Limerick Institute of Technology

Final Year Project Report

Remote Artificial Intelligence for Games

David Morton
K00179391

supervised by
Firstname Surname

Wordcount:

I declare that no element of this assignment has been plagiarised:
Student Signature

September 29, 2015

Abstract

This is the abstract.

Contents

1 Introduction **3**

1.1 Technologies 3

2 Problem Definition and Background **3**

2.1 Background 3

2.2 Objectives 3

2.3 Project Planning 4

3 Literature Review and Research **4**

3.1 API Design 4

3.2 UDP vs TCP 4

3.2.1 Pimpl Idiom 4

3.3 Server System Design 4

3.3.1 Fork or Select 5

3.3.2 Round Robin DNS 7

3.4 Existing Technologies 7

3.5 Design Patterns 7

3.6 Artificial Intelligence & Algorithms 7

3.7 A* Path finding 7

3.8 Brute force 7

3.9 Graph Sturctures 7

3.10 Documentation 7

3.11 Testing 8

3.12 Conclusions 8

4 System Design and Configuration **8**

4.1 System Architecture 8

4.2 Class Diagram 8

4.3 Sequence Diagram 8

5 Example of use **8**

5.1 User Stories 8

6 Testing & Implementation **8**

6.1 Implementation 8

6.2 Testing 9

6.3 Future Features 9

7 User manual (where appropriate) 10

8 Critical analysis and Conclusions 10

References 11

Appendices 12

A Graph Appendix 12

B Another Appendix 12

List of Tables

1 Example table caption 3

List of Figures

1 EXAMPLE Figure and caption 3

Code Listings

1 Server listen socket implementation [1] 5
2 EXAMPLE IRAIG.h interface used for game engine 8

1 Introduction

This section should contain general information on the project and the technology used.

1.1 Technologies

Technologies used in the project.

2 Problem Definition and Background

Table 1: Example table caption

A	B
C	D

This is a reference to table 1

2.1 Background

Figure 1: EXAMPLE Figure and caption

And this is a reference to the figure in Figure 1.

2.2 Objectives

Explain how you understand the project objective(s), what is to be solved or done, your perception of what has to be accomplished including the scope.

2.3 Project Planning

Agile approach, PID, Charter, Phases.

Provide a plan for the project timeline implementation including key tasks, deadlines and outputs.

3 Literature Review and Research

Outline the literature reviewed and research completed to inform the completion of your project. This should include commentary on current and emerging technologies/systems, key data, information, source, code etc. which was sourced as part of your research. Where relevant indicate any conclusions or recommendations arising from your research

3.1 API Design

The API should be designed with developers in mind and must be intuitive to use. If a developer wants a path from A to B on their map, the API must allow the developer to quiery the service and recieve the answwer. While implementing the API the developer needs options or weigths that may determine what path is choosen for a particular situation.

3.2 UDP vs TCP

3.2.1 Pimpl Idiom

The Pimpl idiom is a design pattern used to hide the implemenation details of an API. A pointer to an object contained inside the .cpp implementation file removes any of the declarations and includes needed for the API header file. The implementation code is then stored in a library and the client can then link against the library when building their system. This technique can be used with proprietary code and prevents users from casually reading he header file to find out how the system is coded.

This technique is used in the client side API for RAIG to simplify the interface for the user. APIs should be a simple as possible to support ease of use.

3.3 Server System Design

There are many ways a server systems architecture can be designed. Multiple users may need to connect and use the services provided by the server at any one time. How the server will scale to the number of users is a very important consideration when designing

a remote service. This decision needs to made early in the initial envisioning phase of a project becuase it determines how the resources of the host machine will be used, how to limit or allocate more resources if needed, and also the approach taken to implement the solution.

3.3.1 Benefits

Moving the graph search and processing to the server is a huge benefit to game developers. It allows the developers to forget about how the AI algorithms will run on particular systems or how much time it may take to calculate a specific query. Graph processing for state based AI is an example of a CPU intensive task, when implemented well. Removing this processing to the server frees up the resources needed for the action so they can be used elsewhere.

3.3.2 Efficiencies

Effecient processing and network communication is an integral part of creating a service that can be used by developers effectively.

3.3.3 Fork or Select

Two ways to handle multiple clients connecting to a server are to *fork()* a new process for each client, or using the *select()* function to listen for active file descriptors. Both methods have there advantages and limitations. Forking a process to handle a client request or using select to handle requests starts with creating a listening socket on the server.

Code Listing 1: Server listen socket implementation [1]

```
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
        perror("creating stream socket");
        exit(1);
}

server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(HANGMAN_TCP_PORT);

```

```
11 if (bind(sock, (struct sockaddr *) &server, sizeof(server))
      < 0) {
12         perror("binding socket");
13         exit(2);
14 }
15
16 listen(sock, 5);
```

The socket is created using the *socket()* function, this returns a file descriptor which is an integer value identifying the socket on the system. The *SOCK_STREAM* argument is used to create a TCP streaming socket. Next the servers address structure is filled in with details pertaining to a passive server. The *htonl(INADDR_ANY)* function on line 8, tells the server to listen for any address, IP 0.0.0.0, in order to receive incoming connection requests from any IP address. The address structure is also initialized with a port number, shown on line 9 in code listing 1. Using the *bind()* function while passing in the socket file descriptor and address structure will bind the socket to the port.

This is similar to how a client socket is created and initialized. The main differences being that the active client will use the *connect()* function to connect to a passive server and will also fill in an address structure with details of the server. The server on the other hand uses the *listen()* function to passively listen for incoming client connections on the port number it is bound to.

Once the server is running and listening for incoming clients it can handle each client request by forking a process. Forking a process starts with a parent, this parent will create child processes that will execute along side the parent and will communicate with clients using the file descriptors associated with each. When a server accepts a connection is will *fork()*, this will create another process that contains a copy of the data stored in the parent process, along with the newly created file descriptor for the conenction. The parent will then close the file descriptor it has which reduces the reference count to the descriptor by 1. The child process has a copy of this information and so the child can still use the file descriptor to send and receive data. Once the child process is finished with the client communications it closes the socket, reducing the reference count to 0, and exits. The parent must then kill the process in order to aviod creating *zombie* processes that have been allocated system resources but are not being used. Killing a child process involves handling signals from the kernel in the parent that identify the child process being terminated. An advantage to using *fork()* over *select()* is that the process can handle each client individually and does not have to worry about any more incoming

connections to the server. A disadvantage to forking is that it has to uses shared memory to communicate with other processes. This can introduce concurrency issues that can be very complex, difficult to understand, and hard to debug if something goes wrong.

Another approach to handling client connections is to use *select()*. This method will not create new processes. A single process handles all open file descriptors by looping through them and processing any that are active. The advantages of this approach over forking is that all clients are handled by a single process. This removes the need for shared memory or synchronisation primitives in order to communicate with other processes.[2] The disadvantage to using *select()* over forking is that it cannot act like there is only one client at a time. With *fork()* the developer can program the server as if there is only ever going to be one client which makes the code alot less complex.

In conclusion, handling multiple clients at the server can be achieved through a variety of different ways

3.3.4 Round Robin DNS

Multiple servers running instances of the AI application that can handle a number of clients each.

3.4 Existing Technologies

3.5 Design Patterns

Observer pattern
Stradegy pattern
Lazy loading
RAII

3.6 Artificial Intelligence & Algorithms

3.7 A* Path finding

3.8 Brute force

3.9 Graph Sturctures

Game AI is best represented as a graph like structure. Using a graph allows many different algorithms to be used to process the data efficiently.

3.10 Documentation

LaTeX
doxygen
Make
MetaUML
UMLet

3.11 Testing

Google C++ testing framework

3.12 Conclusions

4 System Design and Configuration

Describe your approach to solving the problem and, where relevant, include sub sections on

4.1 System Architecture

4.2 Class Diagram

4.3 Sequence Diagram

Actual code is to be provided on an accompanying and suitably labeled CD.

5 Example of use

How do you see the solution being used in practise? Illustrate as appropriate with User Stories / Use Cases

5.1 User Stories

6 Testing & Implementation

6.1 Implementation

Describe how the solution was implemented and any problems particular to this stage.

Code Listing 2: EXAMPLE IRAIG.h interface used for game engine

```
const float TimeBetweenCameraChanges = 2.0f;
const float SmoothBlendTime = 0.75f;
TimeToNextCameraChange -= DeltaTime;
if (TimeToNextCameraChange <= 0.0f)
{
    TimeToNextCameraChange += TimeBetweenCameraChanges;

    // Find the actor that handles control for the local
       player.
    APlayerController* OurPlayerController =
       UGameplayStatics::GetPlayerController(this, 0);
    if (OurPlayerController)
    {
        if ((OurPlayerController->GetViewTarget() !=
           CameraOne) && (CameraOne != nullptr))
        {
            // Cut instantly to camera one.
            OurPlayerController->SetViewTarget(CameraOne);
        }
        else if ((OurPlayerController->GetViewTarget() !=
           CameraTwo) && (CameraTwo != nullptr))
        {
            // Blend smoothly to camera two.
            OurPlayerController->SetViewTargetWithBlend
               (CameraTwo, SmoothBlendTime);
        }
    }
}
```

And this is a reference to the code listing 2

6.2 Testing

Describe the testing that was carried out and the results.

6.3 Future Features

Give an account of the results of the project, what was accomplished, what wasn't and the reasons why. If the project is incomplete or has potential for improvement state what further features could be included and how existing ones might be improved.

7 User manual (where appropriate)

Write a concise but complete user manual or guide. This can be light on text so long as a novice user has enough information to use the application unaided.

8 Critical analysis and Conclusions

Evaluate the success of what was achieved against the original objectives.

What difficulties were encountered and what lessons learned from the project?

How well did the project follow the planned timeline and how useful was the timeline?

What are your key conclusions from the work.?

This is a reference to the graph appendix A

This is a reference to the Network Programming Book [1, p. 215]

References

[1] W. Richard Stevens, 2003. *Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition)*. 3 Edition. Addison-Wesley Professional.

[2] The World of select(). 2015. The World of select(). [ONLINE]
Available at: `http://www.lowtek.com/sockets/select.html`. [Accessed 01 October 2015].

Appendices

A Graph Appendix

This is the graph appendix...

B Another Appendix