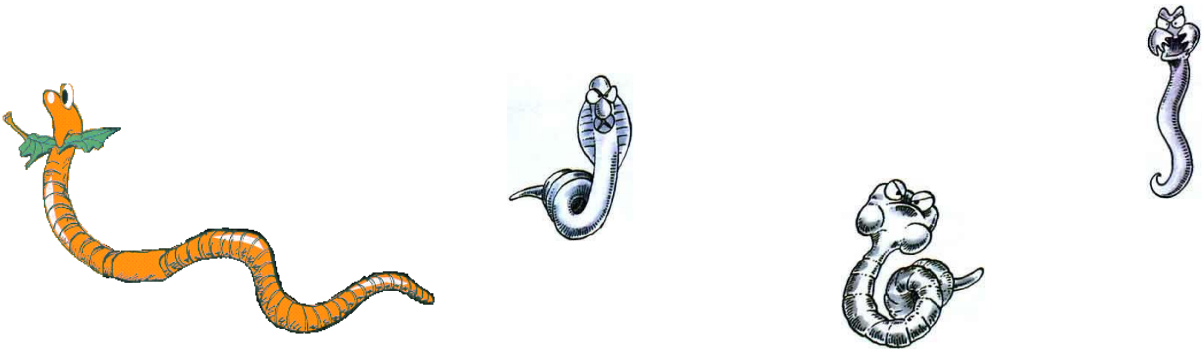
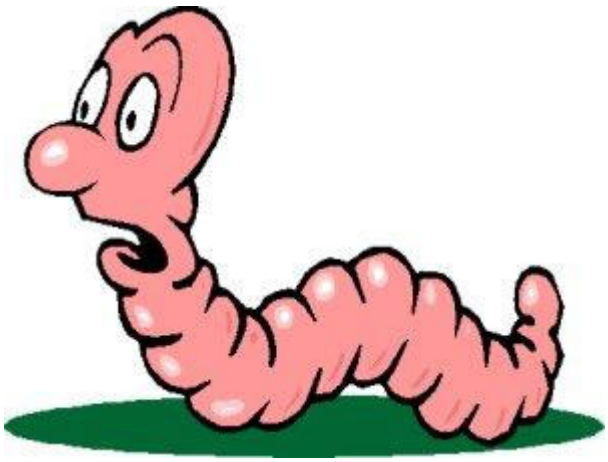


Übungsprojekt 'Wurm'

Dokumentationsbeispiel



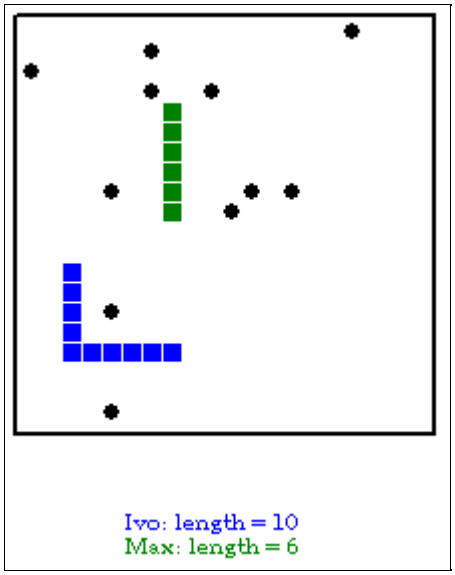
Ein netzwerkfähiges Spiel für mehrere Teilnehmer

Dateiname:	Projekt_Wurm_musterloesung_2.doc
Erstellt am:	3. Mai 2004
Letzte Änderung am:	3. Mai 2004
Erstellt von:	Ivo Oesch
Version:	6

Inhaltsverzeichnis

1 Aufgabenstellung

Es soll ein Wurm-Spiel implementiert werden. Bei diesem Spiel wird ein Wurm mittels der Cursortasten über ein Spielfeld gesteuert. Das Spiel ist für den Spieler zu Ende, sobald sein Wurm mit dem Rand, einen anderen Wurm oder sich selbst zusammenstösst. Innerhalb des Spielfeldes werden periodisch kleine Punkte verteilt, die vom Wurm gefressen werden können. Wenn ein Punkt nach einer gewissen Zeit nicht gefressen wurde, soll er wieder verschwinden. Die Futterpunkte sollten nicht mehr als etwa 2-3% der Spielfläche bedecken. Mit jedem gefressenen Punkt wird der Wurm um ein Feld länger. Der Wurm wird jeweils nach 10 Schritten automatisch um ein Feld länger. Zu Beginn hat der Wurm eine Länge von 1. Der Wurm bewegt sich alle 200ms um einen Schritt weiter, normalerweise in dieselbe Richtung wie beim vorhergehenden Schritt, ausser der Spieler hat eine neue Richtung vorgegeben. Sie können zur Steigerung der Schwierigkeit die Zeitschritte nach einer bestimmten Spielzeit verkleinern.



Jeder Wurm soll sich durch seine Farbe von den anderen unterscheiden. Das Spiel soll für mindestens 2 Spieler ausgelegt werden. Wenn via Netzwerk gespielt wird, sollten auch mehr als 2 Spieler mitspielen können. Von jedem Spieler soll der Name abgefragt werden, und in der Farbe seines Wurms unterhalb des Spielfeldes dargestellt werden. Neben dem Namen soll die aktuelle Länge des Wurms erscheinen. Es sollen zwei High-Score Listen geführt werden, eine für Einzelspieler, dabei wird die erreichte Länge des Wurms gewertet, und eine Mehrspielerliste, hier können sie selbst entscheiden was gewertet wird (Anzahl Siege (Letzter übriggebliebener Spieler, Länge des Wurms sobald man einziger Spieler ist, Verhältnis Siege zu Verlusten)) Die HighScore-Listen sollen dauerhaft in einer Datei abgespeichert werden und bei jedem Spiel aktualisiert werden.

Das Spiel wird über die Tastatur gesteuert: Der erste Spieler spielt mit den Cursortasten, der zweite mit den Funktionstasten:

Cursur Up/F2 -> der Wurm geht geradeaus weiter.
Cursor Left/F1 der Wurm dreht sich nach links,
Cursor Right/ F3 der Wurm dreht sich nach rechts.

(Wobei links und rechts immer von der Laufrichtung des Wurms aus gesehen sind.). Mit Escape kann das Spiel vorzeitig abgebrochen werden. (Sie dürfen selbstverständlich auch eine eigene Steuerungsphilosophie definieren).

Das Spielfeld ist in ein Netz aus quadratischen Feldern unterteilt, Wurmsegmente und Futter bewegen sich nur von Feld zu Feld und können sich nicht zwischen den Feldern aufhalten.

Netzwerkfähigkeit: Das Spiel soll auch über ein Netzwerk gespielt werden können. Dazu soll ein Computer als Server dienen, das Spiel läuft auf diesem Computer, die anderen Teilnehmer agieren als Client. Die Clients senden nur die Tastendrücke an den Server und empfangen Zeichenbefehle von Server, so dass sich auf ihrem Bildschirm dasselbe Abbild zeigt wie auf dem Server. Ein Client muss sich zuerst beim Server anmelden, bevor er an einem Spiel teilnehmen kann, und ein Server muss Anmeldungen akzeptieren können. Bei Netzwerkspielen sollen mindestens 2 Spieler teilnehmen können.

2 Richtlinien

Für dieses Projekt gelten folgende Richtlinien:

Jedes Modul besteht grundsätzlich aus zwei Dateien, einer Schnittstellendefinitionsdatei (Headerdatei .h-File) und einer Implementationsdatei (.c-File). Einzige Ausnahme ist das main-Modul mit der Funktion main(). Ein Modul darf weitere (private) Includedateien besitzen, diese haben die Endung .i.

Für projektweite oder projektübergreifende Typendefinitionen dienen die Dateien general.h und project.h. Darin sollen aber wirklich nur völlig Modulunabhängige Typen definiert werden, wie z. B Byte oder Word für 8-Bit resp. 16-Bit Werte sowie Steueranweisungen (#defines) für Versionsabhängigkeiten.

Die Namen der Header und der Implementationsdatei entsprechen jeweils dem Modulnamen, soweit möglich und sinnvoll.

Der Selbstschutz in den Headerdateien erfolgt in der Form

```
#ifndef NAME
#define NAME

Schnittstellendefinition
#endif
```

Wobei NAME wie folgt aufgebaut ist MODULNAME_DATEINAME_H, Modulname ist fakultativ, aber Dateiname und H sind Pflicht.

Jedes Modul ist mit einem Modulheader ausgestattet, jeweils in der Schnittstellen und der Implementationsdatei. Der Header enthält den Modulnamen, den Dateinamen, einen kurzen Funktionsbeschrieb zum Modul, die Namen aller zur Verfügung gestellten Funktionen, den Namen des Autors, das Erstellungsdatum sowie eine History.

Der Aufbau der Dateien hat strukturiert zu erfolgen. Die einzelnen Teile haben in der folgenden Reihenfolge zu erscheinen: Modulheader, Includes, Typendefinitionen, Konstantendefinitionen, Globale Variablen, lokale Variablen, Deklaration lokaler Funktionen, Definition der Funktionen.

Jede Funktion ist mit einem Funktionsheader ausgestattet. Der Header enthält den Funktionsnamen, einen kurzen Funktionsbeschrieb, die Namen und Funktion der Argumente, eine Beschreibung des Rückgabewertes, den Namen des Autors, das Erstellungsdatum sowie eine History.

Der genaue Aufbau der Header und der Module ist den Beispielen Template.c und Template.h zu entnehmen.

Funktionen sollen klar definierte Aufgaben haben.

Namen von Bezeichnern sollen Aussagekräftig sein, und die Funktion des entsprechenden Objektes erklären oder andeuten.

Namen und Kommentare werden in englischer Sprache geschrieben.

Variablen und Funktionen werden grundsätzlich in Kleinschrift, mit einem grossen Anfangsbuchstaben und Grossbuchstaben bei Wortgrenzen oder zur Strukturierung geschrieben. Underscores dürfen nur in Ausnahmefällen verwendet werden.

Beispiele: Zahl, BenutzerEingabe, FindeGroessten, DeleteAllPlayers

Makros (#defines) werden grundsätzlich in Grossbuchstaben geschrieben, zu Strukturierung können Underscores verwendet werden.

Beispiele: PI, MAXIMAL_FIELD_WIDTH

Für jedes Modul ist grundsätzlich ein Programmierer zuständig, nur dieser darf Änderungen in den Dateien zu seinem Modul durchführen.

Der Code ist grundsätzlich mit sinnvollem und aussagekräftigem Kommentar zu versehen.

3 Analyse

Die Module für die Graphikdarstellung und die Netzwerkkommunikation werden zur Verfügung gestellt. Es müssen deshalb nur noch die Module für das eigentliche Spiel entworfen werden.

In einem ersten Schritt wird auf die Netzwerkfunktionalität verzichtet, das Spiel soll zuerst nur für lokale Spieler implementiert werden.

Auf die Highscoreliste wird verzichtet.

Die exakten Anforderungen an das Spiel können ansonsten der Aufgabenstellung entnommen werden.

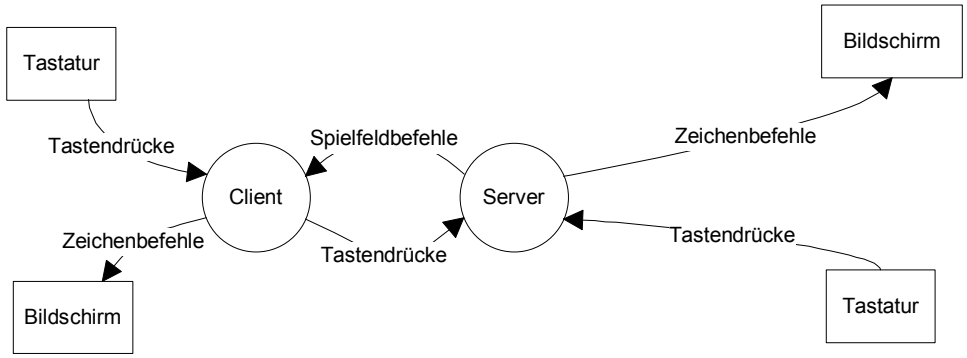
Beim Start des Spiels soll der Anwender entscheiden, ob er Client oder Server sein will. Der Server steuert das Spiel, der Client ist passiv.
Client-Seitig muss der Benutzer die IP-Adresse des gewünschten Servers angeben. Wenn der Server innert einer gewissen Zeit (Einige Sekunden) nicht reagiert, soll der Benutzer erneut zur Eingabe einer IP-Adresse aufgefordert werden und ein erneuter Verbindungsversuch gestartet werden. Wenn die Verbindung geklappt hat, startet der Clientbetrieb. Der Client kann mit der Escape Taste jederzeit aus dem Spiel aussteigen.
Der Server wartet beim Start auf Verbindungen durch Clients, bis der Benutzer das Spiel startet. Wenn sich mehr Clienten als vorgesehen anmelden, können sie nicht aktiv am Spiel teilnehmen, aber den Spielverlauf als Gäste betrachten. Clienten, welche sich während dem Spielverlauf abmelden, werden im aktuellen Spiel nicht mehr weiter beachtet. Der Server kann das Spiel auch jederzeit mit Escape abbrechen. Nach dem Ende des Spiels kann der Benutzer am Server ein neues Spiel mit allen angemeldeten Spielern starten, oder das Programm beenden. Am Server können ein oder zwei lokale Spieler mitspielen. Beim Starten des Servers muss angegeben werden, wieviel lokale und wieviel Netzwerkspieler am Spiel teilnehmen sollen.
Die Bildschirmausgabe soll ohne Flackern erfolgen.
Die Farben der einzelnen Würmer sollen gut voneinander unterschieden werden können.
Bei der Angabe von Spielernamen soll dafür gesorgt werden, dass ungewöhnlich lange Namen sinnvoll dargestellt werden und das Layout nicht beeinträchtigt wird. Die Namen dürfen notfalls bei der Darstellung gekürzt werden (Nicht aber in der Highscoredatei).

Globale Variablen sind möglichst zu vermeiden.

4 Design

4.1 Grobdesign

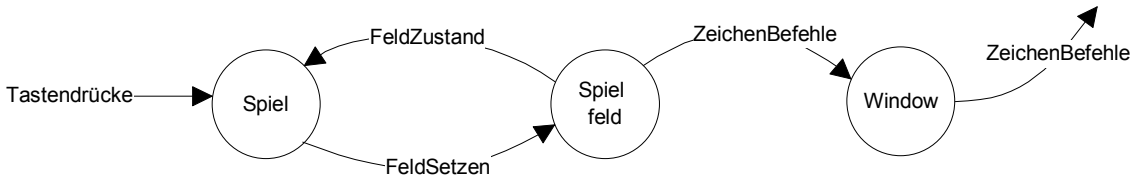
Der grobe Aufbau des Systems sieht etwa wie folgt aus:



Der Client sendet seine Tastendrücke an den Server weiter, und erhält seinerseits Informationen über den Spielzustand, welche er graphisch auf dem Bildschirm darstellt.
Der Server erhält Tastendrücke von den lokalen und den Netzwerkspielern, wertet diese aus, und sendet Informationen über den aktuellen Spielzustand an die Clients und stellt das Spielfeld auch lokal dar.

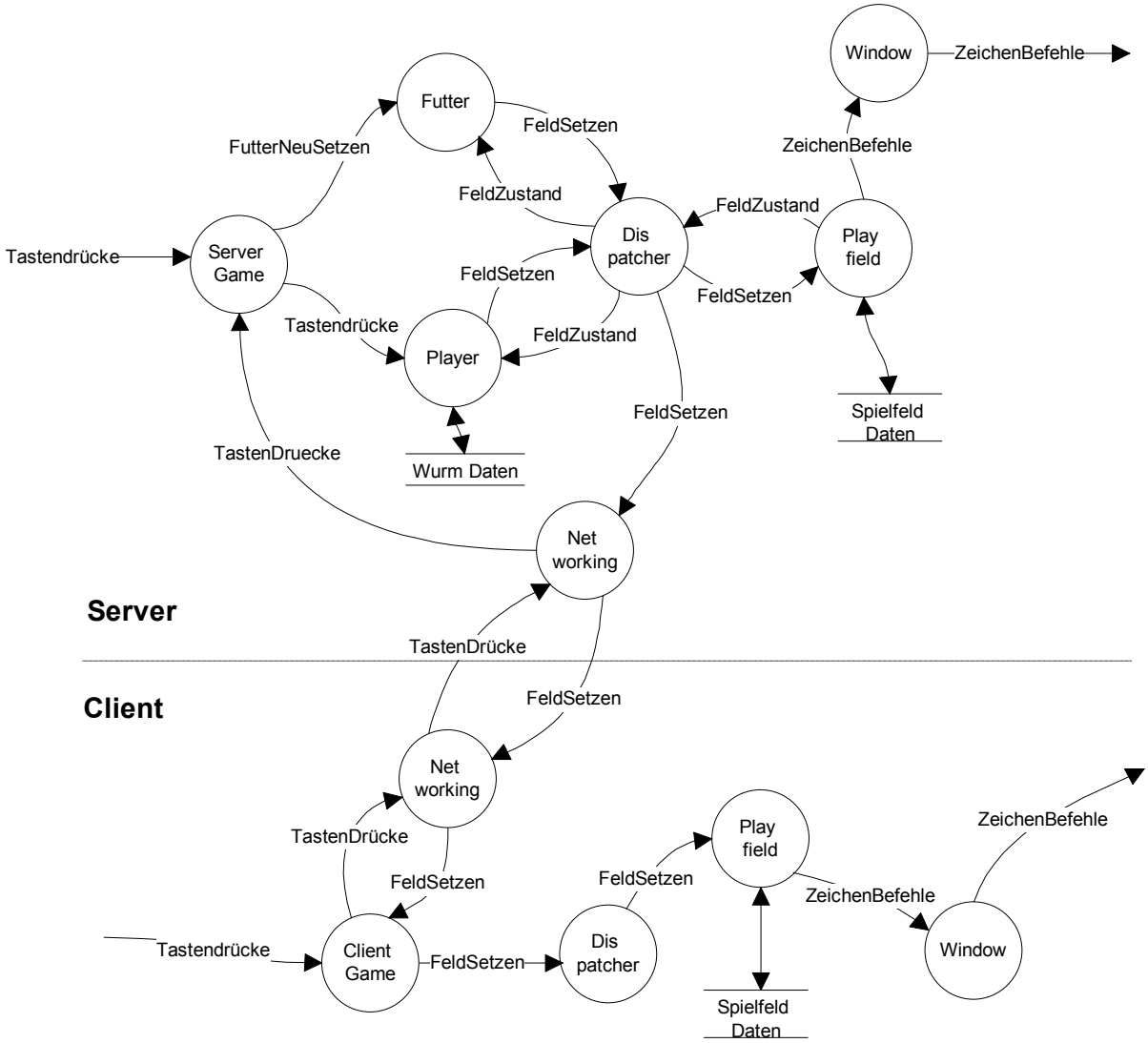
Das Spielfeld, die Spieler, das Futter und die Würmer werden vom Server verwaltet, der Client ist eine Art einfaches Graphikterminal, führt aber ein Abbild des Spielfeldes.

Grundsätzliche Struktur des Servers



Das Spiel steuert den ganzen Spielablauf, verwaltet die Würmer, die Spieler und das Futter. Das Spielfeld ist ein intelligentes Array, welches seinen Inhalt automatisch auch auf dem Bildschirm darstellt. Dazu darf aber nicht direkt auf das Array (via Index) zugegriffen werden, sondern es müssen Setz- und Lesefunktionen benutzt werden. Damit kann auch dafür gesorgt werden, dass nur Änderungen auf dem Bildschirm nachgeführt werden und so ein Flackern verhindert werden. Würde immer der ganze Bildschirm gelöscht und neu gezeichnet, würde unvermeidlich ein störender Flackereffekt auftreten.

Etwas detaillierter lässt sich das System in folgende Blöcke zerlegen:



Aufgaben der einzelnen Module

Gameserver Der Gameserver verwaltet den Ablauf eines Spiels, die Anzahl der Spieler, nimmt Tastendrücke entgegen und verteilt sie an die betroffenen Spieler. Hier wird auch der Start des Programmes und die Art des Spieles festgelegt.

GameClient Der Client ist eine Spezialversion des Servers und im gleichen Modul integriert

Dispatcher Verwaltet die angemeldeten Clients, und verteilt die Spielfeldbefehle an alle Clients sowie an die lokale Anzeige. Dieses Modul wird in einem ersten Schritt nur für lokale Spiele ausgelegt und kann durch das Spielfeld ersetzt werden. Später leitet es alle Befehle an das Spielfeld zusätzlich auch an alle Clients weiter. Anfragen an das Spielfeld werden nur an das lokale Spielfeld weitergeleitet. Spielfeldbefehle vom Netzwerk (Clientbetrieb) werden an das lokale Spielfeld weitergeleitet.

Playfield Verwaltet das Spielfeld, ist ein intelligentes Array mit Funktionen zum Setzten und Lesen von Feldern. Beim Setzen wird zugleich im Fenster gezeichnet.

Player Ist weiter unterteilt in die Module **Wurm** und **Player**. **Player** verwaltet die Informationen eines Spielers und seinen Wurm, es steuert auch den Wurm.
Wurm verwaltet und stellt den Wurm für einen Spieler dar.

Networking Zur Verfügung gestelltes Modul zur Netzwerkkommunikation

Window Zur Verfügung gestelltes Modul zur graphischen Ausgabe auf den Bildschirm

Futter Verwaltet die Futterpunkte, verteilt neue und entfernt alte Punkte

4.2 Grundsätzliche Ideen (Server)

Um die Struktur so einheitlich wie möglich zu gestalten, werden die **lokalen Spieler** gleich wie die **Netzwerkspieler** behandelt, sie erhalten lediglich einen ungültigen Wert für das Handle (Netzwerkverbindung) um sie von echten Netzwerkspielern zu unterscheiden.

Jeder empfangene **Tastendruck** wird inklusive Absender (Netzwerkhandle) grundsätzlich an alle Spieler weitergeleitet. Jeder Spieler entscheidet autonom für sich, ob der Tastendruck für ihn bestimmt war oder nicht. Die Entscheidung fällt aufgrund des Absenders. Somit braucht sich die Hauptroutine nicht um die Bedeutung der Tastendrücke zu kümmern, ausser denjenigen zum Unterbrechen oder Abbrechen des Spiels

Zeichenbefehle vom Spielfeld (Graphikbefehle) werden an das lokale, und via Telegramme an alle angemeldeten Klienten weitergeleitet.

Damit gefressene Futterpunkte, oder vom Wurm verlassene Felder **effizient** wieder **gelöscht** werden können, wird vom der Futterverwaltung eine Tabelle von allen Punkten und ihren Koordinaten geführt, und von der Wurmverwaltung wird eine Liste der Positionen seiner Elemente geführt, welche mit der Länge des Wurms wächst. So muss nicht jedes mal der ganze Wurm neu gesetzt werden, sondern es kann jeweils das letzte Segment gelöscht und die neue Kopfposition neu gezeichnet werden.

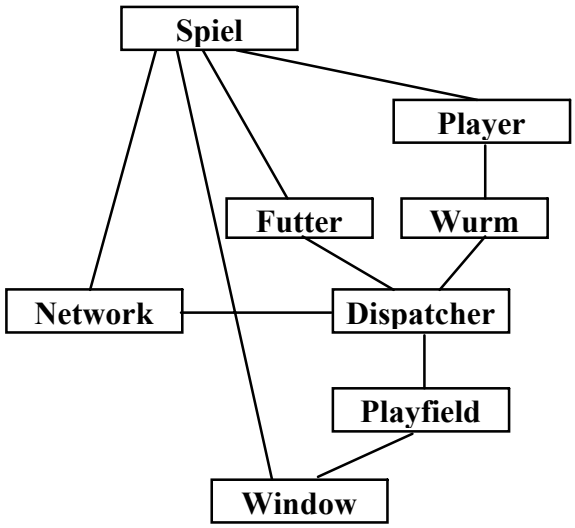
Kollisionen werden auf dem Spielfeld festgestellt. Bevor ein Wurm ein neues Feld betritt (überschreibt) wird geprüft, ob dieses Feld nicht bereits besetzt ist. Um zwischen Futter und Kollisionen unterscheiden zu können, werden Futterpunkte speziell markiert. Bei Würmern wird nur der RGB-Wert in den Bits 0-23 im Feld abgespeichert, bei Futter ist zusätzlich noch das Bit 28 gesetzt. Für Felder ausserhalb des Spielfeldes (Rand) wird der Wert 0xFFFFFFFF zurückgeliefert.

Während eines **Zeitschrittes** werden alle Tastendrücke an die Würmer weitergeleitet, und nach Ablauf des jeweiligen Zeitschlitzes erhalten alle Würmer den Befehl, sich in die zuletzt gewählte Richtung fortzubewegen. Die Würmer liefern danach ihre Länge zurück, oder wenn sie tot (Kollidiert) sind, einen negativen Wert.

Auf globale Variablen soweit wie möglich verzichten.

4.3 Detaildesign

Das gesamte Projekt setzt sich aus folgenden Modulen zusammen:



4.3.1 Das Modul Wormmain "-"

Das Modul enthält das Hauptprogramm (main()) für das gesamte Programm. Es lässt den Benutzer zwischen den Varianten Server, Client oder lokales Spiel wählen und ruft anschliessend die entsprechenden Funktionen aus dem Modul Game auf.

CRC Card des Moduls Wormmain

Modulname: Wormmain		
main	Hauptprogramm, lässt den Anwender eine Spielversion auswählen und startet das entsprechende Spiel	Benötigt die Module: Error Game

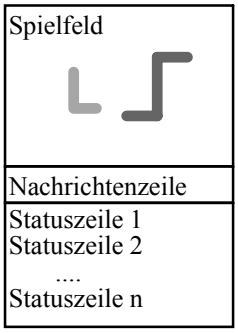
4.3.2 Das Modul Playfield "playfield.h"

Das Modul Playfield stellt Funktionen zur Verwaltung des Spielfeldes zur Verfügung. Es verwaltet das Spielfeld, eine Nachrichtenzeile und mehrere Statuszeilen.

Das Spielfeld ist als Array organisiert. Der Zugriff auf das Array erfolgt mit Get() und Set() Funktionen. Die Get() Funktion liefert den Inhalt der entsprechenden Zelle (0xFFFFFFFF für Zellen ausserhalb des definierten Spielfeldes), die Set() Funktion setzt die gewählte Zelle auf den entsprechenden Wert, und stellt auf dem Bildschirm gleich die entsprechende Figur dar, d.h. das Änderungen im Array automatisch auch auf dem Bildschirm ersichtlich werden.

Im Array werden die Farbwerte, sowie die Form des entsprechenden Punktes abgelegt. Bit 0-23 enthalten den RBG Wert, und mit Bit 28 wird zwischen Futter und Wurm unterschieden

Prinzipieller Aufbau der Anzeige des Spielfeldes:



Die Anzeige ist in drei Bereiche aufgeteilt: In das eigentliche Spielfeld, in eine Nachrichtenzeile, in welcher Systeminformationen dargestellt werden, und einer Menge von Statuszeilen. Für jeden Spieler ist eine eigene Statuszeile reserviert.

CRC Card des Moduls PlayField

Modulname: Playfield		
InitPlayfield	Initialisiert das Spielfeld	Benötigt die Module: Error Window
SetField	Setzt ein Feld auf einen bestimmten Wert und führt die Änderung auf dem Bildschirm nach	
GetField	Liefert den Inhalt des Feldes zurück (Oder 0xFFFFFFFF falls ausserhalb)	
Clearfield	Löscht das Spielfeld	
ShowMessage	Zeigt eine Nachricht in der Nachrichtenzeile an	
ShowStatus	Zeigt eine Statusmeldung in der angegebenen Statuszeile an	

4.3.3 Das Modul Dispatcher "Dispatcher.h"

Das Modul Dispatcher wird im Netzwerkbetrieb dem Modul Playfield vorangestellt. Es leitet alle Playfieldfunktionen an Netzwerkclienten und an das eigene Spielfeld weiter (Serverfunktionalität). Wenn also die Playfieldfunktionen dieses Moduls verwendet werden, ist das Spiel automatisch Netzwerkfähig.

Zusätzlich stellt das Modul Funktionen zur Auswertung und Behandlung von Telegrammen zur Verfügung. Sämtliche Spielfeldbezogene Telegramme werden ausgeführt, resp. die entsprechenden Playfieldfunktionen aufgerufen (Clientbetrieb). Zudem können Tastendrücke gesendet (Client) und Empfangen (Server) werden.

Im Serverbetrieb können beim Dispatcher bis zu 30 Klienten angemeldet werden.

CRC Card des Moduls Dispatcher

Modulname: Dispatcher		
InitDispatcher	Initialisiert den Dispatcher	Benötigt die Module: Error Playfield Communication
DInitPlayfield	Initialisiert das Spielfeld	
DSetField	Setzt ein Feld auf einen bestimmten Wert und führt die Änderung auf dem Bildschirm nach	
DGetField	Liefert den Inhalt des Feldes zurück	
DClearfield	Löscht das Spielfeld	
DShowMessage	Zeigt eine Nachricht in der Nachrichtenzeile an	
DShowStatus	Zeigt eine Statusmeldung in der angegebenen Statuszeile an	
HandlePlayfieldTelegram	Wertet ein Telegramm vom Server aus und ruft die entsprechenden Funktionen des Playfieldes auf. (Client Betrieb)	
SendKeyStroke	Sendet einen Tastendruck an eine bestimmte Netzwerk–verbindung.	
ReceiveKeyStroke	Empfängt einen Tastendruck von einer beliebigen Netzwerkverbindung (Wartet nicht, liefert Tastendruck und dazugehörige Verbindung zurück).	
AddClientToDispatcher	Fügt einen neuen Clienten in die Verteilerliste hinzu. Playfieldbefehle werden von nun an auch an diese Verbindung gesendet	

4.3.4 Das Modul Futter "Futter.h"

Das Modul Futter stellt Funktionen zur Verwaltung des Futters zur Verfügung. Es setzt eine vordefinierte Anzahl von Futterpunkten zufällig ins Spielfeld und verschiebt die ältesten Punkte an eine neue Position. Gefressene Punkt kommen im gleichen Rhythmus auch wieder ins Spiel. Futterpunkte werden nur auf Felder gesetzt, deren Nachbarn leer sind. Futterfelder haben automatisch ein gesetztes Bit 28 im Feld, unabhängig des in Color übergebenen Wertes. Das Modul benötigt die Funktionen SetField() um das Futter zeichnen zu können und GetField() um auf belegte Felder und ungefressenes Futter testen zu können

CRC Card des Moduls Futter

Modulname: Futter		
InitializeFood	Initialisiert das Futtermodul und legt die Anzahl Futterpunkte Fest. Alloziert die benötigten Ressourcen.	Benötigt die Module: Error Playfield oder Dispatcher
MoveFood	Bewegt die angegeben Anzahl von Futterstücken an einen neuen Platz, gefressene werden neu gesetzt. Es werden die am längsten nicht mehr bewegten neu gesetzt.	
FreeFood	Gibt alle Ressourcen wieder frei	

4.3.5 Das Modul Wurm "Wurm.h"

Das Modul Wurm stellt Funktionen zur Verwaltung eines Wurms zur Verfügung. Es verwaltet Würmer, deren Länge und Bewegung, und prüft auf Kollisionen. Für jeden Wurm wird automatisch eine Statusanzeige nachgeführt, welche den Namen des Spielers und die aktuelle Länge des Wurms umfasst. Das Modul benötigt die Funktionen SetField() um die Würmer zeichnen zu können, GetField() um auf Kollisionen testen zu können und ShowStatus() um Informationen zu den Würmern anzeigen zu können. Im Moment werden Informationen für maximal 10 Würmer angezeigt, es können aber mehr Würmer verwaltet werden.

CreateWorm erzeugt einen Wurm und liefert einen Verweis darauf zurück. Allen anderen Funktionen muss dieser Verweis übergeben werden, damit sie auf den entsprechenden Wurm einwirken.

CRC Card des Moduls Wurm

Modulname: Wurm		
CreateWorm	Erzeugt einen neuen Wurm und alloziert benötigte Ressourcen. Zeichnet den Wurm und seine Initialstatuszeile auf das Spielfeld	Benötigt die Module: Error Playfield oder Dispatcher
MoveWorm	Bewegt den Wurm in die angegebene Richtung, prüft auf Kollisionen und Futter, lässt den Wurm bei Bedarf länger werden. Führt die Änderungen auf dem Spielfeld nach	
HighlightWorm	Zeichnet den ganzen Wurm neu.	
DeleteWorm	Löscht den Wurm und gibt alle Ressourcen frei.	
SetWormGrowth	Setzt die Wachstumsrate für den Wurm.	

Vordefinierte Datentypen:

WormData — Handle (Zeiger) auf eine Wurmstruktur, in welcher alle Informationen des entsprechenden Wurms abgelegt sind. Der Anwender hat keinen Zugriff auf den Inhalt dieser Struktur. Für jeden Wurm muss eine solche Struktur mittels CreateWorm() erzeugt werden.

4.3.6 Das Modul Player "Player.h"

Das Modul Player stellt Funktionen zur Verwaltung eines Spielers zur Verfügung. Es verwaltet die Spieler, deren Namen, Punktstände, Steuerereignisse, Netzwerkverbindungen und den Wurm des jeweiligen Spielers.

CreatePlayer erzeugt einen Spieler und liefert einen Verweis darauf zurück. Allen anderen Funktionen muss dieser Verweis übergeben werden, damit sie auf den entsprechenden Spieler einwirken.

CRC Card des Moduls Player

Modulname: Player		
CreatePlayer	Erzeugt einen Spieler und alloziert die benötigten Ressourcen.	Benötigt die Module: Error Wurm Communication Window
PlacePlayer	Plaziert den Wurm des Spielers im Spielfeld (Und erzeugt ihn)	
HandleKeyEvent	Übergibt dem Spieler einen Tastendruck zur Behandlung. Falls die Taste nicht für diesen Spieler ist, wird sie ignoriert, sonst konsumiert.	
MakeMove	Bewegt den Wurm des Spielers in die letztgewählte Richtung.	
DeletePlayer	Löscht den Spieler (und seinen Wurm) und gibt alle Ressourcen wieder frei.	
IsPlayerAvaillable	Prüft ob der Spieler noch vorhanden ist (Seine Netzwerkverbindung noch OK ist).	

Vordefinierte Datentypen:

PlayerData — Handle (Zeiger) auf eine Spielerstruktur, in welcher alle Informationen des entsprechenden Spielers abgelegt sind. Der Anwender hat keinen Zugriff auf den Inhalt dieser Struktur. Für jeden Spieler muss eine solche Struktur mittels CreatePlayer() erzeugt werden.

4.3.7 Das Modul Game "Game.h"

Das Modul Game stellt Funktionen zur Durchführung eines resp. mehrerer Spiele zur Verfügung.

CRC Card des Moduls Game

Modulname: Game		
GameServer	Führt ein Spiel im Servermodus durch. Fragt die Namen und Anzahl der lokalen Spieler, sowie die Anzahl der Netzwerkspieler ab, wartet auf Clienten und spielt anschliessend ein Spiel im Netzwerkbetrieb. Erzeugt Futter und Spieler und löscht sie nach dem Spiel wieder	Benötigt die Module: Error Playfield oder Dispatcher Player Communication Futter
LocalGame	Führt ein lokales Spiel durch. Fragt zu Beginn nach Anzahl und Namen der lokalen Spieler. Erzeugt Futter und Spieler und löscht sie nach dem Spiel wieder	
GameClient	Startet ein Spiel als Client. Tastendrücke werden an den Server weitergeleitet, und Spielfeldtelegramme vom Dispatcher an das lokale Spielfeld weitergeleitet.	

4.4 Detaillierte Modulbeschreibungen

4.4.1 Das Modul Playfield "playfield.h"

Das Modul Kapselt das Spielfeldarray vom restlichen Programm ab, und stellt Funktionen zum Setzen und Lesen der Arrayelemente zur Verfügung. Beim Setzen wird zugleich das Graphikfenster entsprechend aktualisiert.
Das Spielfeld ist ein quadratisches Array vom Typ int, mit der Grösse MAX_FIELD_SIZE * MAX_FIELD_SIZE. Beim Initialisieren des Spielfeldes kann die aktuelle Spielfeldgrösse angegeben werden, darf aber in keiner Richtung grösser als MAX_FIELD_SIZE sein.

Modulschnittstelle

Vordefinierte Konstanten:

MAX_FIELD_SIZE Die grösste unterstützte Spielfeldgrösse (Länge oder Breite).

Funktionen

Das Modul Playfield stellt folgende Funktionen zur Verfügung:

void InitPlayfield(unsigned int Width, unsigned int Height)
Initialisiert das Spielfeld und legt seine Grösse fest (Width * Height). Width und Height müssen kleiner als MAX_FIELD_SIZE sein. Es werden alle Felder auf 0 gesetzt. Muss Aufgerufen werden, bevor irgendeine der anderen Funktionen dieses Moduls verwendet werden darf. Falls Width oder Height zu gross sind, wird das Programm mit einem Fatalen Fehler abgebrochen.

void SetField(unsigned int x, unsigned int y, int Color)
Setzt das Feld mit der Koordinate (x,y) auf die mit Color definierte Farbe und Form. Die Farbe wird durch Bit 0 bis 23 definiert: Bit 16-23 = Red, Bit 8-15 = Green und Bit 0-7 = Blue. Die Form wird durch die Bits 31-28 Definiert: 0000 = Quadrat, 0001 = Kreis (Futter), der Wert 1111 (f) ist reserviert für Fehlermeldung bei GetField(), alles andere ist undefiniert. Bits 24 bis 27 werden nicht benutzt und sind frei verfügbar. Falls x oder y zu gross sind, wird das Programm mit einem Fatalen Fehler abgebrochen.

int GetField(unsigned int x, unsigned int y)
Liefert die Farbe und die Form des Feldes mit der Koordinate (x,y). Es wird derselbe Wert zurückgeliefert, der zuvor mit SetField() geschrieben wurde. Falls x oder y nicht innerhalb des Spielfeldes liegen (Rand oder weiter entfernt), wird 0xffffffff zurückgeliefert.

void ClearField(void)
Initialisiert das Spielfeld neu. Es werden alle Felder auf 0 gesetzt und das Spielfeld neu gezeichnet.

void ShowMessage(char *Message)
Zeigt die Nachricht Message in der Nachrichtenzeile an. Eine bereits existierende Nachricht wird überschrieben.

void ShowStatus(char *Status, unsigned int Line, int Color)
Zeigt die Statusmeldung Status in der Zeile Line mit der Farbe Color an. Die Farbe wird durch Bit 0 bis 23 definiert: Bit 16-23 = Red, Bit 8-15 = Green und Bit 0-7 = Blue. Eine bereits in dieser Zeile existierende Statusmeldung wird überschrieben.

4.4.2 Das Modul Wurm "Wurm.h"

Das Modul Wurm stellt Funktionen zur Verwaltung eines Wurms zur Verfügung. Es verwaltet Würmer, deren Länge, Bewegung und prüft auf Kollisionen. Für jeden Wurm wird automatisch eine Statusanzeige nachgeführt, welche den Namen des Spielers und die aktuelle Länge des Wurms umfasst. Das Modul benötigt vom Modul Playfield (Oder Dispatcher) die Funktionen SetField() um die Würmer zeichnen zu können, GetField() um auf Kollisionen Testen zu können und ShowStatus() um Informationen zu den Würmern anzeigen zu können. Im Moment werden Informationen für maximal 10 Würmer angezeigt, es können aber mehr Würmer verwaltet werden.

Zur Verwaltung der Würmer werden folgende Datenstrukturen verwendet

Das Array StatusLines verwaltet die Statuszeilen, im Moment werden maximal 10 Statuszeilen unterstützt (Makro MAX_STATUS_LINE). Freie Statuszeilen werden mit einer 0 markiert, belegte mit einer 1.

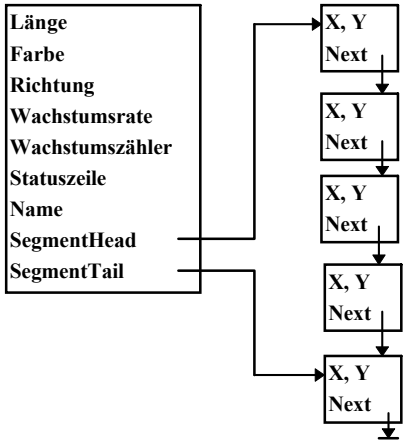
Die Struktur Wormsegment enthält die Informationen für ein einzelnes Wurmelement und einen Zeiger auf ein nächstes Wurmelement, so kann eine Liste der Wurmelemente erstellt werden.

```
typedef struct WormSegment {
   unsigned int x;                  X-Position des Elementes
   unsigned int y;                  Y-Position des Elementes
   struct WormSegment *Next;        Zeiger auf das nächste Element
} WormSegment;
```

Die Struktur WormType enthält die Informationen für einen Wurm. Die Struktur wird mit CreateWorm erzeugt, und mit DeleteWorm wieder gelöscht.

```
typedef struct WormType {
   int Length;                      Aktuelle Länge des Wurms
   int Color;                       Farbe des Wurms, wird bei Kollision 'gedimmt'
   int LastDir;                     Richtung der letzten Bewegung, -1 = Wurm ist tot.
   int GrowthRate;                  Nach wieviel Schritten der Wurm auch ohne Futter wächst
   int GrowthCounter;               Verwaltet das Wachstum (Schrittzähler)
   int StatusLine;                  Nummer der Statuszeile für diesen Wurm
   char *Name;                      Name des Spielers
   WormSegment *SegmentsHead;       Start der Wurmsegmentliste
   WormSegment *SegmentsTail;       Ende der Wurmsegemntliste
} WormType;
```

Ein Wurm bildlich dargestellt:



Modulschnittstelle

Vordefinierte Datentypen:

WormData — Handle (Zeiger) auf eine Wurmstruktur, in welcher alle Informationen des entsprechenden Wurms abgelegt sind. Der Anwender hat keinen Zugriff auf den Inhalt dieser Struktur. Für jeden Wurm muss eine solche Struktur mittels CreateWorm() erzeugt werden.

Funktionen

Das Modul Wurm stellt folgende Funktionen zur Verfügung:

WormData CreateWorm(unsigned int x, unsigned int y, unsigned int Growth, unsigned int Dir, int Color, char *Name)

Erzeugt einen neuen Wurm und liefert das Handle (Einen Zeiger) darauf zurück. Das Handle muss in einer Variable vom Typ WormData abgespeichert werden. Die Startposition des Wurms wird in x und y übergeben, die Startrichtung in Dir (Nach oben: 0, rechts: 1, unten: 2, links: 3). In Color wird die Farbe des Wurms übergeben. Die Farbe wird durch Bit 0 bis 23 definiert: Bit 16-23 = Red, Bit 8-15 = Green und Bit 0-7 = Blue. In Name wird der Name des zum Wurm gehörenden Spielers übergeben und in Growth wird die Wachstumsgeschwindigkeit des Wurms festgelegt. (Nach wieviel Bewegungsschritten der Wurm um ein Segment länger wird).

int MoveWorm(WormData Worm, int Dir)

Bewegt den angegebenen Wurm um einen Schritt in die gewählte Richtung. (Dir = -1: nach links, Dir = 0: Geradeaus, Dir = 1: nach rechts). Liefert die Länge des Wurms zurück, oder einen negativen Wert (-Länge) wenn der Wurm mit einem Hindernis kollidiert ist. Die Routine prüft jedes mal auf Kollisionen, verlängert den Wurm wenn er auf Futter gestossen ist, oder Growth Schritte seit der letzten Verlängerung vergangen sind. Die Anzeigen werden entsprechend angepasst. (Futter wird am gesetzten Bit 28 im Feld erkannt, ein leeres Feld am Wert 0, alles andere gilt als Kollision)

void HighlightWorm(WormData Worm)

Zeichnet den angegebenen Wurm neu aufs Spielfeld.

void DeleteWorm(WormData Worm)

Löscht den angegebenen Wurm und gibt all seine Ressourcen wieder frei. Nach Aufruf dieser Funktion darf das Handle dieses Wurms nicht mehr verwendet werden.

void SetWormGrowth(WormData Worm, unsigned int Growth)

Setzt eine neue Wachstumsrate für den angegebenen Wurm.

4.4.3 Das Modul Futter "Futter.h"

Das Modul Futter stellt Funktionen zur Verwaltung des Futters zur Verfügung. Es setzt eine vordefinierte Anzahl von Futterpunkten zufällig ins Spielfeld und verschiebt die ältesten Punkte an eine neue Position. Gefressene Punkt kommen im gleichen Rhythmus auch wieder ins Spiel. Futterpunkte werden nur auf Felder gesetzt, deren Nachbarn leer sind. Futterfelder haben automatisch ein gesetztes Bit 28 im Feld, unabhängig des in Color übergebenen Wertes. Das Modul benötigt die Funktionen SetField() um das Futter zeichnen zu können und GetField() um auf belegte Felder und ungefressenes Futter testen zu können

Zur Verwaltung des Futters werden folgende Datenstrukturen verwendet:

Die Struktur Futterinfo enthält die Informationen für ein einzelnes Futterstück. Beim Aufruf von InitializeFood wird ein Array mit der gewünschten Menge an Futterstücken erzeugt, MoveFood versetzt jeweils die ältesten Stücke an einen neuen Platz und löscht sie vom alten Standort, wenn sie noch nicht gefressen wurden. Um immer die ältesten Stücke zu finden, wird das Array als Ringbuffer verwendet. (Zu einem Ring geschlossen, also das Ende logisch an den Anfang gefügt)

```
typedef struct FutterInfo {
   int x;
   int y;
} FutterInfo;
```

Das Modul definiert folgende lokalen Hilfsfunktionen:

Die Funktion PlaceFood() versetzt das angegebene Futterstück an einen neuen, zufälligen, aber freien Platz auf dem Spielfeld. Dabei wird darauf geachtet, dass auch die Nachbarfelder frei sind.

Modulschnittstelle

Vordefinierte Datentypen:

Keine

Funktionen

Das Modul Futter stellt folgende Funktionen zur Verfügung:

void InitializeFood(unsigned int Width, unsigned int Height, int MaximumFood, int FColor)

Initialisiert den Futterstreuer. Width und Height geben den Bereich an, innerhalb dessen das Futter verteilt wird. MaximumFood definiert die Anzahl der Maximal gleichzeitig vorhandenen Futterstücke, und Color definiert die Farbe der Futterstücke. Die Farbe wird durch Bit 0 bis 23 definiert: Bit 16-23 = Red, Bit 8-15 = Green und Bit 0-7 = Blue. Wenn Futterstücke ins Feld gesetzt werden, wird zur Farbe noch das Bit 28 gesetzt, um das Futter als solches zu kennzeichnen.

void MoveFood(unsigned int HowMany)

Plaziert die angegebene Anzahl der ältesten Futterpunkte neu. Wenn diese noch nicht gefressen wurden, werden sie von ihrer alten Position entfernt, bevor sie auf eine neue Position gesetzt werden.

void FreeFood(void)

Gibt alle vom Modul reservierten Ressourcen wieder frei.

4.4.4 Das Modul Dispatcher "Dispatcher.h"

Das Modul Dispatcher leitet alle Playfieldfunktionen an Netzwerkclienten und an das Spielfeld weiter (Serverfunktionalität). Wenn also die Playfieldfunktionen dieses Moduls verwendet werden, ist das Spiel automatisch Netzwerkfähig.

Zusätzlich stellt das Modul Funktionen zur Auswertung und Behandlung von Telegrammen zur Verfügung. Sämtliche Spielfeldbezogene Telegramme werden ausgeführt, resp. die entsprechenden Playfieldfunktionen aufgerufen (Clientbetrieb). Zudem können Tastendrücke gesendet (Client) und Empfangen (Server) werden.

Im Serverbetrieb können dem Dispatcher bis zu 30 Klienten angemeldet werden.

Zur Verwaltung der Clienten werden folgende Datenstrukturen verwendet

> In dem Array DispatchTable ist Platz für 30 Handles von Clienten vorgesehen, somit können bis zu 30 Clienten verwaltet werden. Neue Clienten müssen mit der Funktion AddClientToDispatcher() angemeldet werden und werden so in die Tabelle eingetragen. Sämtliche Playfieldfunktionen werden an das eigene Playfield weitergeleitet, aber es werden auch entsprechende Telegramme an alle angemeldeten Clienten weitergeschickt.
>
> Die Maximale Anzahl von Clienten ist mit dem Makro MAX_NUMBER_OF_DISPATCH_ENTRIES festgelegt.

Modulschnittstelle

Vordefinierte Datentypen:

MessagePtr Zeiger auf ein Datentelegramm.

Vordefinierte Konstanten:

MAX_FIELD_SIZE Die grösste unterstützte Spielfeldgrösse (Länge oder Breite).

Zudem sind folgende Ersetzungen definiert:

```
#define InitPlayfield DInitPlayfield
#define SetField      DSetField
#define GetField      DGetField
#define ClearField    DClearField
#define ShowMessage   DShowMessage
#define ShowStatus    DShowStatus
```

Damit können Module die bis anhin mit dem Modul Playfield gearbeitet haben ohne Änderungen mit dem Modul Dispatcher zusammenarbeiten. Die Module müssen einfach Dispatcher.h anstelle von Playfield.h mit include einbinden. (Sämtlichen Dispatcherfunktionen ist einfach ein D vorangestellt).

Funktionen

Das Modul Dispatcher stellt folgende Funktionen zur Verfügung:

void InitDispatcher(void)
Initialisiert das Dispatchermodul. Muss Aufgerufen werden bevor irgend eine andere Funktion dieses Moduls verwendet werden darf.

void DInitPlayfield(unsigned int Width, unsigned int Height)
Sendet das entsprechende InitPlayfield Kommando an alle angemeldeten Klienten und ruft InitPlayfield() vom Modul PlayField auf.

void DSetField(unsigned int x, unsigned int y, int Color)
Sendet das entsprechende SetField Kommando an alle angemeldeten Klienten und ruft SetField() vom Modul PlayField auf.

int DGetField(unsigned int x, unsigned int y)
Ruft GetField() vom Modul PlayField auf. (Es wird kein Telegramm verschickt !!)

void DClearField(void) */* Noch nicht Implementiert !!!! */*
Sendet das entsprechende ClearField Kommando an alle angemeldeten Klienten und ruft ClearField() vom Modul PlayField auf.

void DShowMessage(char *Message)
Sendet das entsprechende ShowMessage Kommando an alle angemeldeten Klienten und ruft ShowMessage() vom Modul PlayField auf.

void ShowStatus(char *Status, unsigned int Line, int Color)
Sendet das entsprechende ShowStatus Kommando an alle angemeldeten Klienten und ruft ShowStatus() vom Modul PlayField auf.

MessageType *HandlePlayfieldTelegram(MessageType *T)
Wertet das übergebene Telegramm aus, und ruft die entsprechenden Funktionen von Playfield auf. Falls das Telegramm keine Playfieldbefehle (CMD_INIT_FIELD, CMD_SET_FIELD, CMD_SHOW_MESSAGE, CMD_SHOW_STATUS) enthielt, wird nichts gemacht und ein Zeiger auf das unveränderte Telegramm zurückgeliefert, sonst wird die entsprechende Funktion aufgerufen, das Telegramm freigegeben und Null zurückgeliefert.

void SendKeyStroke(int Handle, int Key)
Sendet den übergebenen Tastendruck an den Host mit dem Handle Handle.

int ReceiveKeyStroke(int *Handle)
Versucht ein Telegramm mit Tastencode zu empfangen. Wenn kein Telegramm empfangen wurde, kehrt die Funktion sofort zurück. Telegramme, die keinen Tastencode enthalten werden ignoriert und gelöscht. Die Funktion liefert in Handle das Handle des Absenders zurück. Der Rückgabewert ist der empfangene Tastencode. Falls kein gültiges Tastencodetelegramm empfangen wurde, wird 0 zurückgeliefert. Falls irgend ein Telegramm empfangen wurde, steht in Handle das Handle des Absenders, sonst NULL.

void AddClientToDispatcher(int Handle)
Fügt den in Handle angegebenen Klienten in die Dispatchertabelle ein. Sämtliche Playfield Zeichenbefehle werden von nun an auch an diesen Klienten weitergeleitet. Es können maximal 30 Klienten in die Dispatchertabelle eingetragen werden.

4.4.5 Das Modul Player "Player.h"

Das Modul Player stellt Funktionen zur Verwaltung eines Spielers zur Verfügung. Es verwaltet die Spieler, deren Namen, Punktstände, Steuerereignisse und den Wurm des jeweiligen Spielers.

Zur Verwaltung der Spieler werden folgende Datenstrukturen verwendet

Die Struktur Playerinfo enthält die Informationen für einen Spieler. Die Struktur wird mit CreatePlayer erzeugt, und mit DeletePlayer wieder gelöscht.

```
typedef struct PlayerInfo {
   char *Name;              Name des Spielers
   int   Color;             Farbe des Spielers
   WormData Worm;           Zum Spieler gehörender Wurm (Handle)
   int   Handle;            Zum Spieler gehörende Netzwerkverbindung, negative Werte für
                            lokale Spieler
   int   NextDir;           Richtung für die nächste Wurmbewegung
} PlayerInfo;
```

Modulschnittstelle

Vordefinierte Datentypen:

PlayerData — Handle (Zeiger) auf eine Spielerstruktur, in welcher alle Informationen des entsprechenden Spielers abgelegt sind. Der Anwender hat keinen Zugriff auf den Inhalt dieser Struktur. Für jeden Spieler muss eine solche Struktur mittels CreatePlayer() erzeugt werden.

Funktionen

Das Modul Player stellt folgende Funktionen zur Verfügung:

PlayerData CreatePlayer(char *Name, int Handle, int Color)

Erzeugt einen neuen Spieler und liefert einen Verweis (Einen Zeiger) darauf zurück. Der Verweis muss in einer Variable vom Typ PlayerData abgespeichert werden. In Name wird der Name des Spielers übergeben und in Color wird die Farbe des Wurms übergeben. Die Farbe wird durch Bit 0 bis 23 definiert: Bit 16-23 = Red, Bit 8-15 = Green und Bit 0-7 = Blue. In Handle wird die Netzwerkverbindung des Spielers übergeben. Bei lokalen Spielern wird -1 für den ersten, und -2 für den zweiten Spieler angegeben. Positive Werte (= Gültige Handles) sind für Netzwerk-Spieler reserviert.
Achtung, es wird noch kein Wurm erzeugt, und auch noch nichts gezeichnet. Um den Spieler mit einem Wurm auszustatten muss PlacePlayer() aufgerufen werden.
Jeder so erzeugte Spieler muss vor Programmende mit DeletePlayer() wieder gelöscht werden.

void PlacePlayer(PlayerData Player, unsigned int x, unsigned int y, unsigned int Dir)

Stattet eine Spieler mit einem Wurm aus, und plaziert den Wurm an der gegebenen Koordinate (x, y)mit der gegebenen Startrichtung (Dir) ins Spielfeld.
In Player muss das Handle für den betreffenden Spieler übergeben werden.

void HandleKeyEvent(PlayerData Player, int Handle, int *KeyCode)

Behandelt Tastendrücke für den angegebenen Spieler. Falls das Handle oder der Tastendruck nicht für diesen Spieler ist, wird das Ereignis ignoriert und nichts verändert. Wenn das Ereignis für diesen Spieler ist, wird es entsprechend ausgewertet, und der Keycode gelöscht (Auf 0 gesetzt).

In Player wird das Handle für den aktuellen Spieler übergeben, in Handle die Netzwerkverbindung von welcher der Tastendruck stammt (Negative Werte in Handle stehen für lokale Tastendrücke). In Keycode wird ein Zeiger auf den Code der gedrückten Taste übergeben. Der Code wird auf 0 gesetzt, wenn das Ereignis für diesen Spieler bestimmt war (So kann das Ereignis der Reihe nach an alle Spieler übergeben werden, und derjenige, der es brauchen kann, löscht es, so dass es von keinem anderen mehr verwendet werden kann).

int MakeMove(PlayerData Player)

Bewegt den Wurm des angegebenen Spielers um einen Schritt in seine aktuelle Richtung. Diese Funktion muss periodisch für alle Spieler aufgerufen werden.

Die Funktion liefert die aktuelle Länge des Wurms vom Spieler zurück (Positive Werte wenn alles OK ist, negative Werte wenn der Wurm tot ist (Auf ein Hindernis gestossen ist)).

void DeletePlayer(PlayerData Player)

Diese Funktion löscht einen Spieler und gibt alle seine Ressourcen wieder frei. Sie muss vor Programmende für jeden mit CreatePlayer() erzeugten Spieler aufgerufen werden. Die zum Spieler gehörende Netzwerkverbindung wird ebenfalls abgebrochen.

int IsPlayerAvaillable(PlayerData Player)

Kontrolliert, ob der entsprechende Spieler noch verfügbar ist (Ob die Netzwerkverbindung zum Spieler noch steht).

Liefert True (1) zurück, wenn die Verbindung OK ist, sonst False (0).

4.4.6 Das Modul Game "Game.h"

Das Modul Game stellt Funktionen zur Durchführung eines resp. mehrerer Spiele zur Verfügung.

Zur Verwaltung der Spiele werden folgende Datenstrukturen verwendet

Die Anzahl Schritte, nach denen das Futter neu plaziert, resp. der Level erhöht wird, sind mit folgenden Konstanten festgelegt:

```
#define LEVEL_STEP 150
#define FOOD_STEP  10
```

Die Farben der ersten 12 Spieler sind in der Tabelle PlayerColorTable festgelegt

Im Array Players sind die am aktuellen Spiel beteiligten Spieler abgelegt

Das Modul definiert folgende lokalen Hilfsfunktionen:

Die Funktion AcceptNetworkConnections() akzeptiert die angegebene Anzahl von Netzwerkspielern (Clienten), wenn sich mehr Spieler anmelden werden sie als passive Beobachter aufgenommen. Die Funktion erzeugt die nötigen Spielerstrukturen und legt sie im Array Players ab. Sobald Return gedrückt wird, kehrt die Funktion zurück.

Die Funktion DoAGame() führt ein Spiel durch, unabhängig ob Lokal oder über das Netzwerk. Zuerst initialisiert sie das Spielfeld, plaziert die Spieler gleichmässig im Feld und initialisiert das Futter. Anschliessend nimmt sie Tastendrücke von allen Quellen entgegen und leitet Sie an alle Spieler weiter (Die Spieler entscheiden selbst, ob der Tastendruck für sie bestimmt war). Nach Ablauf des Spielzugzeitintervalls werden alle Würmer um einen Schritt weiterbewegt und es wird wieder auf Tastendrücke gewartet. Nach Ablauf der entsprechenden Zeitintervalle wird das Futter neu verteilt, oder der Schwierigkeitsgrad erhöht. Das Spiel dauert solange bis kein Wurm mehr lebt oder Escape gedrückt wird.

Die Funktion AskForLocalPlayers() akzeptiert die angegebene Anzahl von lokalen Spielern, erfragt deren Namen, erzeugt die nötigen Spielerstrukturen und legt sie im Array Players ab.

Modulschnittstelle

Vordefinierte Datentypen:

Keine

Funktionen

Das Modul Game stellt folgende Funktionen zur Verfügung:

void GameServer(void)

Spielt ein Spiel als Server. Wenn ein Spiel zu Ende ist, wird je nach Wunsch gleich wieder ein neues gestartet.

void LocalGame(void)

Führt eine lokale Spielrunde durch, wenn ein Spiel zu Ende ist, wird je nach Wunsch gleich wieder ein neues gestartet.

void GameClient(void)

Startet ein Spiel im Client-Mode. Die Funktion handelt als einfaches Terminal, d.h. Tastendrücke werden an den Server weitergeleitet, und Zeichenbefehle (Playfield Befehle) von Server entgegengenommen und ausgeführt.

4.4.7 Das Modul Wormmain "-"

Das Modul enthält das Hauptprogramm (main()) für das gesamte Programm. Es lässt den Benutzer zwischen den Varianten Server, Client oder lokales Spiel wählen und ruft anschliessend die entsprechenden Funktionen aus dem Modul Game auf.

Vordefinierte Datentypen:

Keine

Funktionen

Das Modul Game stellt folgende Funktionen zur Verfügung:

void main(int argc, char *argv[])

Lässt den Spieler in einem Menu eine Spielvariante auswählen und startet diese.

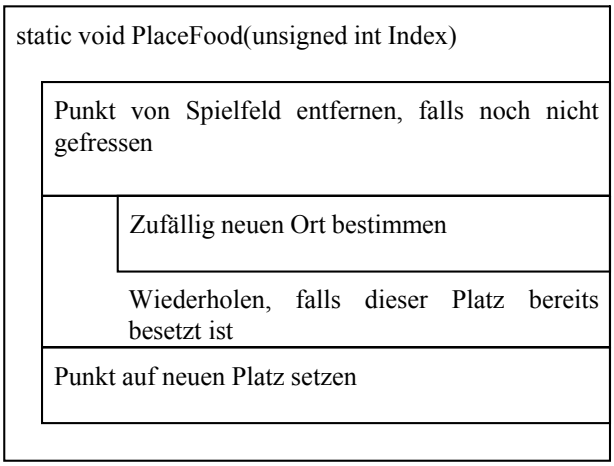
4.5 Struktogramme

In den folgenden Kapiteln sind die Struktogramme der komplexeren Funktionen des Projekts aufgeführt.

4.5.1 Modul Futter

4.5.1.1 static void PlaceFood(unsigned int Index)

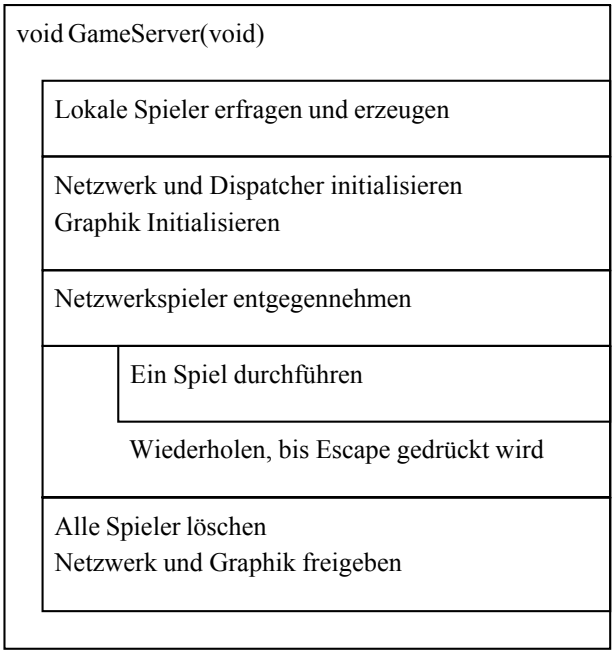
Setzt einen Futterpunkt zufällig auf einen freien Platz.



4.5.2 Modul Game

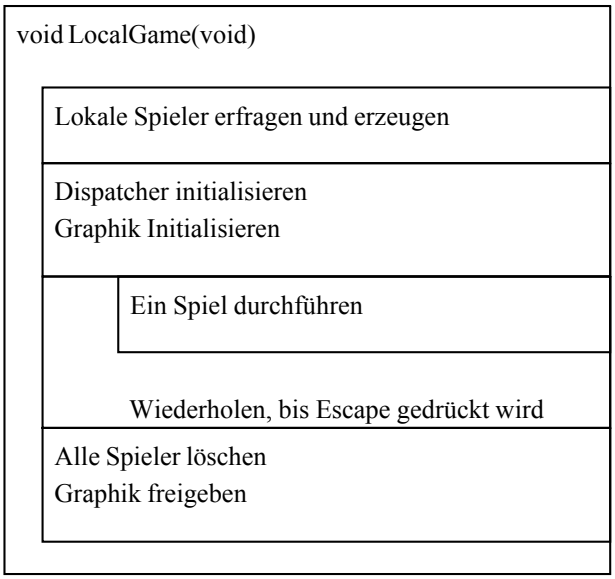
4.5.2.1 void GameServer(void)

Bereitet ein Spiel für den Serverbetrieb vor. Das effektive Spiel wird in einer anderen Funktion (DoAGame()) durchgeführt.



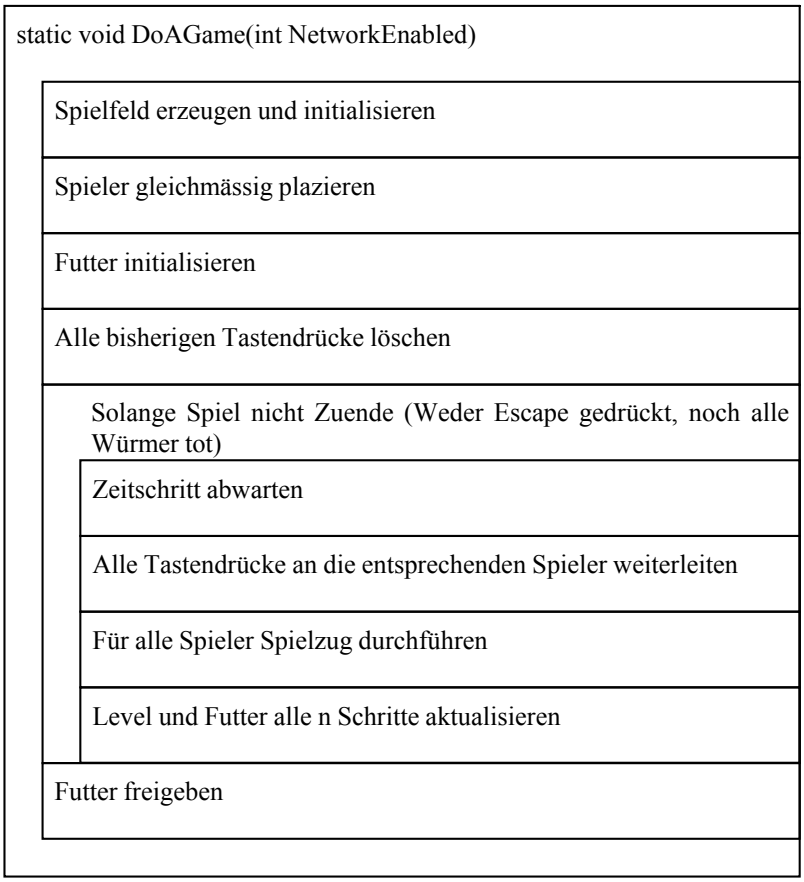
4.5.2.2 void LocalGame(void)

Bereitet ein Spiel für den Lokalbetrieb vor. Das effektive Spiel wird in einer anderen Funktion (DoAGame()) durchgeführt.



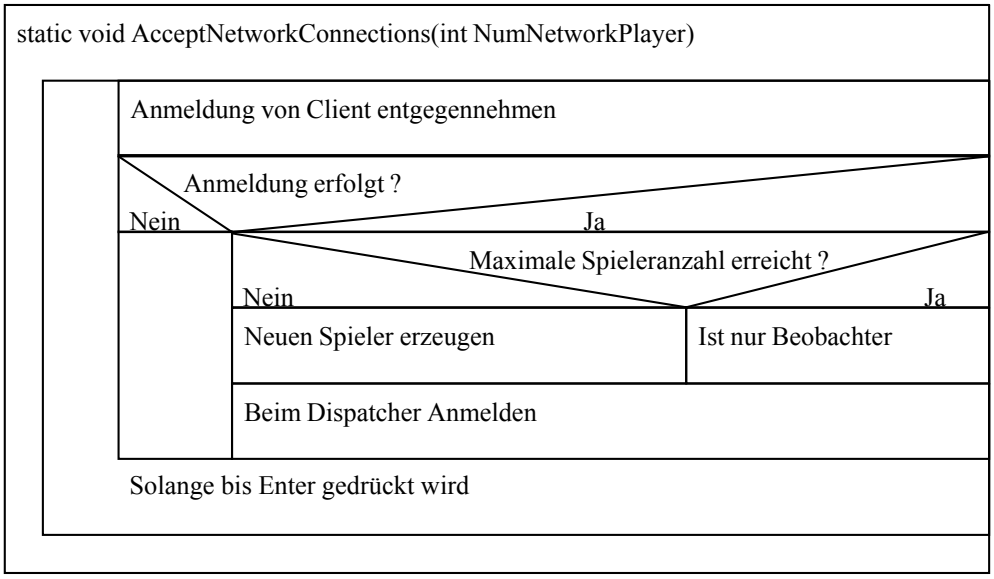
4.5.2.3 static void DoAGame(int NetworkEnabled)

Führt ein Spiel durch, unabhängig ob Netzwerk oder lokal.



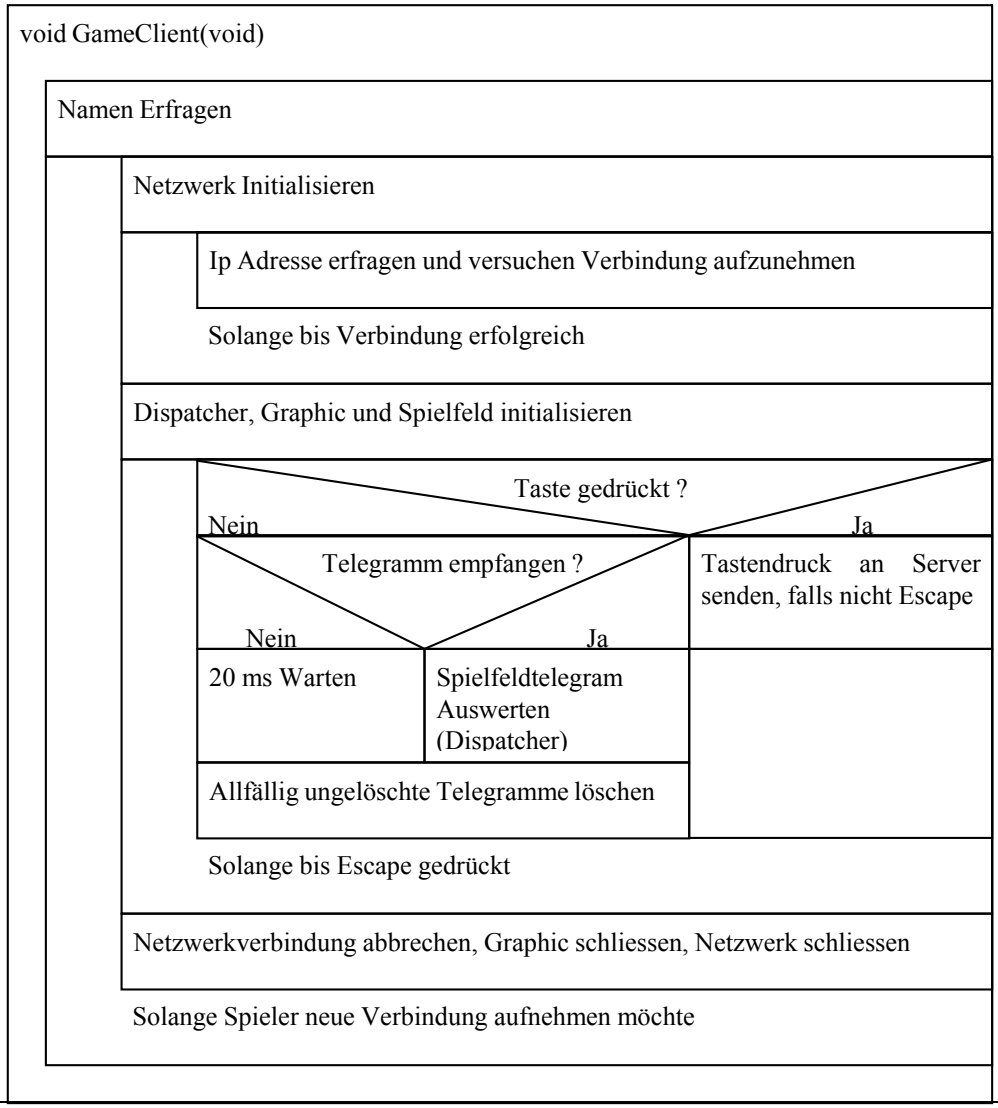
4.5.2.4 static void AcceptNetworkConnections(int NumNetworkPlayer)

Hilfsfunktion für den Serverbetrieb, nimmt Anmeldungen von Clienten entgegen und initialisiert die benötigten Strukturen für jeden neuen Clienten.



4.5.2.5 void GameClient(void)

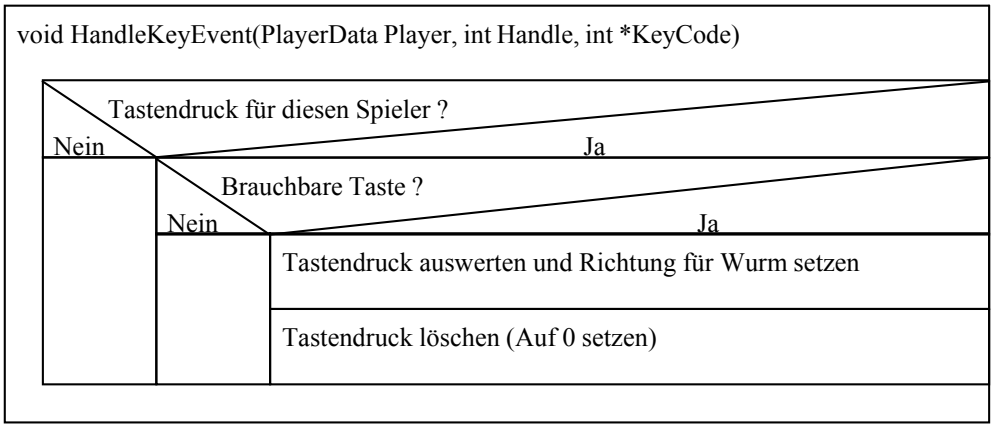
Führt ein Spiel im Clientenmodus durch, sendet Tastendrücke an den Server, und nimmt Spielfeldtelegramme entgegen und wertet sie aus.



4.5.3 Modul Player

4.5.3.1 void HandleKeyEvent(PlayerData Player, int Handle, int *KeyCode)

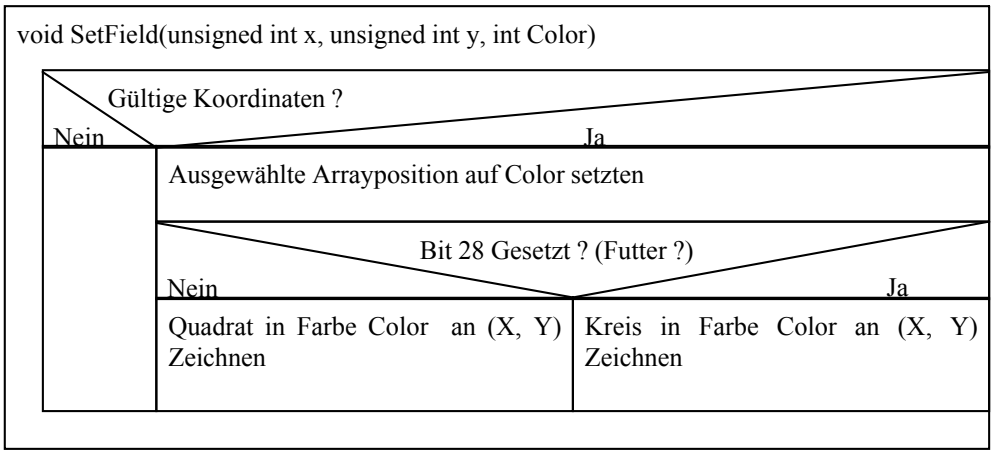
Wertet Tastendrücke für einen Spieler aus. Ignoriert nicht für diesen Spieler bestimmte Tastendrücke.



4.5.4 Modul Playfield

4.5.4.1 void SetField(unsigned int x, unsigned int y, int Color)

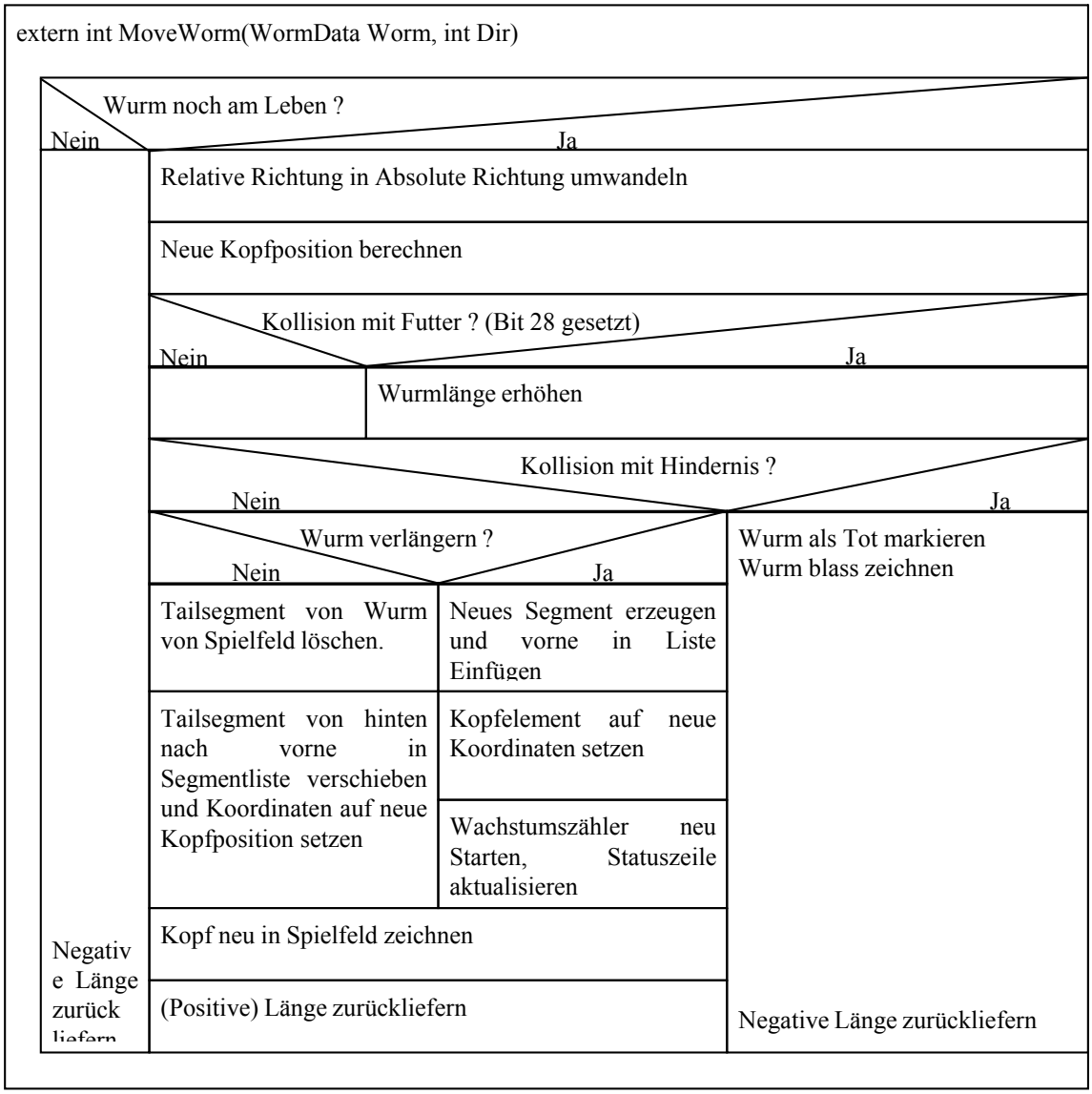
Setzt ein Feld im Spielfeld auf den gegebenen Wert, und führt die Darstellung auf dem Bildschirm nach.



4.5.5 Modul Wurm

4.5.5.1 int MoveWorm(WormData Worm, int Dir)

Bewegt den Wurm in die zuletzt gewählte Richtung. Prüft auf Kollisionen und Futter, ändert die Wurmlänge bei Bedarf und führt das Spielfeld und die Statuszeile für diesen Wurm nach.



5 Implementation

Die Implementation erfolgte gemäss dem Design. Es wurden keine weiteren Module hinzugefügt, auf den Einsatz globaler Variablen konnte vollständig verzichtet werden.

Die Implementation besteht aus den folgenden Dateien:

Dateiname	Funktion
communication.c communication.h	Zur Verfügung gestelltes Modul für die Netzwerkkommunikation
dbgmmgt.c dbgmmgt.h	Zur Verfügung gestelltes Modul zur Fehlersuche in der Speicherverwaltung (Zum Debuggen, sonst nicht benötigt)
dispatcher.c dispatcher.h	Verteilen der Zeichenbefehle an die Klienten (Serverseitig), Entgegennahme und Ausführen von Zeichenbefehlen (Clientseitig)
error.c error.h	Zur Verfügung gestelltes Modul für die Fehlerbehandlung von Runtimefehlern (Zum Debuggen)
futter.c futter.h	Verwaltung der Futterpunkte
game.c game.h	Verwaltung eines Spieles (Spielablauf)
Networking.c networking.h	Zur Verfügung gestelltes Modul für die Netzwerkkommunikation
player.c player.h	Verwaltung eines Spielers
playfield.c playfield.h	Verwaltung des Spielfeldes
winbgim.cpp winbgim.h	Zur Verfügung gestelltes Modul für die Graphikausgabe
window.c window.h	Zur Verfügung gestelltes Modul für die Graphikausgabe
wormmain.c	Hauptprogramm, Entscheidung zwischen lokalem und Netzwerkspiel
wurm.c wurm.h	Verwaltet einen Wurm (Bewegen, Wachstum und Kollisionsdetektion) und zeichnet ihn.

Grau hinterlegte Dateien wurden zur Verfügung gestellt, und sind in diesem Dokument nicht weiter beschrieben.

Weitergehende Dokumentation zur Implementation kann dem Kommentar in den jeweiligen Dateien entnommen werden

6 Tests

Folgende Testfälle wurden getestet:

Testfall	Test	Testdatum	Ergebnis
1	Lokalspiel, 1 Spieler, Steuerung	28.4.2004	OK
2	Lokalspiel, 1 Spieler, Punkte Fressen	28.4.2004	OK
3	Lokalspiel, 1 Spieler, Kollision mit allen Rändern	28.4.2004	OK
4	Lokalspiel, 1 Spieler, Kollision mit sich selbst (Auch Rückwärts in sich selbst)	28.4.2004	OK
5	Lokalspiel, 1 Spieler, Endet Spiel nach Kollision	28.4.2004	OK
6	Lokalspiel, 1 Spieler, Bleibt Anzahl Fresspunkte konstant	28.4.2004	OK
7	Lokalspiel, 1 Spieler, Wird Wurm länger nach 10 Schritten	28.4.2004	OK
8	Lokalspiel, 1 Spieler, Steigt Schwierigkeit (Level) mit der Zeit	28.4.2004	OK
9	Lokalspiel, 1 Spieler, Spiel vorzeitig mit ESC Abbrechbar	28.4.2004	OK
10	Lokalspiel, 1 Spieler, Zu langen oder leeren Namen eingeben	28.4.2004	OK
11	Lokalspiel, 1 Spieler, Namen mit Sonderzeichen eingeben	28.4.2004	OK
12	Lokalspiel, 2 Spieler, Test 1 bis 11 wiederholen	28.4.2004	OK
13	Lokalspiel, 2 Spieler, Test Kollision mit Gegnerwurm	28.4.2004	OK
14	Lokalspiel, 2 Spieler, Läuft Spiel weiter wenn erster Wurm tot	28.4.2004	OK
15	Lokalspiel, 2 Spieler, Endet Spiel wenn beide Würmer tot sind	28.4.2004	OK
16	Highscore anzeigen	28.4.2004	Nicht implementiert
17	Highscore Speichern	28.4.2004	Nicht implementiert
18	Netzwerkspiel, 1 Lokalspieler, 1 Netzwerkspieler Test 1-17 wiederholen	28.4.2004	OK
19	Abmelden des Netzwerkspielers mitten im Spiel	28.4.2004	OK
20	Abmelden des Lokalspielers mitten im Spiel	28.4.2004	OK
21	Mehrere Spieler mit selbem Namen anmelden	28.4.2004	OK
22	Eingabe falscher IP bei Client (Illegale IP, falsches Adressformat, nicht vorhandene IP, IP von Rechner ohne Gameserver)	28.4.2004	OK
23	Server starten, ohne Anmeldungen (Timeout oder Abbruch durch Benutzer möglich?)	28.4.2004	OK
24	Mehr Clienten anmelden als Spieler möglich	28.4.2004	OK
25	Werden überzählige Clienten als Beobachter geführt	28.4.2004	OK
26	Test 18-21 wiederholen mit Maximal möglicher Anzahl von Spielern	28.4.2004	OK
27	Programm 2 mal auf selbem Rechner Starten (Im Netzwerkbetrieb)	28.4.2004	OK (bricht ab)
28	Kann das Programm nach Beendigung ein zweites mal auf dem selben Rechner gestartet werden	28.4.2004	OK
29	Highscoredatei schreibgeschützt / nicht vorhanden	28.4.2004	Nicht implementiert
30	Netzwerkspiel bei PC ohne Netzwerkanschluss (Client und Server)	28.4.2004	OK
31	Sind die Farben der verschiedenen Spieler und des Futters gut unterscheidbar	28.4.2004	OK

Alle Tests wurden bestanden, die Highscore Tests wurden nicht durchgeführt, da dieses Feature noch nicht implementiert ist.

Qualitätskontrolle: Es wurden einige Module stichprobenweise auf Einhaltung der Programmierrichtlinien geprüft und für korrekt befunden.