**Network Discovery**

Durch Network Discovery ermöglichen wir den Benutzern unseres Spiels, andere Geräte in einem Netzwerk zu finden, um mit ihnen in Interaktion zu treten und zu spielen. Unser Network Discovery Tool ist primär für Computer gedacht, da unser Programm nur auf solchen ausführbar ist. Dennoch würde es auch für alle Geräte in einem Netzwerk funktionieren. Ein wichtiger Baustein unseres Network Discovery Tools ist die „iphlpapi.dll“, die wir in unsere Network Discovery Klasse eingebunden haben (Abbildung 1.0).



Abbildung 1.0

Die von Microsoft entwickelte iphlpapi.dll (Abbildung 1.1) ist ein Systemprozess, welcher meistens unter „C:\WINDOWS\system32\“ vorzufinden ist. Dieser wird von unserem Programm verwendet, um ARP-Requests hinauszuschicken, um somit herauszufinden, welche Computer sich im Netzwerk befinden. Dieser Systemprozess belastet die CPU immens stark, daher muss man bei der Implementierung sehr genau aufpassen.

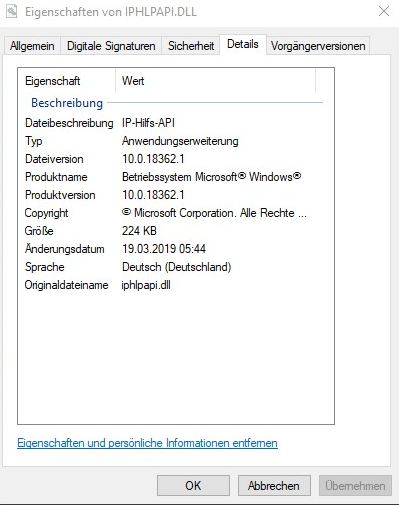


Abbildung 1.1

**Ablauf des Netzwerkscans**

Wenn man die Funktionen unseres integrierten Tools verwenden will, öffnet man die Einstellungen. In der **Registerkarte** unter dem Punkt „Network Discovery“, findet man das GUI vor, welches primär für den Netzwerkscan gedacht ist. In diesem Fenster lassen sich die Firewall Einstellungen adjustieren, sowie die eigentlichen Funktionen „QuickSearch“ und „DeepSearch“, starten. Die Informationen, wie Name, IP-Adresse und Subnetzmaske des jeweiligen Adapters sind in einer „Listbox“ während des Ladens der Einstellungsanwendung schon vorgeladen. Irrelevante Netzwerkadapter, wie der von „VirtualBox“, „Bluetooth“ und andere Netzwerkadapter, mit denen zum jetzigen Zeitpunkt sowieso keine Netzwerkverbindung möglich wären, werden automatisch ausgeblendet. Die angezeigten Netzwerkadapter lassen sich per Rechtsklick auswählen (Abbildung 1.6) und danach gäbe es die Möglichkeit mit dem ausgewählten Adapter einen „QuickSearch“ bzw. „DeepSearch“ durchzuführen.

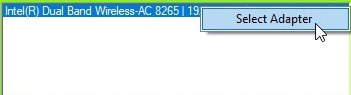


Abbildung 1.6

Der Begriff „QuickSearch“, ist wie der Name schon verrät, für ein schnelles Suchen gedacht. Im Falle, dass jemand die IP-Adresse des anderen nicht auswendig weiß, hilft das Feature indem es sämtliche IP-Adressen auflistet, die im Netzwerk verfügbar sind. Im Durchschnitt dauert dieser Prozess 5-8 Sekunden, anhand eines Testbeispiels in meinem Heimnetzwerk (Abbildung 1.4), kann man das Ergebnis eines „Quicksearches“ näher betrachten. Im weiteren Verlauf besteht die Möglichkeit die angezeigte IP-Adresse sich selbst (Server) oder dem anderen Spieler (Client) zuzuweisen (Abbildung 1.5), damit das Starten bzw. Verbinden reibungslos und ohne Komplikationen verlaufen kann.

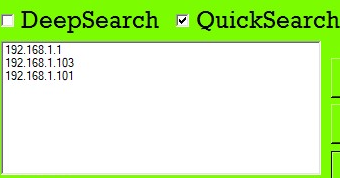


Abbildung 1.4

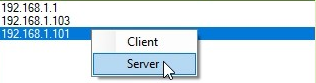


Abbildung 1.5

Das „Deepsearch“ Feature (Abbildung 1.7) ist dafür gedacht zusätzliche Informationen über andere Geräte im Netzwerk herauszufinden, falls jemand nur den Gerätenamen des anderen weiß und nicht die IP-Adresse bzw. MAC-Adresse. Dieser Vorgang dauert vergleichsweise zu „Quicksearch“ tendenziell länger, jedoch hängt dies stark von den Computern im Netzwerk ab.

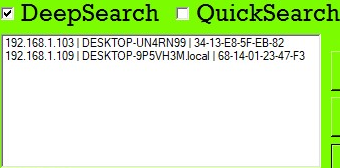


Abbildung 1.7

**Der „Arprequestprozess“ im Fokus**

Bevor man Arprequests an alle Teilnehmer in einem Netzwerk schickt, muss man zuerst alle Netzwerkkarten ermitteln, die in Betracht gezogen werden können für eine eventuelle Verbindung. Hierfür wird eine foreach-Schleife angewandt, um gewisse Eigenschaften jedes Interfaces zu ermitteln und diese dann einer „Listbox“ zu übergeben, welche als Beispiel die in Abbildung 1.7 wäre. In dieser Schleife wird wie vorhin beschrieben, alle irrelevanten Möglichkeiten übersprungen, da diese für unsere Zwecke für den Benutzer nur irreführend wären, falls diese im GUI erscheinen würden.

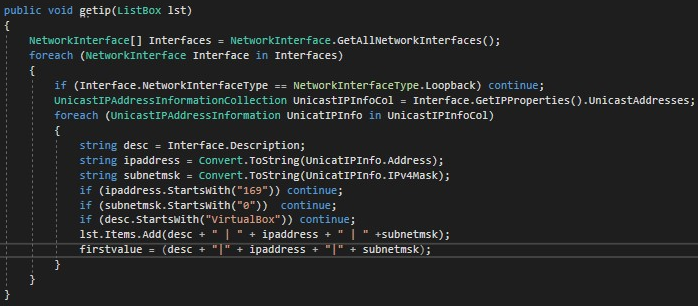


Abbildung 1.8

Nach Auswahl des richtigen Netzwerkadapters über das GUI wie in Abbildung 1.6 beschrieben, kommt es zum eigentlichen Vorgang, welcher im Hintergrund passiert. Die Methode „getSubnet“ (Abbildung 1.9) mit dem Parameter einer Textbox, ist die erste Methode, die beim Netzwerkscan ausgeführt wird. Aus dem Inhalt der Textbox, in welcher die Subnetzmaske steht werden 4 Sektoren gebildet (8bit).

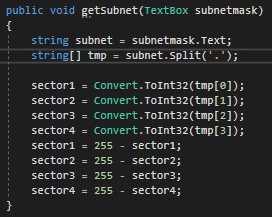


Abbildung 1.9

**Was ist eine IP-Adresse bzw. Subnetzmaske?**

Um den ganzen Vorgang besser verstehen zu können muss man sich erstmal im Klaren sein was eine IP-Adresse und Subnetzmaske ist und welche Rolle diese in unserem Falle spielt.Eine IP-Adresse darf in einem Netzwerk nur einmal vergeben werden. Dies ist notwendig, damit man Datenpakete richtig adressieren kann. Die IPv4-Adresse, welche in diesem Fall zum Einsatz kommt, besteht aus 4 Sektoren von 0-255, welche jeweils 8Bit entsprechen. Mithilfe der Subnetzmaske kann noch zusätzlich ermittelt werden, welcher Teil dieser IP-Adresse für das Netz bzw. die Hosts genutzt werden soll. Eine Subnetzmaske besteht ebenfalls aus 4 Sektoren von 0-255, welche ebenfalls jeweils 8Bit entsprechen. Aus diesen zwei Informationen IPv4 Adresse und Subnetzmaske lässt sich nun das Subnetz ermitteln, in welchem die jeweilige IPv4-Adresse vorzufinden ist. In einem konkreten Beispiel (Abbildung 1.1), werden durch die beiden Parametern und einer logischen UND Verknüpfung ein Netz gebildet. Die logische UND Verknüpfung bestimmt binär, was Netzteil bzw. Hostanteil ist.

Dabei kann die Tabelle in Abbildung 1.2 sehr hilfreich sein, da diese besagt, dass nur 1 und 1 wieder 1 ergibt. Dies bedeutet, dass die IPv4 Adresse 192.168.1.104 mit der Subnetzmaske 255.255.255.0 ein Netz von 192.168.1.0 ergibt. Somit ist der Netzanteil 24 Bit groß und der Hostanteil 8 Bit groß. Der Hostanteil ist nämlich genau der, der für unsere weiteren Berechnungen von Relevanz ist.



Abbildung 1.1

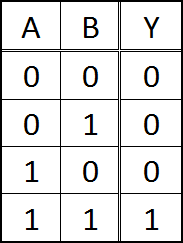


Abbildung 1.2

Daraus kann man Schlussfolgern, dass der erste Host mindestens eine IP-Adresse von 192.168.1.1 besitzt und maximal eine IP-Adresse von 192.168.1.254.

Der Hintergedanke ist es von diesen einzelnen Sektoren den Wert zu bekommen, welcher im späteren Verlauf (Abbildung 2.0) an eine if-Abfrage und Schleife weitergegeben wird, welche alle möglichen Hosts, des Netzwerkes durchgeht.

Zuerst wird in einer if-Abfrage überprüft, ob es sich zumindest um ein /24 Netz handelt. Falls dies zutrifft, wird eine Schleife mit einer gewissen Anzahl, welche wir vorhin berechnet haben, ausgeführt. Falls dies nicht zutrifft, wird überprüft ob es sich zumindest um ein /16 Netz handelt, danach kommt es zum gleichen Prozedere und anschließend werden die Ergebnisse dieser Operationen in eine Liste geschrieben, welche wir später noch benötigen.

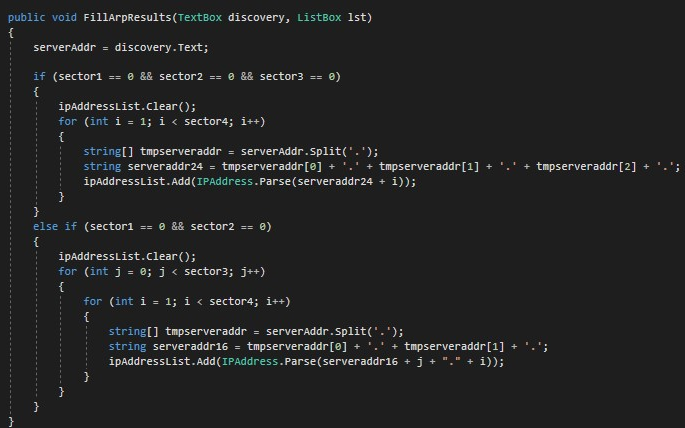


Abbildung 2.0

Bei „Quicksearch“ (Abbildung 2.1) wird für jede IP-Adresse in dieser vorher befüllten Liste ein Thread gestartet, welcher einen Arp-Request (Abbildung 2.2) aussendet und bei Erfolg diese IP-Adresse in die Listbox hineinschreibt, bei Misserfolg wird diese nicht weiter beachtet. Aufgrund einer hohen CPU-Auslastung dieser Requests, wurde das Starten eines neuen Threads mit 25 ms verzögert.

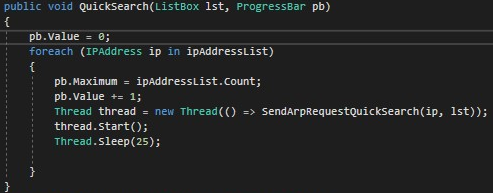


Abbildung 2.1

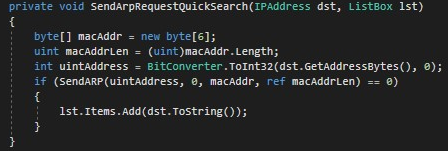


Abbildung 2.2

Bei „Deepsearch“ (Abbildung 2.3) passiert exakt dasselbe jedoch werden bei dem Senden des Arp-Requests noch weitere Faktoren miteinberechnet bevor diese Methode eine IP-Adresse hinzufügt (Abbildung 2.4). Es wird nämlich zusätzlich durch einen Ping (timeoutzeit 90ms) an das jeweilige Gerät getestet, ob dieses auch antwortet, um somit noch darüber hinaus zu garantieren, dass eine reibungslose Kommunikation gewährleistet wird. In Kombination mit Hostnamen und MAC-Adresse werden diese Informationen bei Erfolg in die Listbox hineingeschrieben.

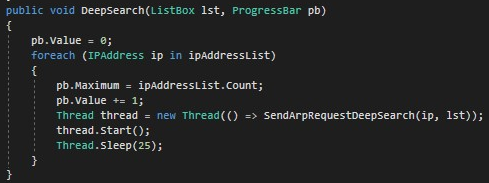


Abbildung 2.3

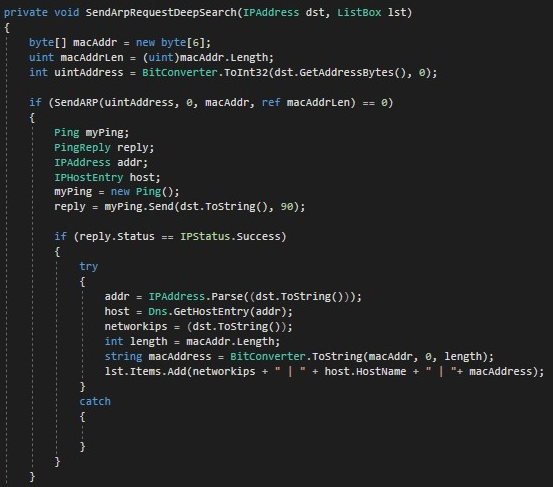


Abbildung 2.4