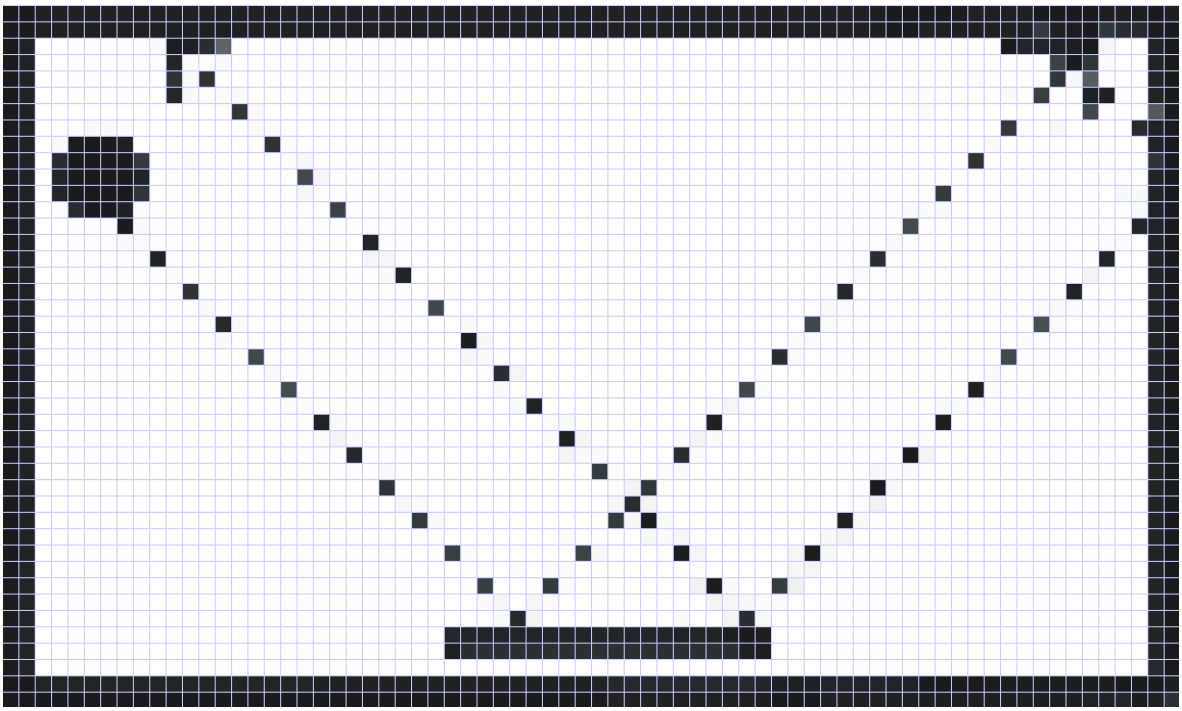

Projekt SuperBall

Inhalt

Modulverantwortlicher:	Alexander Huwaldt
Fertigstellungsdatum:	15.03.2010
Erstellt von:	Anja Raschdorf
Titel	Projektarbeit Superball
Beschreibung:	Entwurf und Realisierung objektorientierter Anwendungen auf einem Mikrocontrollersystem
Fachhochschule	Berufsakademie Sachsen, Staatliche Studienakademie Bautzen, Studienrichtung Wirtschaftsinformatik

Promenadenring 8
02708 Löbau
Deutschland

www.myAVR.de
support@myavr.de

Tel: ++49 (0) 358 470 222
Fax: ++49 (0) 358 470 233

1. Einleitung

Die Laser & Co. Solutions GmbH mit Sitz in Löbau entwickelt im Auftrag Ihrer Kunden Hard- und Software.
Aus unserem Alltag sind Mikrocontroller nicht mehr weg zu denken. Sie begleiten uns überall. In Handys, Autos, MP3-Player, USB-Sticks und natürlich Computern.

Die Praxisphase vom 20. Juli bis 27. September 2009 widmete sich der Entwicklung einer objektorientierten Anwendung auf einer Mikrocontrollerlösung.
Die Aufgabe verfolgt folgende Ziele:

- Erlernen der objektorientierten Softwareentwicklung
- Erfahrungen in der Programmierung von Mikrocontrollern sammeln.

Diese Arbeit wird die Herangehensweise an diese Thematik erläutern. Gleichzeitig legt Sie die erforderlichen Informationen zur Erreichung der oben genannten Ziele.

2 Theoretische Grundlagen

In diesem Abschnitt werden die erforderlichen Hintergrundinformationen zur Bearbeitung der Aufgabe gelegt.

2.1 UML - Unifed Modeling Language

Bei der Unifed Modeling Language (UML) handelt es sich um eine einheitliche Sprache und Notation in der objektorientierten Softwareentwicklung. Ihr Regelwerk umfasst dreizehn Diagramme zur Spezifikation, Konstruktion, Visualisierung und Dokumentation für Modelle der Softwareentwicklung. Zu den wichtigsten Diagrammen gehören unter anderem das Klassendiagramm und das Sequenzdiagramm. Im Klassendiagramm werden die verwendeten Klassen und deren Beziehungen zueinander grafisch dargestellt, sowie Schnittstellen und parametrisierte Klassen.
Das Sequenzdiagramm beschreibt das dynamische Verhalten und die Interaktionen zwischen den Objekten.

2.2 Objektorientierung

Die Objektorientierung betrachtet die Objekte, die in der realen Welt tatsächlich vorkommen, wie zum Beispiel Telefon, Fahrrad oder Auto. Die Objektorientierung sieht die Daten und Funktionen simultan an und diese werden bei der Modellierung zu Objekten zusammengefasst.
Die Grundbegriffe der Objektorientierung werden in Tabelle 1 kurz erläutert.

Tabelle 1 - Begriffe OOP

Begriffe	Erläuterung
Objekt	• Gegenstand oder Person in der Realität oder Anschauung • Instanz eines Bauplans oder einer Klasse • z. B.: Auto, Gegenstand der Realität
Klassen	• Bauplan gleichartiger Objekte • mit den Klassen werden die Objekte erst erzeugt • bestehend aus Attributen (Eigenschaften, z. B.: Farbe, Modell, Typ, Baujahr) und Methoden (Zustand repräsentieren) • z. B.: TAuto (Bauplan von Objekt Auto)
Methoden	• sind „Dienstleistungen“, diese können von einem Objekt angefordert werden • Implementationen von Operationen (meist Sequenzen von Anweisungen) • UML: Operationen als Implementierung • Praxis: irrelevant, ob man Methode oder Operation sagt • z. B.: gas geben oder bremsen
Nachrichten	• Objekte kommunizieren über Nachrichten • Aufruf einer Methode eines Objektes durch ein andres Objekt • Objekt ändert daraufhin selbst seine Eigenschaften
Beziehungen	• Assoziation, dies ist die loseste Beziehung, dabei kennt eine Klasse die Andere • Komposition, ist die strengste Beziehung, eine Klasse kann ohne die Andere nicht existieren • Aggregation, dabei kann jede Teilklasse extra existieren, aber die gesamte Klasse benötigt die Teilklasse • Dependency, verdeutlicht, dass eine Änderung eines unabhängigen Elements eine Änderung eines abhängigen Elements nach sich zieht
Sichtbarkeit	• „private“ (Zeichen im Klassendiagramm „-“), die Attribute und Methoden sind nur für die Klasse sichtbar, die sie besitzt. Für alle anderen bleiben sie vollständig verborgen. Damit sind die vor Angriffen von außen geschützt. • „protected“ (Zeichen im Klassendiagramm „#“), die Attribute und Methoden einer Klasse sind für die besitzende und deren abgeleiteten Klassen sichtbar. • „public“ (Zeichen im Klassendiagramm „+“), Attribute und Methoden sind für alle Klassen sichtbar. • package (Zeichen im Klassendiagramm „~“), Ansammlung von Modellelementen, die von verschiedenen Typen sein Können
Datenkapselung	• Attribute und Methoden einer Klasse werden zusammengefasst, durch Einschränkungen der Sichtbarkeiten „private“ auf die Eigenschaften • Eigenschaften der Objekte sind geschützt andere Teile von Anwendungen können auf diese Attribute nicht zugreifen • außerdem wird die Änderbarkeit wird erhöht • die Komplexität der Anwendung nimmt ab • der gesamte Quellcode wird sicherer
Polymorphie	• Vielgestaltigkeit • Operationen verhalten sich in unterschiedlichen Objekte einer Klasse unterschiedlich • Arten: • statischen Polymorphie: gleichnamigen Operationen, die sich nur in ihrer Signatur unterscheiden • dynamische Polymorphie: Fähigkeit einer Variable Objekteunterschiedlichen Typs zu zuordnen, dabei beschreibt der Typ eine Schnittstelle, Voraussetzung: späte Bindung der genaue Speicherort wird erst dann ermittelt, wenn der Aufruf • erfolgt, das heißt, erst wenn eine Nachricht an das Objekt gesendet wird
Vererbung	• Attribute und Methoden an abgeleitete Klassen weitergeben • Arten: Einfach- (eine Klasse kann von maximal einer anderen Klasse abgeleitet werden) und Mehrfachvererbungen (eine Klasse kann von mehr als einer anderen Klasse abgeleitet werden) • Prinzipien: Generalisierung (Suche nach der Oberklasse, z.B.: TFahrzeug) und Spezialisierung (Suche nach Unterklassen, z.B.: TAuto, TFahrrad)

2.3 Mikrocontroller

Ein Mikrocontroller ist ein Prozessor. Der Unterschied zu PC-Prozessoren besteht darin, dass bei einem Mikrocontroller Speicher, Digital- und Analog- Ein und -Ausgänge etc. meist auf einem einzigen Chip integriert sind. Damit kommt die Mikrocontroller-Anwendung mit wenigen Bauteilen aus. Mikrocontroller werden als erstes an der Bit-Zahl des internen Datenbusses unterschieden: 4bit, 8bit, 16bit und 32bit. Diese Bit-Zahl kann man als die Länge der Daten interpretieren, die der Controller in einem Befehl verarbeiten kann. Die größte in 8 Bit (= 1 Byte) darstellbare Zahl ist die 255. Der verwendete AVR Atmega 2560 ist ein 8-Bit-Controller. Somit kann dieser z.B. in einem Additionsbefehl immer nur Zahlen kleiner gleich 255 verarbeiten. Zur Bearbeitung von größeren Zahlen werden dann jeweils mehrere Befehle hintereinander benötigt.

Ein Mikrocontroller braucht zum Betrieb, wie jeder andere Prozessor auch, einen Takt. Die maximale Taktfrequenz mit der ein Controller betrieben werden kann, reicht von 1 MHz bei alten Controllern bis hin zu über 100 MHz bei teuren 32-Bittern. Quelle: http://www.mikrocontroller.net/articles/Mikrocontroller). Diese Taktfrequenz sagt jedoch noch nichts über die tatsächliche Geschwindigkeit eines Prozessors aus. Der ATmega2560 arbeitet mit 16 MHz. Da seine Frequenz in voller Höhe genutzt wird, verarbeitet dieser Controller 16 Millionen Befehle pro Sekunde. Es gibt auf dem Markt aber auch Controller, bei denen nicht die technisch mögliche Frequenz genutzt wird. Diese verarbeiten dann weniger Befehle pro Sekunde. Die Programmierung dieser erfolgt meist in den Sprachen Assembler und C. Es gibt verschieden Größen der Mikrocontroller, diese werden in der Tabelle 2 mit einigen Beispielen der Hersteller dargelegt.

Tabelle 2 – Mikrocontrollergrößen

Größe	Hersteller	Beispiel Mikrocontroller oder Serie
8 Bit	Atmel Intel	AVR (ATmega2560) 8048, 8051
16 Bit	Infineon Texas Instruments	C16x MSP430
32 Bit	Futjisu NXP (früher: Philips)	FR50 LPC2xxx

Abbildung 1 zeigt den Aufbau des verwendeten Mikrocontrollers.

Dessen Bauteile und deren Funktion werden in Tabelle 3 erläutert

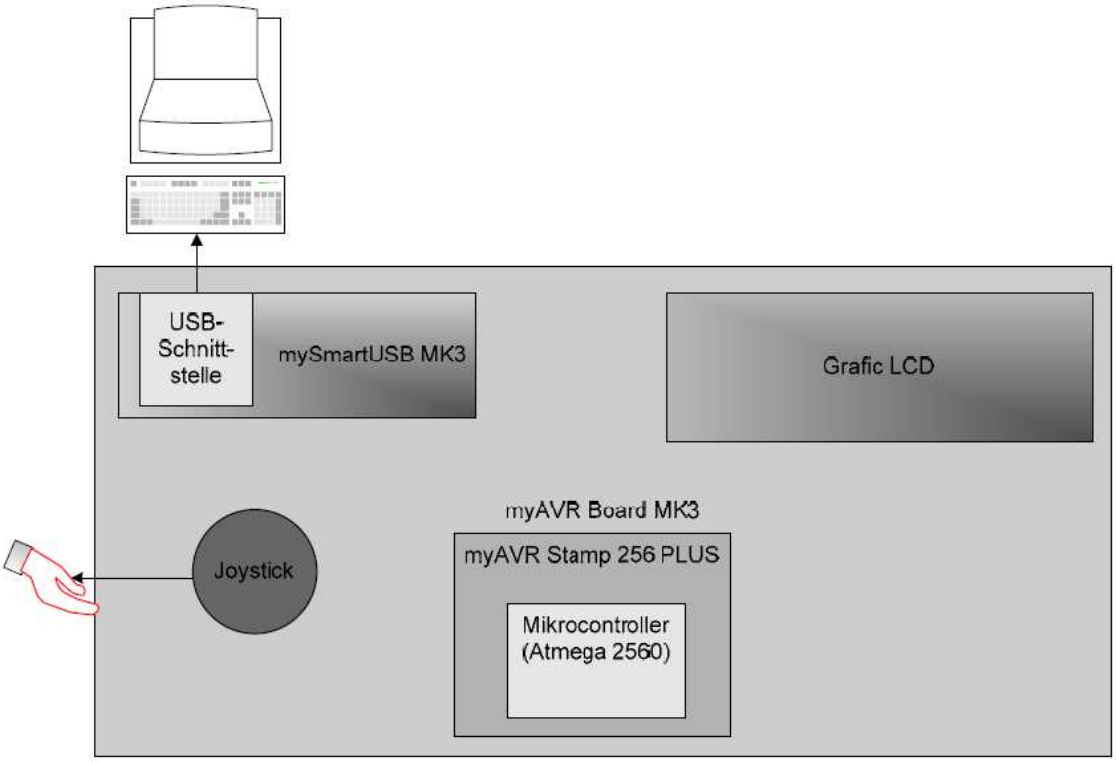

Abbildung 1 - Prinzipskizze vom myAVR Board MK3

Tabelle 3 - Bestandteile des myAVR Board MK3 und Funktionen

Bauelement	Funktion
mySmartUSB MK3	verbinden des PC's mit Board
myAVR Stamp256 PLUS	interner Programmspeicher (Chip) mit einer Kapazität von 256 KByte, beinhaltet (eingebrannt) Spielanleitung
Grafik LCD	Grafische Darstellung des Spieles

3. Analyse und Entwurf

Der Lösungsweg gliedert sich in die in Abbildung 2 dargestellten Phasen. Sie werden anhand des objekt-orientierten Vorgehens bearbeitet.

Quelle: S. Geisel: Methoden der Wirtschaftsinformatik – Systemanalyse Staatliche Studienakademie Bautzen 2009

Abbildung 2 – Projektphasen

3.1 Systemanalyse

Diese Analysephase ist gekennzeichnet durch:

vorliegende Informationen:	Objektorientierte Programmierung eines Ballspiels auf einem Mikrocontroller Kenntnisse über die Programmierung von Mikrocontrollern (Abschnitt 2.3)
Aufgaben zur Lösung:	Erstellen der Spielanleitung und Verarbeiten der darin enthaltenen Informationen Techniken: Nominal- und Verbalphrasenanalyse
Ergebnis:	Klassennamen und deren Beziehungen zueinander

Spielanleitung

Das Spiel Superball ist wie Abbildung 3 zeigt, aufgebaut. Der Ball bewegt sich ohne Steuerung des Spielers zick-zack-förmig ab einer definierten Startposition. Berührt er den linken, oberen und rechten Rand des Spielfeldes, wird er ebenfalls zick-zack-förmig abgestoßen. Am unteren Bildrand bewegt sich der Schläger in horizontaler Richtung. Mit diesem versucht der Spieler zu verhindern, dass der Ball das Spielfeld am unteren Ende verlässt. Der Ball bewegt sich mit konstanter Geschwindigkeit. Der Schläger dagegen bewegt sich schneller als der Ball. Bei jedem Zusammentreffen des Schlägers mit dem Ball werden die Punkte jeweils um eins erhöht. Bewegt sich der Ball durch die untere Begrenzung des Spielfeldes wird die Punktzahl auf dem LC-Display angezeigt und das Spiel wird beendet. Durch betätigen der Reset-Taste wird der Punktestand auf Null zurückgesetzt und das Spiel neu gestartet.

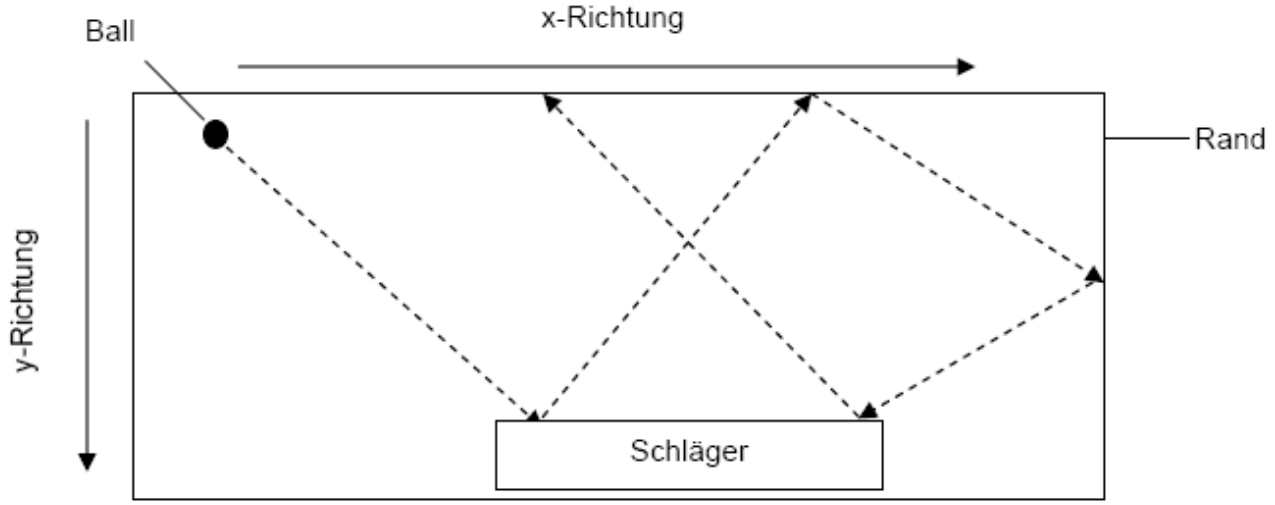

Abbildung 3 – Spielaufbau

Anhand der Spielbeschreibung werden mit Hilfe der **Nominalphasenanalyse** alle Substantive herausgefiltert. Redundanzen und Synonyme werden dabei nicht beachtet. Das Verfahren ist in Anlage 1 dargestellt.

Ball	Steuerung	Spieler	LC- Spielfeld
Startposition	Bildrand	Begrenzung	Display
Richtung	Schläger	Punktzahl	Reset-Taste
	Rand	Punkte	Null
	Zusammentreffen		Ende

Gemäß der Definition der Begriffe der Objektorientierung (Abschnitt 2.2) werden im Anschluss die Substantive herausgefiltert, die als potenzielle Klassenkandidaten in Frage kommen:

Controller	Ball
Schläger	Rand

Sie sind durch folgende Eigenschaften gekennzeichnet:

- Sie kommen in der Realität vor (z.B. Gegenstände, Personen)
- Mit Ihnen können konkrete Funktionen durchgeführt werden (z.B. Abfangen des Balls, der Ball bewegt sich).

Danach erfolgt die Verbalphrasenanalyse (Anlage 1), bei der anhand der Spielanleitung „Subjekt-Prädikat-Objekt“ Aussagen gesucht und daraus die Beziehungen zwischen den Klassen abgeleitet werden.

bewegen	berühren	abstoßen	verlassen
versuchen	verhindern	anzeigen	erhöhen
beenden	starten	betätigen	zurücksetzen

Als grafisches Darstellungsmittel werden Class Responsibility Collaboration (CRC) - Karten eingesetzt. Diese sind ein Darstellungsmittel zur Veranschaulichung der Klassenkandidaten, deren Eigenschaften und Verhalten (Attribute und Methoden), sowie ihre Beziehungen zu anderen Klassen (siehe Abbildung 4). Abbildung 5 zeigen die CRC-Karten des Spiels Superball.

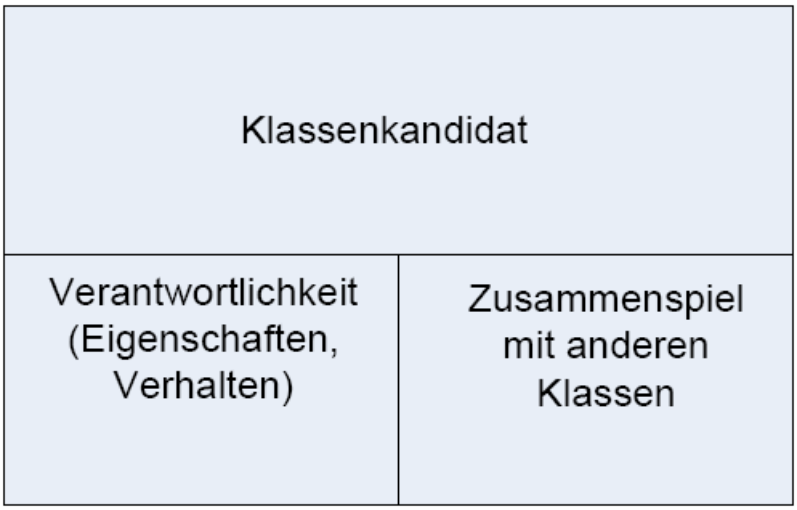

Abbildung 4 - Grundstruktur von CRC-Karten

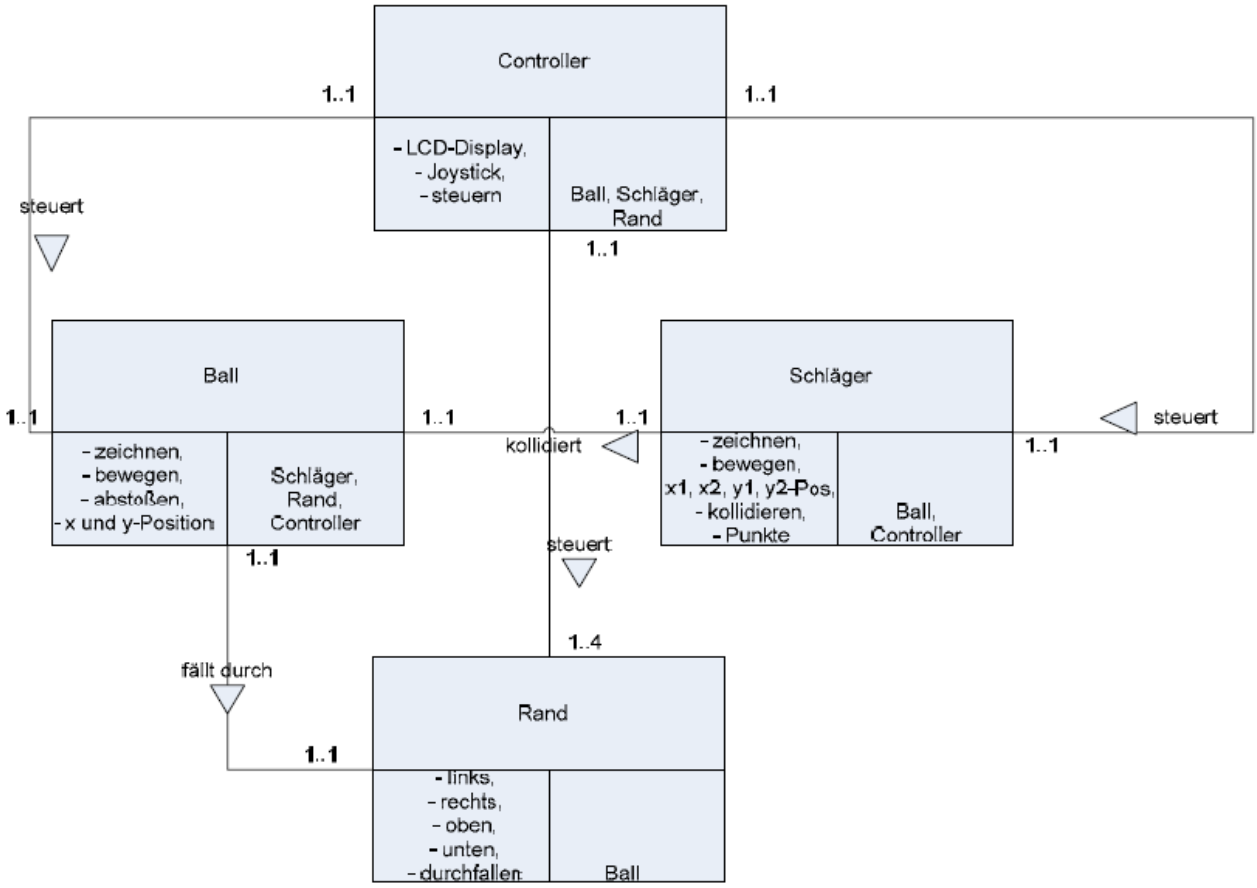

Abbildung 5 - CRC-Karten am Beispiel des Superballs

3.2 Entwurf der Systemstruktur

vorliegende Informationen:	Klassennamen und deren Beziehungen zueinander
Aufgaben zur Lösung:	Entwurf der softwaretechnischen Umsetzung
Ergebnis:	grobes Klassendiagramm mit Ausprägung der Klassennamen und Attributen

In der Entwurfsphase werden die Ergebnisse der Analysephase hinsichtlich aller Gegebenheiten verifiziert. Des Weiteren folgt hier der softwaretechnische Entwurf. Dazu zählen die Festlegung der Variablen und Datentypen, Konstruktion der Methodenaufrufe und Beziehungen zwischen den Klassen. Als Darstellungsmittel werden UML-Diagramme benutzt. Oben auf der CRC-Karte ist der Klassenname zusehen, auf der linken Seite sind alle Attribute und Methoden der Klasse verzeichnet und rechts stehen die Beziehungen zwischen den einzelnen Klassen. Folgende Abschnitte beschreiben die Überlegungen für die einzelnen Klassen.

Klasse Controller:

Die Klasse Controller steuert das gesamte Spiel auf dem Board. Für die Eigenschaften (Attribute) LCD-Display und Joystick müssen Variablen und Datentypen eingeführt werden:

Eigenschaft	Variable	Datentyp	Beschreibung
LCD-Display		graphicLCD	Datentyp aus Hilfedatei des
(Anzeige des Spiels)			myAVR Board MK3
Joystick	btnLeft	PushButton	Datentyp aus Hilfedatei des
	btnRight	PushButton	myAVR Board MK3

Der Spieler steuert den Schläger mit Hilfe des Joysticks. Der Joystick wird vom Spieler nach links bzw. nach rechts bewegt. Diese Unterscheidung muss durch zwei Variablen eingeführt werden: btnLeft und btnRight. Durch den für diese Variablen zugewiesenen Datentyp PushButton wird jeweils auch das Drücken des Joysticks gewährleistet.

Klasse Ball:

Die Eigenschaft Startposition der Klasse Ball ist charakterisiert durch die Anfangswerte x und y. Der Ball wird in diesem Spiel als Kreis gezeichnet. Daher ist die Variable radius einzuführen. Aufgrund der automatischen Bewegung des Balls in zick-zack-Richtung (ri). Diese wird durch die Variablen riX und riY charakterisiert.
Die Datentypen der Attribute sind Zahlen, deshalb wird als Datentyp Integer gewählt.

Eigenschaft	Variable	Datentyp	Beschreibung
Startposition	pos_x	Integer	ganze Zahl zwischen -2^{31} bis
	pos_y		
Bewegungsrichtung	riX und riY		
Radius	radius		

Klasse Schläger:

Da auch der Schläger in der reellen Welt existiert ist dies eine Klassen. Er wird wegen seiner rechteckigen Form durch die zwei Anfangs- und Endwerte spezifiziert. Da sich der Schläger schneller bewegt als der Ball wird die Variable step eingeführt. Diese Geschwindigkeitsänderung wird mit der Konstante 1,5 festgelegt. Jede neue Position des Schlägers errechnet sich aus der Bewegungsrichtung des Joysticks multipliziert mit der Geschwindigkeitsänderung step und das zusammen mit der Startposition des Schlägers addiert. Bei der Berechnung der erneuten Startposition des Schlägers muss die Bewegungsrichtung des Joysticks berücksichtigt werden:

Bewegung des Joysticks	Berechnung	
Links (-1)	step = -1 * step pos_x1 = pos_x1 * step pos_x2 = pos_x2 * step	pos_y1 = pos_y1 * step pos_y2 = pos_y2 * step
Rechts (1)	step = 1 * step pos_x1 = pos_x1 * step pos_x2 = pos_x2 * step	pos_y1 = pos_y1 * step pos_y2 = pos_y2 * step

Wegen der Inkonsistenz der Datentypen Integer und Float wird der Datentyp Float eingesetzt. Bei jeder Kollision des Balls mit dem Schläger wird jeweils ein Punkt erzielt. Diese werden in der Variable punkt aufgenommen und im internen Speicher des Controllers gehalten. Bei Erhöhung des Punktestands wird der Wert der Variablen punkt aus dem internen Speicher des Controllers gelesen, um eins erhöht und anschließend wieder zurück geschrieben. Der Datentyp volatile sichert dieses Vorgehen.

Eigenschaft	Variable	Datentyp	Beschreibung
Startposition Rechteck	pos_x1, pos_x2 pos_y1, pos_y2	float	Fließkommazahl, im Bereich von +/-
Geschwindigkeitsänderung	step		3.40282347*1038
Punktestand	punkte	volatile	

Klasse Rand:

Gemäß der CRC-Karte besitzt die Klasse Rand die Eigenschaften links, rechts, oben und unten. Die Grenzen des Spielfelds befinden sich an den sichtbaren Grenzen des LCD-Displays. Nachdem alle Klassen beschrieben wurden, ist das gesamte UML-Klassendiagramm in Abbildung 6 zu sehen.

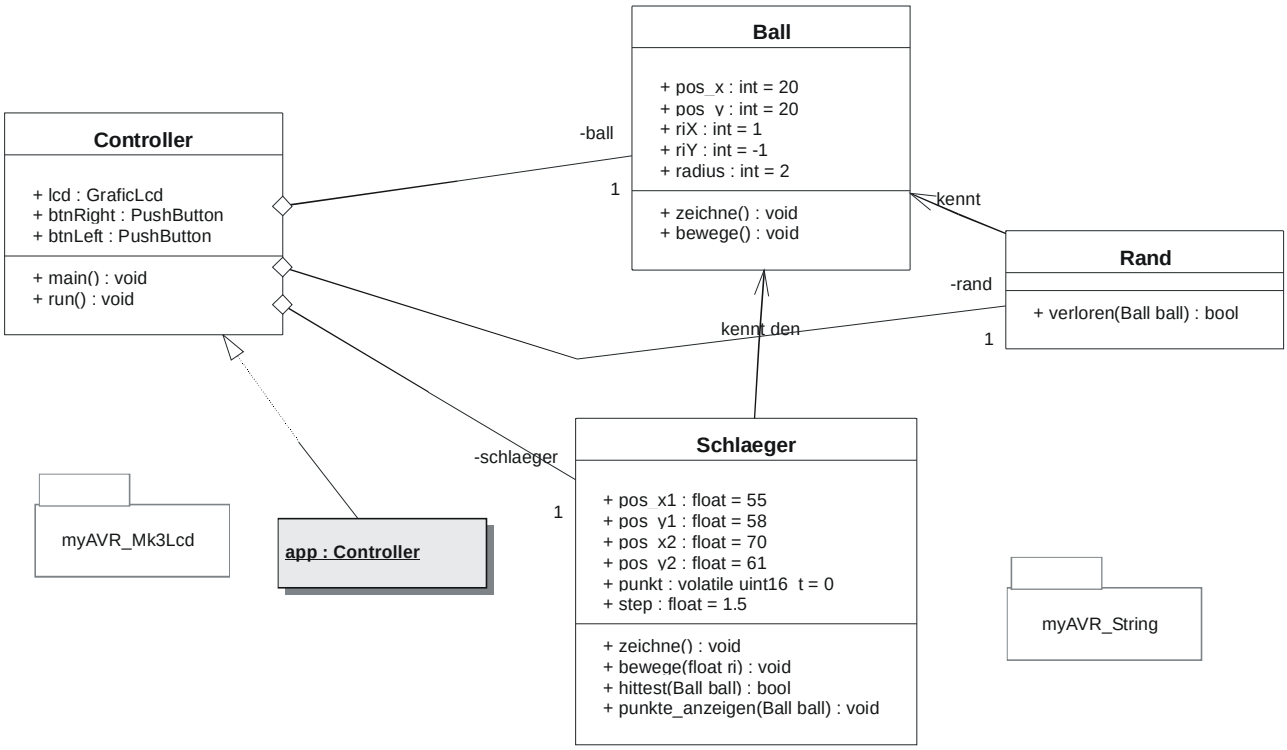

Abbildung 6 - Klassendiagramm

3.3 Entwurf des Systemverhaltens

Vorliegende Informationen:	Eigenschaften und Zustände (Attribute) der einzelnen Klassenkandidaten
Aufgaben zur Lösung:	Entwurf der softwaretechnischen Umsetzung
Ergebnis:	Verfeinertes Klassendiagramm als Voralge für Implementierung Sequenzdiagramm zur Darstellung der Änderung des Systemverhaltens

Die grafische Darstellung des Spiels auf dem LCD-Display übernimmt die auf dem Controller vordefinierte Bibliothek myAVR_MK3Lcd.

Klasse Controller:

Aus der Analysephase geht hervor, dass die Klasse Controller keine Verhaltensänderungen aufweist. Damit besitzt sie keine eigenen Methoden. Zur Sicherstellung der Funktionalität der Anwendung dienen die Methoden main()und run(), die die Software beim Anlegen der Klasse Controller bereitstellt. Beide Methoden besitzen hierbei keinen Rückgabetypen. Der dafür entsprechende Datentyp unter C++ ist void. Die Methode main() initialisiert die Variablen der Klasse. Die Methode run() steuert die Reaktion des Spiels auf Aktivität des Benutzers, wie beispielsweise das Bewegen des Joysticks. Des Weiteren werden in ihr die Methoden der abhängigen Klassen Ball, Schläger und Rand aufgerufen.

Klasse Ball:

Im Gegensatz zur Klasse Controller ändert der Ball während der Spiels sein Verhalten. Er muss gezeichnet werden und bewegt sich. Die Methode zeichne() zeichnet den Ball. Die Bewegung des Balls wird in der Methode bewege() implementiert mit den Eingabewerten der Bewegungsrichtung (Variablen riX und riY).

Klasse Schläger:

Ebenso wie der Ball muss der Schläger an jeder neuen Position während des Spielverlaufs gezeichnet werden. Dazu dient die Methode zeichne(). Die Änderung der Bewegungsrichtung des Schlägers wird in der Methode bewege() aufgenommen. Die entsprechende Bewegungsrichtung des Joysticks wird als Eingabeparameter in der Variable ri übergeben. Zur Prüfung einer Kollision zwischen Ball und Schläger wird die Methode hittest() eingeführt. Sie nimmt als Eingabeparameter die x- und y- Position des Balls auf. Der Rückgabewert kann nur wahr oder falsch sein (Datentyp bool, Abkürzung für Boolean). Die Abbildung 7 zeigt die Kollision zwischen Ball und Schläger in einem Sequenzdiagramm. Es stellt das dynamische Verhalten und die Interaktionen zwischen den Objekten grafisch dar.

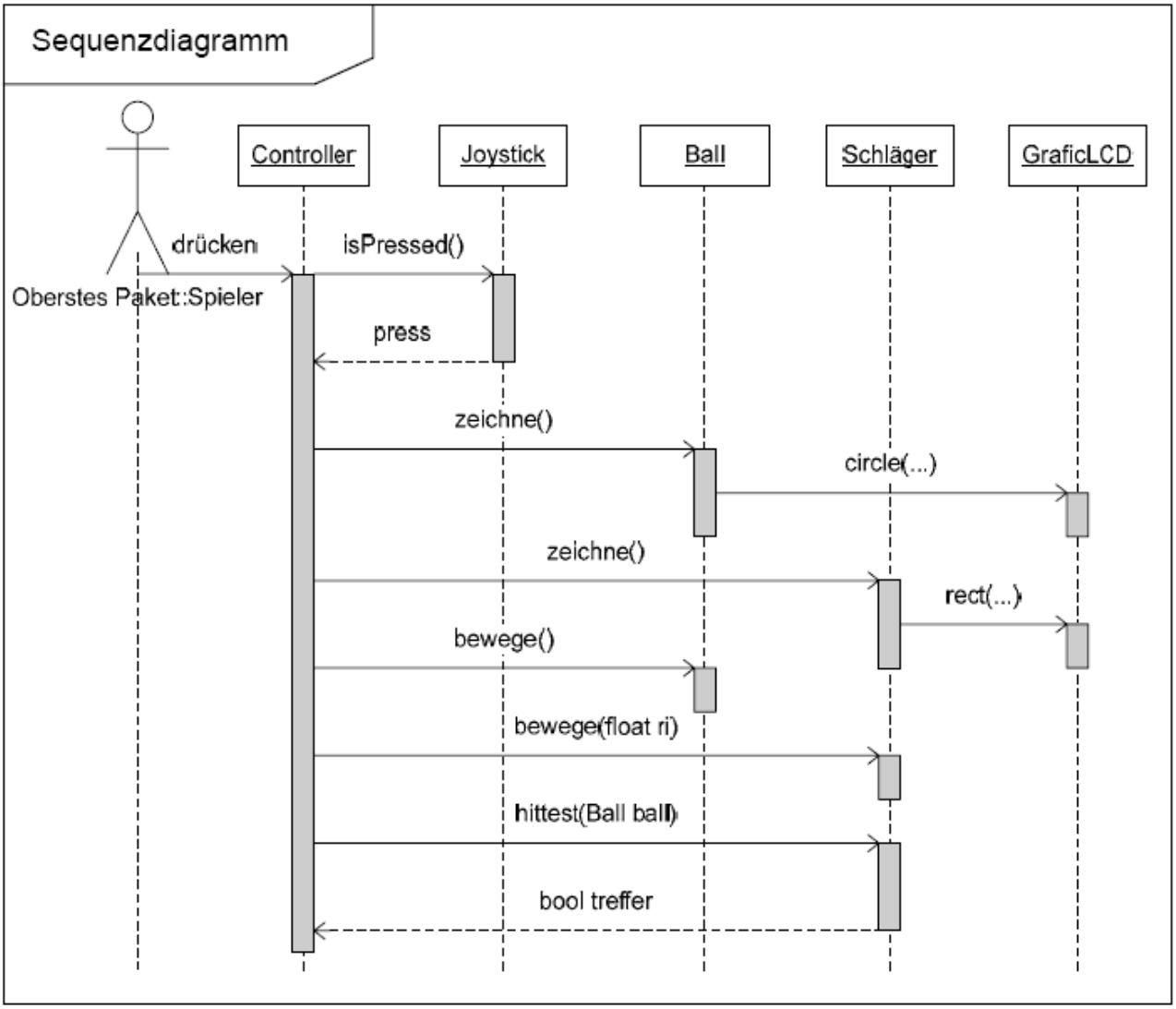

Abbildung 7 - Sequenzdiagramm (Kollision zwischen Ball und Schläger)

Jede Kollision des Balls mit dem Schläger erhöht den Punktestand um jeweils einen Punkt. Der Gesamtpunktestand wird bei Beendigung des Spiels auf dem LCD-Display angezeigt. Die Anzeige wird durch die vordefinierte Bibliothek myAVR_String übernommen. Die Realisierung findet in der Methode punkte_anzeigen() statt. Deren Eingabeparameter sind die Ballkoordinaten. Sie besitzt ebenfalls keine Rückgabewerte.

Klasse Rand:

Die Analysephase ergab, dass die Klasse Rand ihr Verhalten nur ändert, wenn der Ball das Spielfeld am unteren Rand verlässt. Die dafür eingeführte Methode verloren(), beinhaltet als Eingabeparameter die Koordinaten der aktuellen Ballposition. Diese werden mit der größten LCD-Position in y-Richtung verglichen und als Rückgabewert die Zustände wahr oder falsch vom Datentyp Boolean, kurz bool, ausgegeben. Zu den Beziehungen der Klassen untereinander ist zu sagen. Die Klassen Ball, Schläger und Rand sind teil des gesamten Spiels. Sie werden alle von der Klasse Controller gesteuert. Damit haben sie den Beziehungstyp Aggregation. Zwischen den Klassen Ball und Rand bzw. Ball und Schläger besteht eine Assoziation. Der Ball kollidiert mit dem Schläger, außerdem kann der Ball durch den Rand fallen. Die Klassen Rand und Schläger verarbeiten jeweils die Koordinaten des Balls durch Aufruf der Klasse Ball. Die Klasse Ball

dagegen nutzt keine Informationen aus den Klassen Rand und Schläger. Die im Klassendiagramm gezeigte Klasse app : Controller wird durch die Software automatisch ergänzt. Sie stellt eine Realisierung / Verfeinerung der Basisklasse Controller dar.

4 Realisierung

Die Phase der Realisierung beschäftigt sich mit der Implementierung des Spiels. Aus den vorangegangenen Phasen liegen folgende Informationen vor:

Vorliegende Informationen:	detailliertes Klassen- und Sequenzdiagramm
Aufgaben zur Lösung:	Implementierung mit softwaretechnischer Unterstützung
Ergebnis:	Spiel ist auf dem Mikrocontroller programmiert

Die Implementierung erfolgt mit Hilfe des Programms SiSy AVR, welches von der Laser & Co. Solutions GmbH entwickelt wurde. Die zu verwendende Sprache zur Programmierung des Mikrocontrollers kann zwischen Assembler, C und C++ gewählt werden. Die in dieser Arbeit behandelte Aufgabenstellung wurde mit C++ realisiert. Bei der Programmierung mit SiSy AVR, womit man Mikrocontroller brennen kann, entstehen zwei Dateien, in der der Quelltext drin steht. Die erste Datei lautet *.h, dies ist eine Headerdatei, in der nur die eingebundenen Klasssenbibliotheken, der Klassenname, die verwendeten Attribute mit entsprechender Sichtbarkeit und die Methodennamen stehen, deshalb wird diese auch als Deklarationsdatei bezeichnet. Dabei ist zu sagen, dass nur die Sichtbarkeit und der Name der Eigenschaften und des Verhaltens darin beschrieben sind, jedoch keine Startwerte. Die zweite Datei, namens *.cpp, ist die erzeugte Definitionsdatei in der Sprache C++, die Attribute, aber mit Startwerten enthalten, sowie die Implementierung der Methoden. In dieser Datei sind jedoch keine Sichtbarkeiten enthalten.

4.1 Umsetzung der Systemstruktur

Die Implementierung des Klassendiagramms (Abbildung 6) wird durch die Software automatisch in den Quellcode umgewandelt. Abbildung 8 zeigt die syntaktischen Unterschiede zwischen der Notation im Klassendiagramm und im Quellcode, am Beispiel der Klasse Ball.

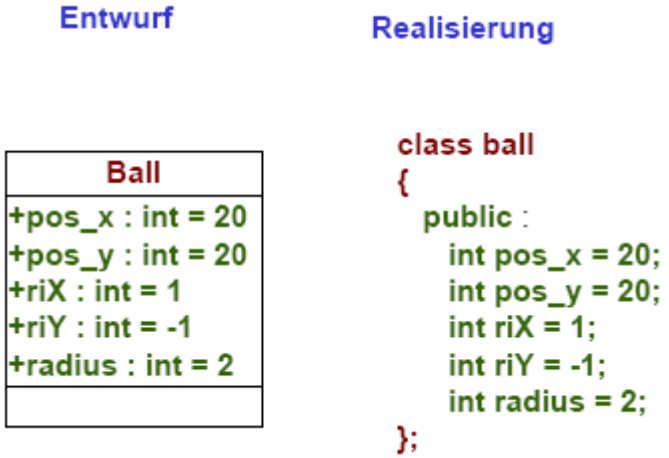

Abbildung 8 - Gegenüberstellung des Entwurfs und der Implementierung der Klasse Ball

Die Klassen Controller und Schläger haben denselben Aufbau wie die Klasse Ball, nur unterschiedliche Attribute und Datentypen. Die Abbildung 9 verdeutlicht, dass die Notation der Klasse Rand im Quellcode der gleichen Regel folgt. Da diese keine Attribute besitzt, sind diese an dieser Stelle im Quellcode nicht sichtbar.

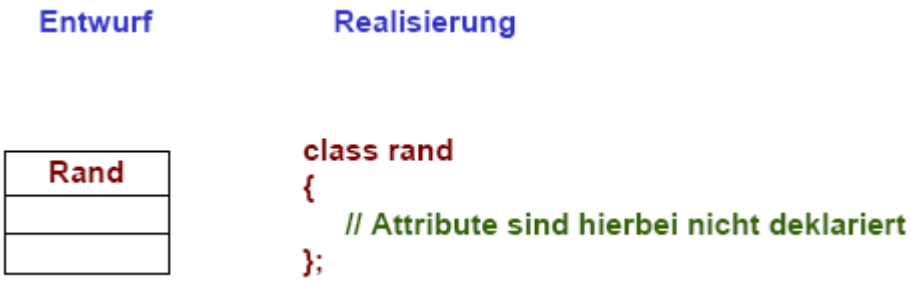

Abbildung 9 - Gegenüberstellung des Entwurfs und der Implementierung der Klasse Rand

4.2 Umsetzung des Systemverhaltens

Diese Phase beschäftigt sich mit der Implementierung der Methoden. In der main-Methode der Klasse Controller werden die Attribute initialisiert. Anhand des in Abbildung 10 dargestellten Quelltextauszugs (aus der Datei „Controller.cpp“) der Klasse Controller soll die Syntax erläutert werden.

```
void Controller::main()
// Initialisierung der Attribute
{
   // Jumper auf "LCD on Port C+A" stecken auf dem Board
   lcd.configLcd(PORTA, PORTC);
   // Rechtsbewegung des Joysticks
   // Angabe des zuständigen Ports und des
   // entsprechenden Bits
   btnRight.config(PORTK, BIT6);
   // Linksbewegung des Joysticks
   // Angabe des zuständigen Ports und des
   // entsprechenden Bits
   btnLeft.config(PORTK, BIT4);
};
```

Abbildung 10 - Quellcode-Auszug zur Initialisierung

```
void [Klassenname]::[Bezeichnung der Methode]()
{
   // Jumper auf "LCD on Port C+A" stecken auf dem Board
   [Name des Attributs].configLcd(PORTA, PORTC);
   // Rechtsbewegung des Joysticks
   [Name des Attributs].config(Port, bit);
};
```

Abbildung 11 - Erklärung des Quelltext-Auszuges

Die Anzeige auf dem LCD-Display wird durch das Umstecken des Jumpers von Port A+C off (Werkseinstellung) auf Port A+C on aktiviert. Der Joystick befindet sich bereits auf dem Board. Die Operation zur Initialisierung des Displays ist configLCD, die im Datentyp GraphicLCD, der Bibliothek myAVR_mk3Lcd, definiert ist. Der Joystick wird durch die vordefinierte Operation config initialisiert, die durch den Datentyp PushButton bereitgestellt wird.
Abbildung 12 zeigt die Gegenüberstellung des Entwurfs im UMLKlassendiagramm und der Realisierung, am Beispiel der Klasse Ball und einen Auszug der Methode bewege().

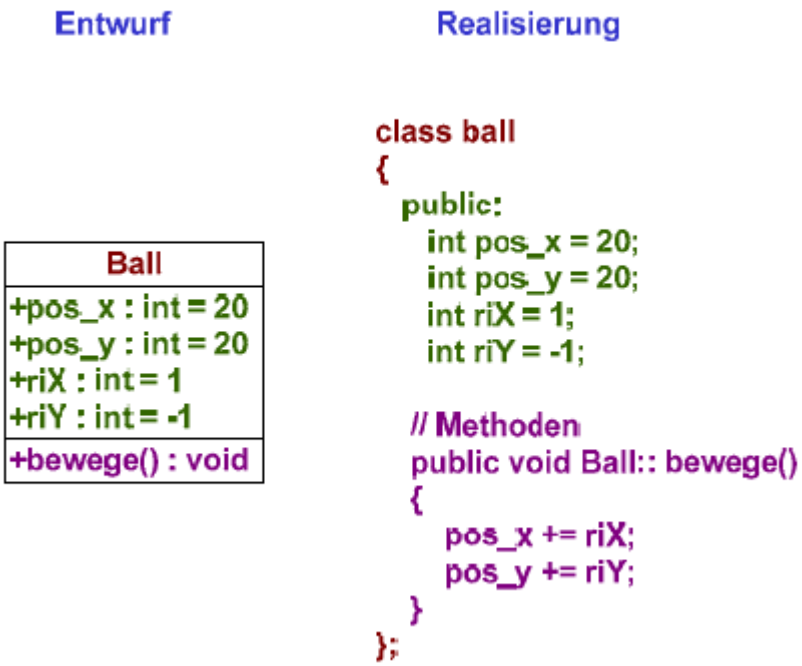

Abbildung 12 - Gegenüberstellung Entwurf und Realisierung

Der komplette Quelltext des Projektes ist in Anlage 2 enthalten.

4.3 Programmieren und Testen

Nachdem die Implementierung des Quellcodes hinter einem liegt, muss das Programm auf den Mikrocontroller gebrannt werden. Dazu müssen folgende Voraussetzungen geschaffen werden:

- Verbindung des PC, mit Software SiSy AVR, mit dem USB-Anschluss des Mikrocontrollers

Prüfen, ob der eingesetzte Controller auch korrekt vom System erkannt wird (Abbildung 13)

Abbildung 13 - Controllerauswahl

Der Start des Brennvorgangs geschieht über die Schaltfläche „Erstellen, Brennen und Ausführen“ (Abbildung 14).

Abbildung 14 - Schaltfläche zum Start des Brennvorgangs

Durch die Software werden dabei folgende Schritte im Hintergrund durchgeführt:

Quellcode	Erstellung durch den Programmierer
⇩	
Compilieren	Übersetzen des C++ Quellcodes in Maschinensprache Erstellung der *.o-Dateien aus *.h und *.cpp
⇩	
Linken	Einbinden der benötigten Bibliotheken und Klassen Erstellung der Hexadezimaldatei (*.hex)
⇩	
Brennen	Brennen der Maschinenbefehle (Hexadezimaldatei) auf den Flash des Controllers

Beim Brennvorgang werden elektrische Impulse auf den Controller gebracht. Da das Spiel auf den internen Flash gebrannt wird, kann es bis zu zehn tausendmal elektronisch überschrieben werden. Nach Abschluss der Implementierung folgt der Funktionstest. In diesem wird geprüft, ob das Spiel auf dem LCD-Display angezeigt wird und ob das Spiel gemäß Anleitung funktioniert. Der Schläger muss auf die Bewegung des Joysticks reagieren. Der Ball muss sich automatisch auf dem Display bewegen und vom rechten, linken und oberen Spielfeldrand abgestoßen werden. Des Weiteren muss der Ball mit dem Schläger kollidieren und die Punkte erhöhen. Bei durchfallen des Balls am unteren Rand müssen die Punkte auf dem LCD-Display angezeigt werden.

5 Zusammenfassung und Ausblick

Diese Belegarbeit behandelt die objektorientierte Implementierung von Anwendungen auf Mikrocontrollern. Nach Behandlung der Grundlagen wird die Umsetzung unter Verwendung der objektorientierten Vorgehensweise dargestellt. Die einzelnen Schritte auf dem Weg von der „Aufgabe zur fertigen Anwendung“ werden anhand eines praktischen Beispiels veranschaulicht. Das Projekt Superball bietet weiteres Optimierungspotenzial. Zum Beispiel kann mit weiteren Bausteinen, die Kollision des Schlägers mit dem Ball unterbunden werden.

Anlagen

Anlage 1

Techniken der Systemanalyse

Nominalphrasenanalyse

Der Ball bewegt sich ohne Steuerung des Spielers zick-zack-förmig ab einer definierten Startposition. Berührt der den linken, oberen und rechten Rand des Spielfeldes, wird er ebenfalls zick-zack-förmig abgestoßen. Am untern Bildrand bewegt sich der Schläger in horizontaler Richtung. Mit diesem versucht der Spieler zu verhindern, dass der Ball das Spielfeld am unteren Ende verlässt. Der Ball bewegt sich mit konstanter Geschwindigkeit. Der Schläger dagegen bewegt sich schneller als der Ball. Bei jedem Zusammentreffen des Schlägers mit dem Ball werden die Punkte jeweils um eins erhöht. Bewegt sich der Ball durch die untere Begrenzung des Spielfeldes wird die Punktzahl auf dem LC-Display angezeigt und das Spiel wird beendet. Durch betätigen der Reset-Taste wird der Punktestand auf Null zurückgesetzt und das Spiel neu gestartet.

Verbalphrasenanalyse

Der Ball bewegt sich ohne Steuerung des Spielers zick-zack-förmig ab einer definierten Startposition. Berührt der den linken, oberen und rechten Rand des Spielfeldes, wird er ebenfalls zick-zack-förmig abgestoßen. Am untern Bildrand bewegt sich der Schläger in horizontaler Richtung. Mit diesem versucht der Spieler zu verhindern, dass der Ball das Spielfeld am unteren Ende verlässt. Der Ball bewegt sich mit konstanter Geschwindigkeit. Der Schläger dagegen bewegt sich schneller als der Ball. Bei jedem Zusammentreffen des Schlägers mit dem Ball werden die Punkte jeweils um eins erhöht. Bewegt sich der Ball durch die untere Begrenzung des Spielfeldes wird die Punktzahl auf dem LC-Display angezeigt und das Spiel wird beendet. Durch betätigen der Reset-Taste wird der Punktestand auf Null zurückgesetzt und das Spiel neu gestartet.

Anlage 2

Quelltext der Mikrocontrollerlösung Superball

Klasse Controller

```
class Controller
{
	// eingebundene Klassen
	private:
	Ball ball;
	Schlaeger schlaeger;
	Rand rand;
	// Attribute
	public:
	GraficLcd lcd;
	PushButton btnRight;
	PushButton btnLeft;
	// Methoden
	public void main()
	{
		// diese Operation als Start-Operation für main angeben
		// hier Initialisierungen durchführen
		// Jumper auf "LCD on Port C+A" stecken auf dem Board
		lcd.configLcd(PORTA, PORTC);
		// Rechtsbewegung des Joysticks
		// Angabe des zuständigen Ports und des entsprechenden Bits
		btnRight.config(PORTK, BIT6);
		// Linksbewegung des Joysticks
		// Angabe des zuständigen Ports und des entsprechenden Bits
		btnLeft.config(PORTK, BIT4);

		// mainloop starten
		run();
	}

	public void run()
	{
		do
		{
			// bewege den Ball
			ball.bewege();
			// zeichne den Ball
			ball.zeichne();
			// warte 10 ms bis nächster Punkt gezeichnet wird, es wird
			// darüber auch der Schläger mitgesteuert, ohne diese
			// Zeile ist der Ball gar nicht zu sehen
			waitMs(10);
			// löscht den Inhalt auf dem LCD-Bildschirm
			// daraus folgt, dass ständig ein neuer Punkt gezeichnet
			// wird und der andere Punkt gelöscht wird
			lcd.clear();
			// der Schläger soll sich bewegen,
			// wenn der Joystick oder Button nach links gedrückt wird
			if (btnLeft.isPressed())
			{
			    // ..., dann bewege den Schläger nach links, also -1
			    // (da x verringert wird)
			    schlaeger.bewege(-1);
			}
			// rechter Rand
			if(schlaeger.pos_x2 == 127 || schlaeger.pos_x2 > 127)
```

```
                {
                    schlaeger.bewege(-1);
                }
                // wenn der Joystick oder Button nach rechts gedrückt wird
                if (btnRight.isPressed())
                {
                    // ..., dann bewege den Schläger nach rechts, also
                    // +1 (da x ständig erhöht wird)
                    schlaeger.bewege(+1);
                }
                // linker rand
                    if(schlaeger.pos_x1 == 0 || schlaeger.pos_x1 < 0)
                {
                    schlaeger.bewege(+1);
                }
                // zeichne den Schläger
                schlaeger.zeichne();
                // Kollisionsprüfung, wenn der Schläger ein „true“ an den
                // Ball zurücksendet, dann entscheidet der Ball selber was
                // er macht, wird angewiesen die Richtung von sich selbst
                // zu ändern
                if (schlaeger.hittest(ball))
                {
                    // wenn der Ball den Schläger getroffen hat, dann
                    // ändere die Y-Richtung des Balls auf -1 (Ball geht
                    // nach oben)
                    ball.riY *= -1;
                    // bei jedem Zusammenstoß Punkte zählen, immer
                    // um eins nach oben zählen
                    schlaeger.punkt = schlaeger.punkt + 1;
                    }

                    // wenn der Ball im unteren Rand verschwindet, ...
                    if (rand.verloren(ball))
                    {
                        // ... zeige die erreichten Punkte auf dem LCD an
                        schlaeger.punkte_anzeigen(ball);
                        // warte 500ms
                        waitMs(500);
                    }

                } while (true);
            }
}
```

Klasse Ball

```
class Ball
{
    // Attribute
    public:
        pos_x = 20;
        pos_y = 20;
        riX = 1;
        riY = -1;
        radius = 2;
    // Methoden
    public void zeichne()
    {
        // zeichne ein Pixel bei der angegebenen Position x und y
        app.lcd.circle(pos_x, pos_y, radius, true, LCD_MODE_SET);
    }

    public void bewege()
    {
        // der Ball soll sich automatisch bewegen, das heißt, dass ich auch
        // 2 Richtungen benötige, die x- und die y-Richtung
        // die Bewegung des Balles ist nur einzelner Schritt, da sich der
        // Ball senkrecht nach unten bewegt, ändern sich die x- und y-
        // Position des Balls zusammen
        // x-Position des Balls entspricht auch der x-Richtung
        pos_x += riX;
        // y-Position des Balls entspricht auch der y-Richtung
        pos_y += riY;
        // wenn Ball an Rand stößt, dann Richtung *1 rechnen wechseln
        // wenn die x-Position des Balls 0 oder 127 beträgt, ...
        if (pos_x == 0 || pos_x == 127)
        // ... dann ändere die Richtung des Balls, rechne *(-1) (Ball geht
        // nach unten)
        riX *= -1;
        // wenn y-Position des Balls 0 beträgt, ...
        if (pos_y == 0)
        // ... dann ändere auch diese Richtung, in dem *(-1) gerechnet wird
        // (Ball geht nach oben)
        riY *= -1;
    }
}
```

Klasse Schläger

```
class Schlaeger
{
    // Attribute
    public:
    float pos_x1;
    float pos_y1;
    Anlage 2 / Blatt 6
    float pos_x2;
    float pos_y2;
    volatile uint16_t punkt;
    float step;
    // Methoden
    public void zeichne()
    {
        // zeichne ein Rechteck bei der angegebenen Position
        app.lcd.rect(pos_x1, pos_y1, pos_x2, pos_y2, false,
        LCD_MODE_SET);
    }
    public void bewege(float ri)
    {
        // beim Bewegen, bleibt die y-Position gleich, nur die x-Position
        // ändert sich, deswegen brauche ich nur eine Richtung, außerdem
        // soll der Schläger sich nicht automatisch bewegen
        pos_x1 += ri*step;
        pos_x2 += ri*step;
    }
        // Kollision zwischen Ball und Schläger; Parameter: Klassenname, auf
        // die zurückgegriffen werden soll Parametername
        public bool hittest(Ball ball)
    {
        // der Schläger gibt Message, ob er vom Ball getroffen wurde,
        // entweder „true“ oder „false“; muss als Parameter den Ball vom
        // Typ Ball übergeben, bevor das passiert, muss eine gerichtete
        // Bewegung zwischen "Schläger und Ball" bestehen
        // Schlüsselwort "this" ist ein Verweis auf sich selbst
        if ((ball.pos_x >= this -> pos_x1) &&
        (ball.pos_x <= this -> pos_x2) &&
        (ball.pos_y >= this -> pos_y1) &&
        (ball.pos_y <= this -> pos_y2))
        {
            // wenn diese Maße erreicht sind, gib` „true" zurück
            return true;
        }
    }
    public void punkte_anzeigen(Ball ball)
    {
        // erreichte Punkte sollen auf dem LCD-Display angezeigt werden
        // lege die Variablen "txt" und "t1" vom Typ String an, damit der
        // Datentyp String akzeptiert wird, muss das Paket
        // "myAVR_String" hinein und in der Klassendiagramm-
        // Eigenschaft, bei „Includes für alle Klassen dieser Anwendung:"
        // der Typ "#define String String 16m" hinein
        String16m txt,t1;
        // die Variable "txt" enthält das Wort "Punkte:"
        txt = "Punkte: ";
        // die erreichten Punkte werden in der Variable "punkt" gesichert
        // dieser Inhalt der Variable wird mit der umgewandelten Zahl
        // gefüllt und in die Variable "t1" geschoben vom Datentypen String
        t1.fromInt(punkt);
        // die beiden Teile werden zusammengefügt
        txt += t1;
```

```
            // füge zum Schluss noch ein Leerzeichen dran
            txt += " ";
            // gebe die erreichten Punkte auf der Position (00, 32) aus
            app.lcd.setPos(00,32);
            // gebe auf dem LCD-Display die erreichten Punkte aus, die nun in
            // der Variable "txt" stehen
            app.lcd.writeTextRam(txt,LCD_MODE_SET);

            // Schläger vom LCD-Bildschirm löschen
            app.lcd.rect(pos_x1, pos_y1, pos_x2, pos_y2, false,
            LCD_MODE_CLR);
            // Ball vom LCD-Bildschirm löschen, läuft trotzdem weiter
            app.lcd.circle(ball.riX = 0, ball.riY = 0, ball.radius = 0, true,
            LCD_MODE_CLR);
        }
}
```

Klasse Rand

```
class Rand
{
    // Attribute
    // keine Attribute deklariert, Feld leer

    // Methode
    public bool verloren(Ball ball)
    {
        // beim Rand interessiert nur die y-Position nicht die x-Position
        // wenn der Ball die y-Position 63 oder höher erreicht hat, ...
        if (ball.pos_y >= 63)
        {
            // ... dann gib ein wahr (true) zurück
            return true;
        }
    }
}
```