Multiplayermodus für das Internet

Serie Realbasic, Folge 4 Internet-Spiele sind Kult, und warum nicht mal eine Partie „Mensch ärgere Dich nicht“ mit einem Partner in Asien oder in den USA spielen? In dieser Folge programmieren wir die dafür nötigen Netzwerkgrundlagen nach dem Client/Server-Prinzip

von Christian Schmitz

EINSTIEG

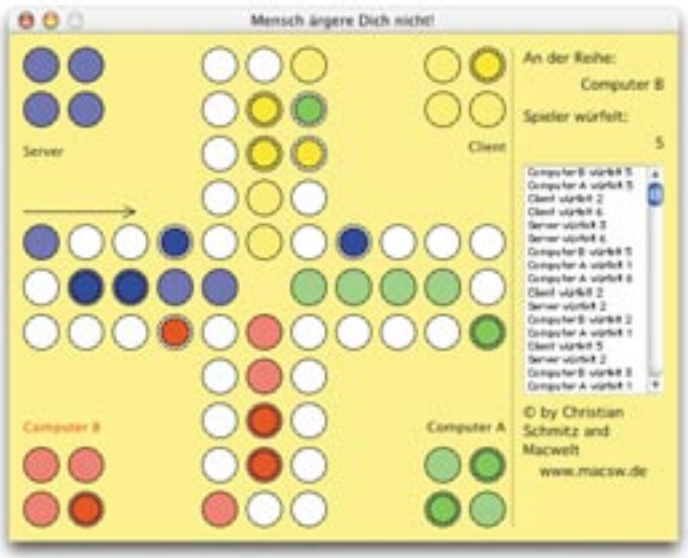
Realbasic ist eine leicht zu erlernende Programmiersprache, mit der auch Anfänger schnell zurecht kommen. In dieser Folge lernen wir, wie wir die „Mensch ärgere Dich nicht“-Simulation für das Internet anpassen, um es mit Spielern rund um die Welt zu spielen.

FÜR UNSER NETZWERKSPIEL beschränken wir uns zunächst auf zwei Spieler. Wir legen dazu ein neues Fenster mit dem Titel „Netzwerkspiel“ an, in das wir einige Steuerelemente, wie in der Abbildung auf Seite 67 zu sehen, platzieren. Der Schließknopf unten rechts führt lediglich den Befehl „hide“ aus, um das Fenster verschwinden zu lassen.

Wir dienen dem Imperium

In unser Spielfenster legen wir ein Socket-Steuerelement und geben ihm den Namen „sock“. In der Eigenschaftenpalette stellen wir für das Socket den Port 20 000 ein, damit die Kontaktaufnahme nur zwischen unseren Spielen funktioniert und wir keinen anderen Serverdienst (zum Beispiel: Webserver auf Port 80) behindern. Im Netzwerkdialog brauchen wir eine weitere Zugriffsmöglichkeit auf dieses Socket, weshalb wir eine Eigenschaft „sock as socket“ anlegen und diese später mit einem Verweis auf das Socket „sock“ im Spielfenster verbinden. Wir erhalten dadurch zwei Referenzen auf ein Objekt.

Der Startknopf für den Server ruft den Realbasic-Befehl „Sock.Listen“ auf und teilt damit dem Socket mit, dass es auf eingehende Anforderungen warten soll. Jedes andere Gerät im Netzwerk, das unsere IP-Adresse erreicht, kann nun eine Verbindung zu diesem Socket aufbauen.



Server und Client
Hier sieht man, wie zwei Computergegner, ein Server und ein Client, gemeinsam über das Internet ein Spielchen wagen.

Damit wir im Spiel wissen, ob wir Server oder Client sind, legen wir zwei Eigenschaften „client as boolean“ und „server as boolean“ im Codeeditor des Netzwerkdialogs an.

Um den Server im Netzwerkdialog starten zu können und automatisch die anderen Buttons zu deaktivieren, fügen wir den Code aus Listing 1 in den Action-Event des Startknopfs ein. Passend dazu folgt im Action-Event des Stoppbuttons der Code aus Listing 2.

Das Socket lässt sich damit nun schon auf Empfang einstellen und wieder beenden. Damit der Dialog im Programm erscheint, ergänzen wir im Menü einen Eintrag für das Netzwerkspiel, den wir im Spielfenster mit dem Netzwerkdialog verbinden (siehe Listing 3). Die Zeile „netzwerkspiel.sock = sock“ übergibt dabei eine Referenz auf das Socket im Spielfenster an den Einstelldialog. Der Server soll in einem Textfeld seine eigene Adresse

Serie: Realbasics für Profis

1	Interface	Heft 12/2001
2	Computerspieler	Heft 01/2002
3	Mensch spielt mit	Heft 02/2002
4	Multiplayer	Heft 03/2002
5	Kompilieren und Release	Heft 04/2002

(IP) anzeigen. Die aktuelle IP des Computers findet man in der Eigenschaft „Localaddress“, so dass die Zeile „EigeneIP.text = sock.LocalAddress“ die IP-Adresse des Mac in das Textfeld „EigeneIP“ schreibt. Da aber der Netzwerkdialog kein eigenes Socket hat, müssen wir das Gleiche noch mal im Menü-Handler machen.

Wir sollten nicht vergessen, den Menü-Handler im Event „EnableMenuItems“ des Spielfensters durch eine Zeile „spielnetz.enabled = not isnetzwerkSpiel and not isnetzwerkSetup“ zu aktivieren.

Der Client meldet sich an

Wenn der Server läuft, wollen wir uns mit einem Client anmelden. Dazu brauchen wir ein paar Zeilen Code im ConnectButton-Action-Event, wie uns Listing 4 zeigt. Hier deaktivieren wir zunächst alle Buttons und das Eingabefeld, dann leiten wir die eingegebene Adresse an das Socket und starten einen Verbindungsversuch. Nun müssen wir mit einer Reaktion des Sockets rechnen, weshalb wir im Spielfenster ein paar Variable für den aktuellen Status benötigen. Wir nehmen die Variablen „IsNetzwerkSetup as boolean“, um zu vermerken, dass der Netzwerkdialog mit korrekten Daten ausgefüllt ist. Die Variable „IsNetzwerkSpiel as boolean“ wird wahr, wenn ein Netzwerkspiel läuft. Vor der Zeile „netzwerkspiel.showmodal“ setzen wir die Variable „IsNetzwerkSetup“ auf True und nach dem Aufruf des Dialogs wieder auf False. Man beachte den Unterschied zwischen .show und .showmodal, denn bei .show folgt direkt die nächste Programmzeile während bei .showmodal der Rechner darauf wartet, dass der Anwender den Dialog schließt. Wie diese Routine aussieht finden wir in Listing 5.

Schauen wir jetzt einmal auf das Socket im Spielfenster. Dieses Socket verfügt über vier Events wobei wir den Error-Event zuerst bedienen. Es gibt über 100 Gründe (Onlinehilfe) warum der Error-Event aufgerufen wird, uns interessiert jedoch nur, dass bei einem Fehler keine IP-Verbindung zu Stande gekommen ist. Dies passiert, wenn entweder auf der Gegenseite kein Server-Socket auf eine Verbindung wartet oder die IP nicht stimmt. Tritt also ein Fehler auf, während der Netzwerkdialog geöffnet ist, dann soll unser Programm den Fehler verarbeiten und anzeigen. Wie das im Code aussieht erfahren wir in Listing 6.

Bei geöffnetem Netzwerkspiel-Dialog reagieren wir auf den Fehler, indem wir als Server auf weitere Clients warten oder als Client eine Fehlermeldung anzeigen. In allen anderen Fällen interessiert uns der Error-Event nicht, was wir aus Listing 7 entnehmen.

Im Spielfenster kann über das Socket-Steuerelement auch der Event „Connected“ kommen, der uns anzeigt, dass eine Verbindung besteht. Sind wir jedoch kein Server, bleibt unsere Spielhölle für den Kunden geschlossen und dessen Verbindung wird beendet. Wir schreiben also in den Connected-Event des Sockets den Code aus Listing 8

Wir rufen hier die Methode „connected“ im NetzwerkSpiel-Dialog auf, die noch nicht existiert. Wir erzeugen sie und füllen sie mit dem Code aus Listing 9.

Wir haben nun eine Verbindung und sind Server. In diesem Fall läuft ein Spiel, wir schließen den Dialog jedoch nicht sofort, denn wir müssen dem Spielfenster noch mitteilen, dass wir der Server sind. Dies machen wir im Spielfenster beim Menü-Handler für ein neues Netzwerkspiel. Vor dem „return true“ ist noch Platz und wir ergänzen den Code, der den aktuellen Server/Client-Status setzt. Danach schließen wir den Dialog.

Nun teilt der Server dem Client mit, dass der seinen Netzwerkdialog schließen und sich auf das Spiel vorbereiten soll. Wie der Befehlscode dazu aussieht ist bedeutungslos. Wir schicken der Einfachheit halber den Text „Get Ready!“ + chr(13). Mit chr(13) ist das Zeichen für die Eingabetaste gemeint, die auf dem Mac die Bedeutung eines Zeilentrenners hat. Wir ergänzen also den Menü-Handler „Spiel-Netzwerkspiel“ um die Zeilen aus Listing 10.

Zum Schluss rufen wir die Methode „CreateNetzwerkSpiel“ auf, die wir als leere Methode anlegen. Auch die Eigenschaften „Server as boolean“ und „Client as boolean“ erzeugen wir vorab im Fenstercode.

Das Programm lernt lesen

Fließen Daten über eine Netzwerkverbindung, ruft Realbasic den Event „DataAvailable“ auf. Dort kann man über die Socket-Methoden „Lookahead“ und „Read(anzahl)“ beziehungsweise „Readall“ den Datenpuffer bearbeiten.

Der Code für den DataAvailable-Handler, der ungefähr alle 1,5 KB (Größe eines TCP/IP Pakets) aufgerufen wird, sollte möglichst schnell arbeiten, damit die Übertragung ohne Datenstau und Wartepausen abläuft.

Tipp | Ideen für Erweiterungen

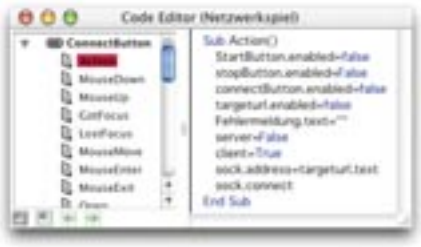
Programmieren lernt man, indem man es tut. Nehmen Sie sich Zeit und bauen Sie das Spiel weiter aus:

- Chat und Cheat? Wie wäre es mit einem kleinen Edit-Feld, das beliebigen Text über das Socket abschickt. Doch Achtung: Stellt man dabei einer Chat-Meldung keine besondere Kennung voran, dann wird sie die Gegenseite als Befehl interpretieren, was sich letztlich auch gut zum Mogeln eignet.
- Sicherheit? Unser Programm ist nicht gegen das Empfangen von Datenmüll gesichert. Eingegangene Befehle werden bislang auch noch nicht bestätigt.
- Zu schwer? Wer meint, dass der Computergegner zu schwer zu besiegen ist, kann den Computer so programmieren, dass er alle möglichen Züge ermittelt und dann den besten aussucht. Dabei kann der Computer versuchen, maximal viele andere Figuren rauszuwerfen oder möglichst wenige.
- Mehr Mitspieler? Was mit zweien geht, geht auch mit mehreren. Erweitern Sie das Spiel doch auf drei oder vier Netzspieler.
- Zu spät und Server zu? Man könnte auch erlauben, sich später einzuloggen und einen Computerspieler zu übernehmen. Genau so gut kann man sein Spiel abgeben, wenn die Lust mitten im Spiel nachlässt.

Listing 1

Listing 2

Listing 3



Listing 4

Listing 5

Listing 6

Tipp | TCP/IP in Realbasic

In Realbasic gibt es ein Steuerelement namens „Socket“, mit dem man über das Netzwerk-Protokoll TCP/IP Daten zwischen zwei Computern austauschen kann. Dazu wartet ein Socket beim Server auf eine Verbindung zu einem Client. Jeder Computer in einem lokalen Netzwerk oder im Internet hat eine eindeutige Identifikations-Nummer wie beispielsweise 129.168.1.10 oder einen symbolischen Namen dafür wie zum Beispiel „www.macwelt.de“. Innerhalb von TCP/IP verfügt jeder Computer über 65536 logische Ports. Vergleichbar mit einer Firmentelefonnummer und der Durchwahl. Viele dieser Ports sind fest vergeben. Bei www.macwelt.de findet man unter Port 80 die Internet-Seite der Macwelt und unter Port 110 und Port 25 den E-Mailserver der Macwelt-Redaktion.

Zuerst kopieren wir den Puffer in eine Stringvariable ohne ihn zu löschen. Wir durchsuchen den String dann auf ein Zeilenende. Wenn wir eines finden, befindet sich mindestens eine Meldung im Puffer, die wir inklusive des Zeilenendes auslesen. Nun überprüfen wir, ob die Meldung der „Get Ready!“-String ist, falls ja, wird unser Mac zum Client erklärt (der Server wird diese Meldung hoffentlich nie bekommen!) und blendet das Netzwerkspielfenster aus. Den Realbasic-Code hierzu sehen wir in Listing 11.

Sobald der Dialog geschlossen wird, führt Realbasic den Code hinter „showmodal“ aus und sowohl der Server als auch der Client starten die Routine „CreateNetzwerkSpiel“, der wir uns jetzt zuwenden.

Schnell entwickeln durch Kopieren und Einfügen

Kreatives Kopieren und Einfügen erleichtert uns das Tippen der Methode „CreateNetzwerkSpiel“. Wir kopieren den kompletten Code aus „NeuesSpiel“ und füllen damit die noch leere neue Methode. Alles was der Dialog „NeuesSpiel“ braucht, löschen wir. Die Spieler benennen wir wie folgt:

```
Spieler(0) = new Spieler("Server",
rgb(0,0,255), 0, self)
Spieler(1) = new Spieler("Client",
rgb(255,255,0), 10, self)
Spieler(2) = new Spieler("Computer A",
rgb(0,255,0), 20, self)
Spieler(3) = new Spieler("Computer B",
rgb(255,0,0), 30, self)
```

Am Ende der Routine legen wir fest, wer als menschlicher Spieler agiert:

```
Spieler(0).Mensch = true
Spieler(1).Mensch = true
Spieler(2).Mensch = false
Spieler(3).Mensch = false
```

Zum Schluss stellen wir ein, dass es sich um ein Netzwerkspiel handelt und starten, falls wir nicht der Client sind, den Timer.

```
isnetzwerkSpiel=true
if not client then
nexttimer.mode=2
end if
```

Das Spiel läuft nun! Jetzt kann allerdings noch der Fall auftreten, dass unser Mac weder Client noch Server ist. In diesem Fall spielt der Computer allein.

Der Client dient als reiner Befehlsempfänger

Da wir an dieser Stelle nicht viel Platz für lange Code-Listings im Heft haben, gestalten wir unseren Client etwas weniger intelligent. Als braver Dienstmann wird er die Befehle vom Server ausführen und fleißigst die Reaktionen der Mitspieler an seinen Meister schicken.

Im Folgenden bauen wir viele kleinere Änderungen ein und beschreiben diese aus Platzgründen nur kurz. Zunächst schicken wir in der Methode „log“ im Spielfenster alle Einträge als Kopie zum Client:

```
Sub log(s as string)
  liste.insertRow 0,s
  if server then
    sock.write "log"+s+chr(13)
  end if
End Sub
```

In der Methode „Nextplayer“ sind ein paar Änderungen notwendig. Wir schützen den ganzen Befehlsblock mit einem „if not client then“ davor, dass er auf dem Client ausgeführt wird.

Hinter der Ausgabe des Spielernamens mittels „Spielername.Text = Currentplayer.Name“ schicken wir ein Paket an den Client, damit dieser den Namen des Spielers anzeigt:

```
if isnetzwerkSpiel and server then
   sock.write "next"+str(spielercounter)
   +chr(13)
end if
```

Weiter unten schicken wir hinter der Zeile „war.movetohaus“ eine Meldung an den Client, damit er die Spielfigur an der Startpo-

Listing 7

Listing 8

Listing 9

sition des aktuellen Spielers hinauswirft:

```
if server then
  sock.write "kick"+str(currentplayer.
  offset)+chr(13)
end if
```

Hinter der Zeile „n.movetobrett(currentplayer.offset)“ melden wir dem Client eine neue Figur:

```
if server then
  sock.write "neuefigur"+chr(13)
end if
```

In der Methode „menschzug“ verschicken wir einen Auftrag an den Client dort einen Zug auszuführen, wenn der Client an der Reihe ist.

```
if spielerCounter=1 and server then
  sock.write "zug"+chr(13)
end if
```

Die Methode „klickinFeld“ wird sowohl beim Client als auch beim Server aufgerufen. Aber man darf die Figur eines anderen Spielers nicht verschieben, weshalb wir da etwas komplizierter prüfen müssen, bevor wir einen Klick erlauben. Wenn ein Netzwerkspiel läuft, müssen wir dem Client den Zug mitteilen, damit dieser ihn bei sich ausführen kann. Falls das System nicht der Client sein sollte, starten wir anschließend den Timer zum Spielerwechsel. Den neuen Code der Methode „KlickInFeld“ sehen wir in Listing 12.

Jetzt wird es ernst! Die Befehlsausführung

Weiter geht es im Steuerelement „Sock“. Wir erweitern den Event „DataAvailable“ um neue Variablen: „dim f, war as figur“, „dim w as integer“, „dim t as string“ und „dim b as brettfeld“. Bisher haben wir nur einen Befehl pro Paket zugelassen. Durch unsere Log-Funktion kommen aber wesentlich mehr Pakete an und das Mac-OS wird zur Steigerung der Geschwindigkeit mehrere Pakete zusammen ausliefern. Mit einer „Do-Loop“-Schleife:

```
do
loop until n=0
```

gehen wir solange auf Paketsuche, bis wir keines mehr im Puffer finden. Im weiteren Verlauf wertet unser Programm alle Befehle aus und führt sie durch. Sie finden den kompletten Code im Projekt-File auf der Abo-CD beziehungsweise im Internet unter www.macwelt.de/_magazin. Die Kommentare sollten zur Erklärung ausreichen.

Die Figuren brauchen einen Index

Im Netzwerkspiel müssen wir die Figuren durch eine eindeutige Nummer auswählen können. Bis jetzt befinden sich die Figuren in

Listing 10

Listing 11

Listing 12

einem Feld, und wissen ihren Index nicht. Daher bekommt die Klasse Figur eine neue Eigenschaft „index as integer“ und wir ändern den Konstruktor, indem wir die Methode Figur wie folgt erweitern:

```
Sub Figur(c as color,i as integer)
  farbe=c
  index=i
end sub
```

Dadurch müssen wir auch den Konstruktor des Spielers anpassen, indem wir den neuen bei jedem Aufruf Index mit übergeben. Folgende Zeilen leisten dies:

```
figur(0)=new figur(c,0)
figur(1)=new figur(c,1)
figur(2)=new figur(c,2)
figur(3)=new figur(c,3)
```

In der Methode „würfeln“ schicken wir nun die gewürfelte Augenzahl nach der Zeile „Würfel = rnd*6 + 1“ zum Client:

```
if Fenster.isnetzwerkSpiel then
  Fenster.sock.write "würfeln"
  +str(würfel)+chr(13)
end if
```

Neuer Dialog im Spiel Der Netzwerkdialog erlaubt es, Server und Client in einem einzigen Fenster zu steuern.

Weiterhin fehlt in der Methode „gehefelder“ noch eine Benachrichtigung an den jeweils anderen Rechner. Wir schreiben hinter der Zeile „gehemitFigur f,c“:

```
if fenster.isnetzwerkSpiel then
  if (fenster.server and fenster.spieler
  counter<>1) or (fenster.client and
  fenster.spielercounter=1) then
    fenster.sock.write"Move"+str(f.Index)
    +" "+str(würfel)+chr(13)
  end if
end if
```

Wenn wir unser Projekt jetzt kompilieren, sollte die Netzwerkfähigkeit funktionieren. Probieren Sie ein Spiel über das Internet, vielleicht mit dem Onkel aus Amerika?

Fazit

Der aktuelle Stand unserer Simulation zeigt, wie man ein einfaches Server/Client-Prinzip unter Realbasic implementiert. Das öffnet nicht nur Spielern Tür und Tor zum Internet. Auch ernsthafte Anwendungen sind denkbar. In der nächsten Folge widmen wir uns dem finalen Schliff unseres Programms sowie dem Kompiliervorgang und der Optimierung. *cm* ✖

ONLINE....

Alle Quellcodes zu dieser Serie finden Sie im Internet unter www.macwelt.de/_magazin und auf der Heft-CD, weitere Programmierbeispiele für Realbasic bei der Firma ASH unter www.application systems.de