Prof. Dr. Franz Schweiggert   Institut für Angewandte Informationsverarbeitung   23. November 2009
Michael Mattes / Wolfgang Kaifler   Projektaufgabe

WiMa–Praktikum I / Programmierpraktikum im Wintersemester 2009/2010

Abgabetermin: Meilenstein 1 bis 08.12.2009, Meilenstein 2 bis 22.12.2009

Organisatorisches

Die Projektaufgabe ist in freier Zeiteinteilung zu machen und besteht aus zwei Meilensteinen, welche zu unterschiedlichen Zeitpunkten abgegeben werden. Die Abgabe jedes Meilensteins geschieht zu einem je Gruppe individuell ausgemachten Termin von etwa 30 Minuten Dauer. Wenden Sie sich zur Abgabe mit 2-3 Terminvorschlägen an `michael.mattes@uni-ulm.de`.

Wenn Sie nur das WiMa-Praktikum abschließen wollen, ist es ab sofort nicht mehr nötig, die Übungen in Systemnahe Software zu machen. Falls Sie jedoch die Vorlesung Systemnahe Software I weiterhin besuchen wollen und die Prüfungsklausur für 3 Leistungspunkte mitschreiben, müssen Sie die Übungen ab Blatt 11 (erscheint am 21.12.2009, Abgabe bis 19.01.2009) wieder machen. Die Blätter 7 bis 10 werden nicht gewertet. *Es ist natürlich trotzdem dringend empfohlen, sich mit dem Stoff dieser Blätter zu beschäftigen!*

Die Gruppeneinteilung der Übungsblätter ist für die Projektaufgabe nicht zwingend gleich, d.h. Sie können auch neue Gruppen bilden. *Die Gruppengröße für das Projekt kann bis zu 6 Personen betragen. Größere Gruppen sind jedoch nicht erlaubt.* Beachten Sie, dass jedes Gruppenmitglied einen Teil des Programmcodes selbst schreiben muss. Reines Verständnis von fremden Code reicht hier NICHT aus!

Aufgabenstellung

Im Projekt sollen Sie verschiedene rundenbasierte Brettspiele am PC umsetzen. Die Ein- und Ausgaberoutinen (also z.B. das Spielbrett ausgeben, den Spieler zur Eingabe auffordern, den Zug einlesen) nennt man Engine. Die Engine soll von jedem Spiel gemeinschaftlich genutzt werden – die einzelnen Spiele machen also keine eigenen Ein- und Ausgaben.

Jedes Spiel soll als eigenständiges Modul realisiert werden. Damit die Integration mit der Engine klappt, muss das Spiel aber eine gemeinsame Schnittstelle (sozusagen eine allgemeine Headerdatei `game.h`) implementieren.

Meilenstein 1: Datenhaltung, Ein-/Ausgabe, Spiele-Interface

Im ersten Teil des Projekts geht es um die Grundlagen der Engine. Entwerfen Sie zunächst die nötigen Datenstrukturen zur Speicherung des Spielbretts, des Zustands einer Partie und eines einzelnen Spielzuges (passende Namen wären `struct board`, `struct gamestate` und `struct move`). Denken Sie dabei an eine sinnvolle Beschränkung der maximalen Spieleranzahl bzw. Brettgröße – beispielsweise mit zwei Makros MAXPLAYERS und MAXBOARDSIZE. Das Spielbrett selbst kann z.B. als zweidimensionales `char`-Array modelliert werden, der Zustand des Spiels muss mindestens den aktiven Spieler und die Punktzahlen aller Spieler speichern können – wobei die Punktzahlen eben nicht von jedem Spiel genutzt werden. Ein Spielzug kann z.B. als Array von `int`s gespeichert werden.

Wenn die Datenstrukturen vorhanden sind, implementieren Sie eine Funktion zur Ausgabe des aktuellen Spielbretts einschließlich des Spielzustands und eine weitere Funktion zum Einlesen eines Zuges. Funktionen zur Manipulation des Spielbretts sind ebenfalls nötig. Achten Sie auf eine schöne Formatierung der Ausgabe!

Als letzten Teil dieses Meilensteins sollen Sie sich einen sinnvollen Satz von Funktionen überlegen, die jedes Spiel implementieren muss. Jedes Spiel muss beispielsweise mindestens eine Funktion bereitstellen, die prüft, ob ein eingelesener Zug gültig ist. Ebenso werden Funktionen zur Durchführung eines Zuges und zur Abfrage, ob ein Spieler gewonnen hat, benötigt. All diese Funktionen zusammen werden *Interface* (Schnittstelle) des Spiels genannt.

Folgendes ist in diesem Meilenstein noch NICHT nötig: Die tatschliche Implementierung eines Spiels sowie die `main`-Funktion der Engine, die das eigentliche Spiel durchführt. Testprogramme mitsamt `main`-Funktion werden Sie jedoch brauchen.

Bevor Sie mit dieser Aufgabe anfangen, sollten Sie sich das Beispiel im zweiten Meilenstein anschauen.

Meilenstein 2: main-Funktion, Spiele-Implementierung

Im zweiten Meilenstein haben Sie die Aufgabe, die Engine mit einer passenden `main`-Funktion fertigzustellen und pro Gruppenmitglied ein Spiel zu implementieren. Folgende Spiele wären dabei beispielsweise möglich: Tic-Tac-Toe als einfachen Einstieg, Vier Gewinnt, Dame, Schach, Go oder auch das klassische Computerspiel Snake. Diese Auflistung ist aber nur als Vorschlag zu sehen, Sie können auch gern andere Spiele implementieren.

Zur Verdeutlichung des Modularisierungskonzepts folgt nun ein kleines Beispiel.

Seien `engine.c` und `engine.h` die Dateien der Engine, `game.h` die Headerdatei des Interfaces und `dame.c` sowie `4gewinnt.c` zwei Spiel-Implementierungen. Dann enthält `engine.h` die Definitionen der Verbünde `struct board` etc. sowie die Deklaration von Funktionen wie `set_field(int x, int y, char value)`. Die Datei `game.h` enthält unter anderem die Deklaration von `int is_valid_move(move* m)`. Die C-Dateien der Spiele binden `engine.h` ein und enthalten beide ihre eigene Implementierung der Funktionen aus `game.h`. Die Datei `engine.c` enthält natürlich alle Funktionsimplementierungen von `engine.h` und zusätzlich die `main`-Funktion.

Am Ende sollen die Spiele wie folgt kompilierbar sein:

```
# Erzeuge engine.o
$ gcc -Wall -std=c99 -c engine.c
```

```
# Erzeuge dame.o
$ gcc -Wall -std=c99 -c dame.c
# Erzeuge dame.exe
$ gcc -o dame.exe engine.o dame.o

# Erzeuge 4gewinnt.o
$ gcc -Wall -std=c99 -c 4gewinnt.c
# Erzeuge 4gewinnt.exe
$ gcc -o 4gewinnt.exe engine.o 4gewinnt.o

# Was passiert, well man versucht, beide Spiele
# gemeinsam zu linken?
$ gcc -o alles.exe engine.o dame.o 4gewinnt.o
4gewinnt.o: In function ‘is_valid_move’:
4gewinnt.c:(.text+0x0): multiple definition of ‘is_valid_move’
dame.o:dame.c:(.text+0x0): first defined here
[...]
collect2: ld returned 1 exit status
$
```

Viel Erfolg!