling, unit testing

Lucas - Formatting, bot testing, development, code editing, profiling, unit testing

Roland - Encounters, bot testing, development, github maintainer, profiling, unit testing

Maggie- Prototyping, bot testing, development, coding encounters, profiling, unit testing

Technology Stack/Tool Chain

Language Used: C++

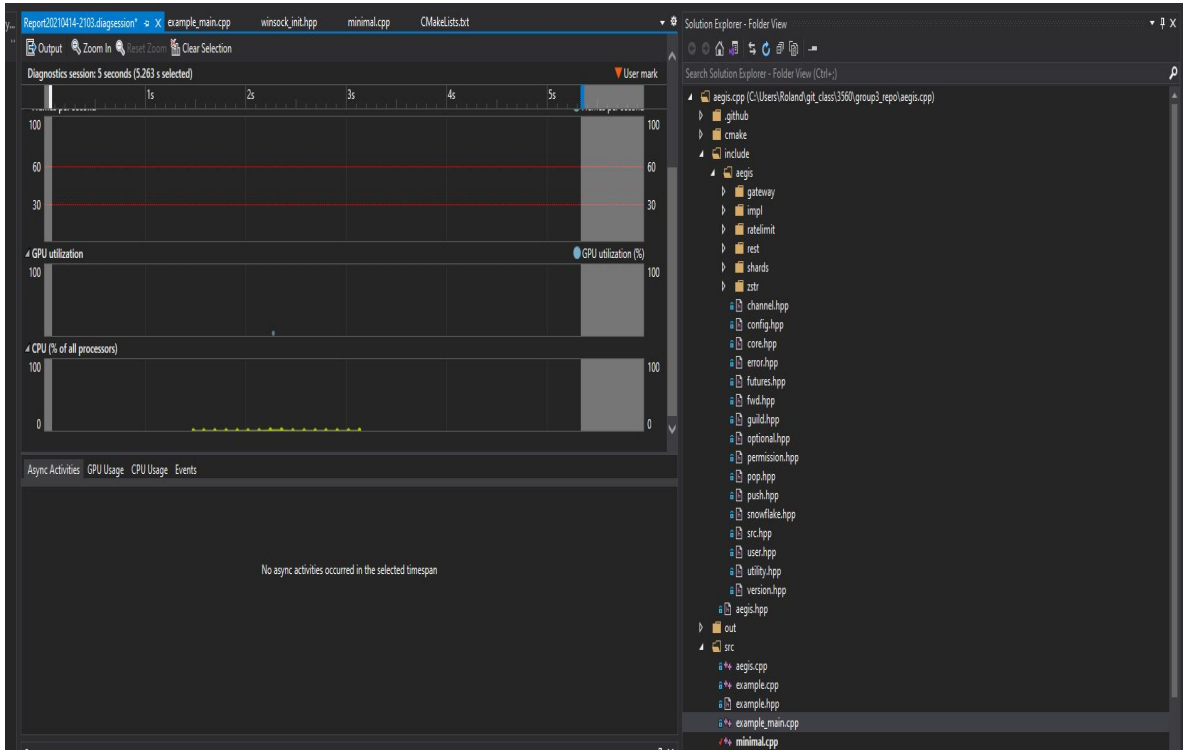
- Visual Studio:
  - Used for coding the bot through the Aegis.cpp library
- Github:
  - Project work is shared on the repository so that everyone can access one another’s contributions
- Discord Developer Portal
  - Bot information is stored here
- Proto.io
  - Used for developing a GUI prototype of the app
- Valgrind
  - Used to test and check base code
- Catch2
  - Performed unit testing for the bot

Reflections

Because our project was very dependent on a repo (aegis.cpp), it was very difficult to run unit tests, prototype, and profile

- Valgrind: because we all have Windows computers, using Valgrind was exceptionally difficult and ended up being mostly useless, we had to explore other options to attempt to check for memory leaks.
- Catch2: the way our program is set up made it especially difficult to break apart and unit test using Catch2, it took many different attempts as well as meetings with the TA to attempt to figure out what to do.

Profiling



For profiling, it was very difficult to debug and run with our project, so we had to prototype the executable within the prototyping frame instead. Because the executable is very minimal compared to a large portion of our project and runs through quickly, there is not much GUI or CPU usage at all. The CPU shows up as almost a dot, but at a closer look is actually a small increasing graph of data. GUI stayed the same throughout.

Unit Testing

Have been having issues with this and have met with Krerkkiat to discuss where to go from here

We are able to split the main functions into 4 and all individually focus on one part while unit testing.

Rocco: main() - holds the main game play and requires inputs to run through different aspects

Lucas: replace() - replaces the users file based on their outcome of the game and keeps track of inventory

Roland: deleteSpace() - deletes unwanted spaces within the users file

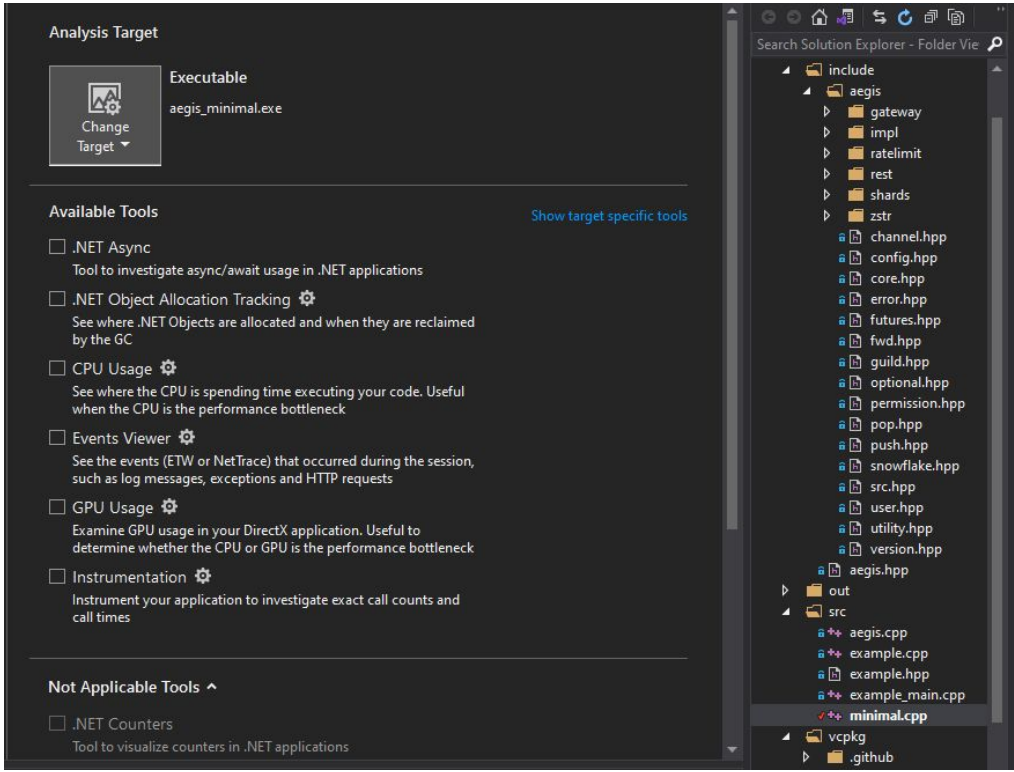
Maggie: pdeath() - determines when and what happens if the user dies in the game

When testing the different functions manually through discord, we are able to get the expected outputs.

Memory Leak

Memory leak issues:

- Visual studio project is within a git repository
- It does not contain a solution and memory leaks cannot be found using performance profiler
- Valgrind cannot be used because all of the individual files that allow the project to run cannot be compiled using GNU/Linux



Git Repo

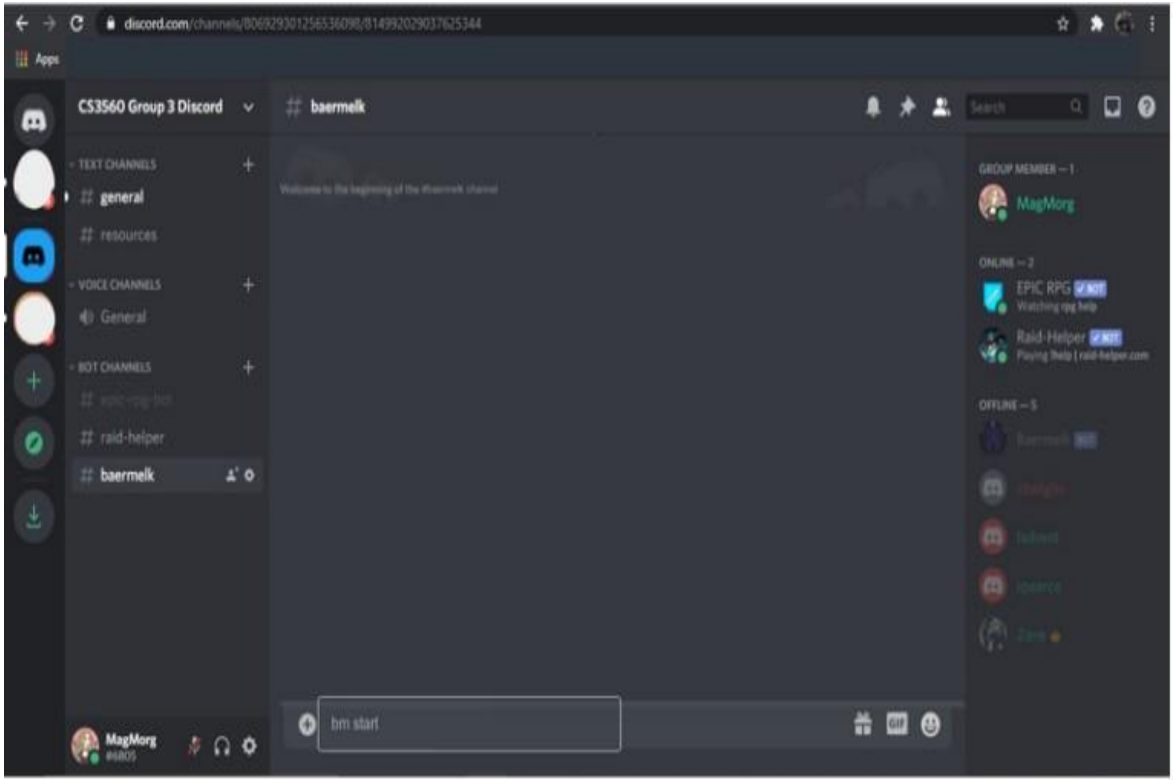
Link: https://github.com/roland199/CS3560-C--group-3

- Branches:
  - Main
    - Contains source code, milestone powerpoint, and README.md
    - Individual contributions are merged into the main_minimal.cpp file
  - Aegis
    - Aegis library
  - Encounters
    - Minimal file with added encounters
  - Filework
    - Minimal file with filework saving user information to a file
  - Format
    - Minimal file with formatting for command outputs
  - Prototype
    - GUI prototype for Baermelk
  - Final
    - All testing and profiling can be accessed

Prototype

Link: https://pr.to/WJW73B/

Tool Used: proto.io



Discord Bot Demo

Discord Bot Final Version

- Added emojis to outputs
- Final boss, baermelk is added
- Weapons & armor impact the outcome of encounters/final boss
- Bandaids and health kits can be used to heal
- Progression system was added using experience
- Arrows are used up in situations that call for a bow