

Project Overview

Software Information

The Game Client executable and Game Updater executable require Microsoft.Net version 2.0. When you run the game installer, it will let you know if you have the correct version installed to run the software. The game also requires the [libmap.dll] binary that Stephan wrote, however this is automatically installed as part of the game. Aside from .Net 2.0 and our custom binaries, there are no software requirements to run the game. The minimum RAM requirement is 256MB, and it is recommended you have over 512MB of ram. The game will run on less, however the game tends to “lag” during play else wise.

The software used to create the Game Client and Game Updater was Visual Basic Express 2005. This is freely available from the Microsoft website, and is required for you to view the source code. All of the programming was completed within VB.Net 2005, and requires no other external editors. During development there were no add-ons used in the development environment. Visual Studio 2005 is an easy tool to use and allows both beginners and experts to feel comfortable in their programming environments. VB.Net is built off of C#, so should one feel inclined to decompile the executable, the source will be in C#, and is automatically optimized by the compiler.

Installation Instructions

If you have run [SetupClient.exe], you have already installed the game, the source code and all of the documentation required for the game (including the MSDN style documentation for the source, the Network Protocol documentation and any other relevant pieces of information). As well as having the game already installed, the installer checked for dependencies (such as the DLL and .Net 2.0) and alerted you accordingly. To uninstall the game, source and documentation, you can do so from the Add/Remove Programs list in Control Panel.

From the Programs folder in the Start menu, there are two subdirectories: Documentation and Source Code, along with Run Game. Upon starting the game, you may have to specify the server’s target IP (If you are running both the client and the server locally, you would enter the IP 127.0.0.1). You must do this to overcome the disadvantage of a free roaming server, as it can be loaded anywhere, and is not bound to a single address. The in class demonstration will have several users connected to a single central server.

Obstacles

The biggest obstacles that I overcame during the development of the server were threading issues (cross threading, memory leaks, and synchronization). There were other issues, such as code optimization and packet merging (more on that later), but overall I think we accomplished about what I expected we would. We have a functional, graphically pleasing client that is both functional and reasonably efficient (given the knowledge I had about graphics programming).

The biggest issue I ran into during development was threading issues. These ranged from learning how to pass data from a worker thread to a main thread or graphical thread. The graphics thread in VB.Net does not work well with internal threads such as the socketsClient and dataHandler class threads. To overcome the limitations present in the language, the WithEvents handler is used and events are raised on the corresponding thread. Then a local sub or function with the same signature as the event raised on the worker thread handles the event. An example of this would be Public Function socketsClient_onConnect() Handles socketsClient.OnConnect. The event on the socketsClient class thread is raised, and the function with the same signature is called on the handling, or main thread.

The next largest issue I ran into during development was packet merging. In brief, TCP/IP is a streaming protocol, so the data received comes in streams rather than distinct packets of data. As a result, multiple "packets" (packet referring to a complete set of data to carry out an operation in the program) are stuck together. We did not encounter problems like this last year, as VB6 had automatic packet handling. VB.Net sockets however leave this up to the programmer. While this adds flexibility and customizability to the code, it makes for another programming headache to overcome. We decided to separate data based on the expected length for a packet of data, and send the remaining data found back to the data parser. This was accomplished by writing the Check_Merge() function. The function allowed for the quick and easy separation of data within the program.

The last large issue in the development of the game was related to the rendering of graphics. Originally I intended to load all the information on the fly from the binary map file; however this was rather slow and made the graphics extremely choppy. Second was to load the bitmap file of the map into memory and render it on a PictureBox control; however issues arose with CPU usage and transparency. The final method decided upon was to store the map in memory and directly draw it to the surface using the form's onPaint method. This turned out to be rather efficient and used the least amount of RAM compared to all other options.

There were a couple minor issues such as thread synchronization and memory leaks which were easy and quick to solve, however the issues still did take up time during development, and along with function optimization and code checking, I lost approximately two to three weeks of in class time to overcome issues and optimize the code.

Unimplemented Features

We were unable to implement many of the features that we originally planned to such as the party and trade systems and the items and loot systems. These were very ambitious plans to accomplish, but when we originally planned the game development guidelines, I neglected to consider the learning curve for new programming languages, research time and debugging and optimization requirements. Overall, we successfully built a simple and flexible game, with an extendable and customizable game engine. If I had another month, perhaps we would have items and loot enabled with a DirectDraw capable front end, which would make the game more efficient and enriched. Overall, I believe that we achieved more than what was expected given the time constraints and uses we ran into during development of the program, the protocol and file formats.