Limerick Institute of Technology

Final Year Project Report

Remote Artificial Intelligence for Games

David Morton
K00179391

supervised by
Eugene Kenny

Wordcount:

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

Typeset in LaTeX2e

October 10, 2015

Abstract

This is the abstract.

Contents

List of Tables

List of Figures

1 Introduction

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

Aritificial Intelligence in games is becoming more and more popular due to new advancements in technology and increasing hardware resources on user devices. Running complex AI on users machines can sometimes be infeasible as the necessary CPU time or speed is just not available. Games are very CPU intensive applications that utilize a systems hardware for graphics rendering, audio output, networking, user inputs, and feedback. Artificial Intelligence can sometimes demand alot of processing in order to accurately generate realistic behaviours that can mimic sentient entities in a game. The type of algorithm needed for a particular behaviour in a game comes down to the knowledge and skill of the developer. This however means that highly skilled developers may attempt to implement a very complex and efficient ai algorithm when it may not be needed at all, and sometimes can produce inferior results compared to a simple animation or pre-defined behaviour. Choosing whether to use an algorithm to generate aritificial intelligence or to simply pre-define a behaviour can be very difficult and could lead to huge amounts of time and resources being wasted in a feable attempt to create SkyNet[1].

Network bandwidth is also becoming cheaper to deliver to users through home broadband connections and on their mobile devices. The combination or cheap bandwidth, increased network speeds, and high performance computer hardware opens up the possibility of distributed processing for faster results from computing large data sets. Distributed processing is used to divide large datasets into smaller, more managable pieces in order to process them individually and produce results faster through the use of multiple machines connected over a network. It is the use of more than one processor to perform the processing for an individual task[5]. Multiplayer games can utilize a network to run a game instance on high powered machines that will then send rendering information to

the connected clients. MMORPG Massively Multiplayer Online Role Playing Games use the client/server architecture to manage thousands of users simultaneously while they are connected to remote servers running instances of the game. World of Warcraft[6] is an example fo a very popular MMORPG that efficiently utilizes available network bandwidth and the internet to deliver fast, reliable, and relatively bug free gameplay to thousands of players all over the world. Because aritificial intelligence algorithm processing can be intensive, using a similar model to the client/server architecture used in multiplayer games to process large amounts of data on a higher end machine instead of the users device can free up the devices CPU processing time and allow it to allocate more to other processes.

What this project aims to achieve is the use of the internet as a means to transmit the results of executing CPU intensive path finding algorithms used in real time strategy games on a remote server machine to the client. Abstracting the implementations of path finding algorithms and providing developers with a simple to use Application Programming Interface that communicates with a remote server to generate results may save individually developers and studios a considerable amount of time, money, and resources. The A* pathfinding algorithm is a popular algorithm used in games to calculate paths through a games geometry. The graph consists of nodes and connections, also known as vertex and edges, with weights or values associated with each edge. Moving from one node to another node may incur a cost which is the value assigned to a connection. A* works on a non-negative weighted graph[4, p. 198] and so the game geometry must be converted into a state that the algorithm can understand. This algorithm, once it recieves the data it needs over the network, will process the data sending the result back to the client for use in the game. Many times in real time strategy games the player or enemy units will need to move from one location on the map to another along the shortest possible route while avoiding obstacles such as villages, cliffs, towers, or oceans. It would not be acceptable for the units to just cut across the map ignoring its explicit structure. The A* pathfinding algorithm has many different variations depending on specific use cases and available hardware for efficiencies.

The implementation details for the client side API will be explained in detail in the section 6.2. The API itself needs to be usable by game developers in an intuitive way in order to get the results that they need from the system and so will need to be designed with a focus on developers. The communication layer that will be used to transmit data between the client and the server will need to be efficient and reliable in order to deliver data that is complete and integral. Missing packets or corrupt data is unacceptable when accurate processing needs to be done by complex algorithms to produce results fast. Further, congestion on a network needs to be handled efficiently to prevent delays in communication bewteen the client and server when requesting path data for in game entites. Large delays may annoy users of the system and can cause a bad gaming experience. This will be described in the section 6.3. The server system that will communicate with each client to handle requests and schedule tasks for the Aritificial Intelligence system will be discussed in section 6.4. The server system will need to communicate with clients concurrently, handling data I/O and managing a task record in order to efficiently service each client fairly. Efficiently allocating resources as needed and recovering from interrupted or terminated client communications will be the job of the server. Identifying each client and routing information is also necessary to ensure a good user experience. The AI module residing on the server that will communicate with the server system will be responsible for all algorithm management. This will be explained in section 6.5. Choosing the right implementation of an algorithm in order to achieve the best possible speed and efficiency will be key to producing fast and accurate results for the client games using the system. There are many differnet types of games that fall under the umbella of Real Time Strategy. The AI module will attempt to service client requests that vary greathly from one another in the RTS genre of games in an effort to provide a generic solution to pathfinding that can be applied to many different situations.

2 Problem Definition, Background, and Planning

The problems faced by game developers when implementing Artificial Intelligence in their games come under two categories. The first is the choice of whether to use a complex implementation of an algorithm to produce expected behavioural output data or to just use some nice animation effects to emulate smart entity characteristics. The other problem arises when the choice is made to use an algorithm to produce behavioural data, what is the best and most efficiently way to implement the algorithms execution. When it comes down to using an algorithm instead of some animation effects more problems arise such as finding a developer that is skillful enough to produce the results efficiently and also has the knowledge available to recognized whether a simple implementation will produce better results than a more complex and conveluded one.

2.1 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.

2.2 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 Overview of Aritificial Intelligence in Games

3.1.1 Hardware Resources

3.1.2 Levels of Aritificial Intelligence

3.2 Networked Aritificial Intelligence

3.2.1 Distributed Client Side Processing

3.2.2 Central Server Architecture

3.3 Current Issues with Aritificial Intelligence in Games

3.3.1 Complex Algorithms & Implementations

3.3.2 Network Latency

3.3.3 Human Perception

3.4 Conclusions of Research

Explain the reasoning behind the decisions made for the client api, the sockets layer using TCP, the server application forking, and the AI system structure using C++ and Python.

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

```
1  const float TimeBetweenCameraChanges = 2.0f;
2  const float SmoothBlendTime = 0.75f;
3  TimeToNextCameraChange -= DeltaTime;
4  if (TimeToNextCameraChange <= 0.0f)
5  {
6      TimeToNextCameraChange += TimeBetweenCameraChanges;
7
8      // Find the actor that handles control for the local
       player.
9      APlayerController* OurPlayerController =
       UGameplayStatics::GetPlayerController(this, 0);
10     if (OurPlayerController)
11     {
12         if ((OurPlayerController->GetViewTarget() !=
           CameraOne) && (CameraOne != nullptr))
13         {
14             // Cut instantly to camera one.
15             OurPlayerController->SetViewTarget(CameraOne);
16         }
17         else if ((OurPlayerController->GetViewTarget() !=
           CameraTwo) && (CameraTwo != nullptr))
18         {
19             // Blend smoothly to camera two.
20             OurPlayerController->SetViewTargetWithBlend
               (CameraTwo, SmoothBlendTime);
21         }
22     }
23 }
```

Figure 5.1: This is a code listing

And this is a reference to the code listing 5.1

A	B
C	D

Table 5.1: Example table caption

This is a reference to table 5.1

Figure 5.2: Picture of the LIT Logo

And this is a reference to the figure in Figure 5.2.

6 Remote Artificial Intelligence for Games

6.1 Introduction

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

Client side API

Sockets layer

Server side application

AI Algorithms

6.1.1 Networking Model

Client server model

6.1.2 Design Patterns

Observer pattern

Stradegy pattern

Lazy loading

RAII

6.2 Client 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.

6.2.1 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.

6.2.2 Game Representation

Custom structs and contraining how the game will be presented to the AI

6.3 BSD Sockets Layer

6.3.1 UDP vs TCP

6.3.2 Communication Protocol

Packet header overview and details.

6.4 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.

6.4.1 Benefits of Remote Processing

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.

6.4.2 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.

```
1  sock = socket(AF_INET, SOCK_STREAM, 0);
2  if (sock < 0) {
3          perror("creating stream socket");
4          exit(1);
5  }
6
7  server.sin_family = AF_INET;
8  server.sin_addr.s_addr = htonl(INADDR_ANY);
9  server.sin_port = htons(HANGMAN_TCP_PORT);
10
11 if (bind(sock, (struct sockaddr *) &server, sizeof(server))
       < 0) {
12         perror("binding socket");
13         exit(2);
14 }
15
16 listen(sock, 5);
```

Figure 6.1: This is another code listing [2]

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 **??**. 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.[3] 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

6.4.3 Round Robin DNS

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

6.5 Artificial Intelligence & Algorithms

6.5.1 A* Path finding

6.5.2 Brute force

6.5.3 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.

6.6 Analysis, Testing, and Results

Describe the testing that was carried out and the results.

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.

Google C++ testing framework

6.7 Summary

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.

LaTeX

doxygen

Make

MetaUML

UMLet

8 Critical analysis, Conclusions, and Future Work

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 [2, p. 215]

References

[1] Skynet (Terminator) - Wikipedia, the free encyclopedia. 2015. [ONLINE] Available at: *https://en.wikipedia.org/wiki/Skynet_(Terminator).* [Accessed 09 October 2015].

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

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

[4] Ian Millington, 2009. *Artificial Intelligence for Games.* 2 Edition. CRC Press.

[5] Distributed Processing . 2015. Distributed Processing . [ONLINE] Available at: *http://docs.oracle.com/cd/A87860_01/doc/server.817/a76965/c29dstpr.htm.* [Accessed 10 October 2015].

[6] World of Warcraft - Wikipedia, the free encyclopedia. 2015. World of Warcraft - Wikipedia, the free encyclopedia. [ONLINE] Available at: *https://en.wikipedia.org/wiki/World_of_Warcraft.* [Accessed 10 October 2015].

Appendices

A Graph Appendix

This is the graph appendix...

B Another Appendix