abitur.nrw
BERUFSKOLLEG
Berufliches Gymnasium

Zentrale Abiturprüfung 2023
Hauptttermin
18.04.2023

Profil bildendes Leistungskursfach
Informatik
Fachbereich Informatik

Bearbeitungszeit: 270 Minuten
zusätzliche Rüstzeit: 30 Minuten

Unterlagen für die Schülerinnen und Schüler

Aufgabenstellung

Beschreibung der Ausgangssituation:

Sie arbeiten als Praktikant in der Firma Next Generation Pinball Machines (NGPM) GmbH.
NGPM GmbH ist ein junges Start-Up Unternehmen und möchte an die Erfolge der Flipper-Industrie der 1980er und 1990er Jahre anschließen.

Flipperautomaten sind Geschicklichkeitsspiele, bei denen die Spielerin oder der Spieler eine Kugel mittels zweier Schlaghebel auf einer schrägen Fläche im Spiel hält. Dabei muss die Kugel bestimmte Wege durch das Flipperlabyrinth zurücklegen. Diese Wege werden durch Aufträge beschrieben und es werden Punkte dafür vergeben.

Die heutige Computertechnik eröffnet neue Möglichkeiten gegenüber den damaligen Modellen. Ziel könnte es sein, bereits bestehende Flipperautomaten auf die neue Technik zu modifizieren oder gänzlich neue Systeme zu entwickeln. Sie haben die Gelegenheit, in einigen Bereichen mitzuarbeiten.

Aufgabe 1 – Datenbanksysteme

Die Firma möchte ein Webarchiv installieren, sodass alle bisher und zukünftig entwickelten Flippermodelle nach bestimmten Suchkriterien gelistet werden können.
Grundlage des Webauftrittes soll eine relationale Datenbank sein.

An der Entwicklung von Flippermodellen sind mehrere Mitarbeitende beteiligt. Mitarbeitende sind je einer Abteilung zugeordnet. Mitarbeitende können an mehreren Modellen gleichzeitig arbeiten. Ein Flippermodell wird jedoch ausschließlich von einem externen Designer bzw. einer Designerin entworfen. Flippermodelle bestehen aus einer Vielzahl von technischen Komponenten und optional aus Baugruppen. Technische Komponenten können direkt verbaut sein oder Bestandteil von Baugruppen sein.
Ein Filterkriterium zur Suche im Webarchiv sind die Ausstattungsmerkmale der einzelnen Flippermodelle (z. B. Gehäusematerial oder Softwareversion). Auch andere Ausstattungsmerkmale können auftreten.

1.1 (19 Punkte)

Erstellen Sie aus der obigen Beschreibung ein ER-Modell in MC-Notation. Auf die Angabe von Attributen soll verzichtet werden.

Der Einsatz der Software DIA ist zwingend vorgeschrieben.

Die produzierten Flipper werden an Kunden verkauft, die wahlweise einen Wartungsvertrag abschließen können. Das folgende ER-Modell modelliert diesen Zusammenhang:

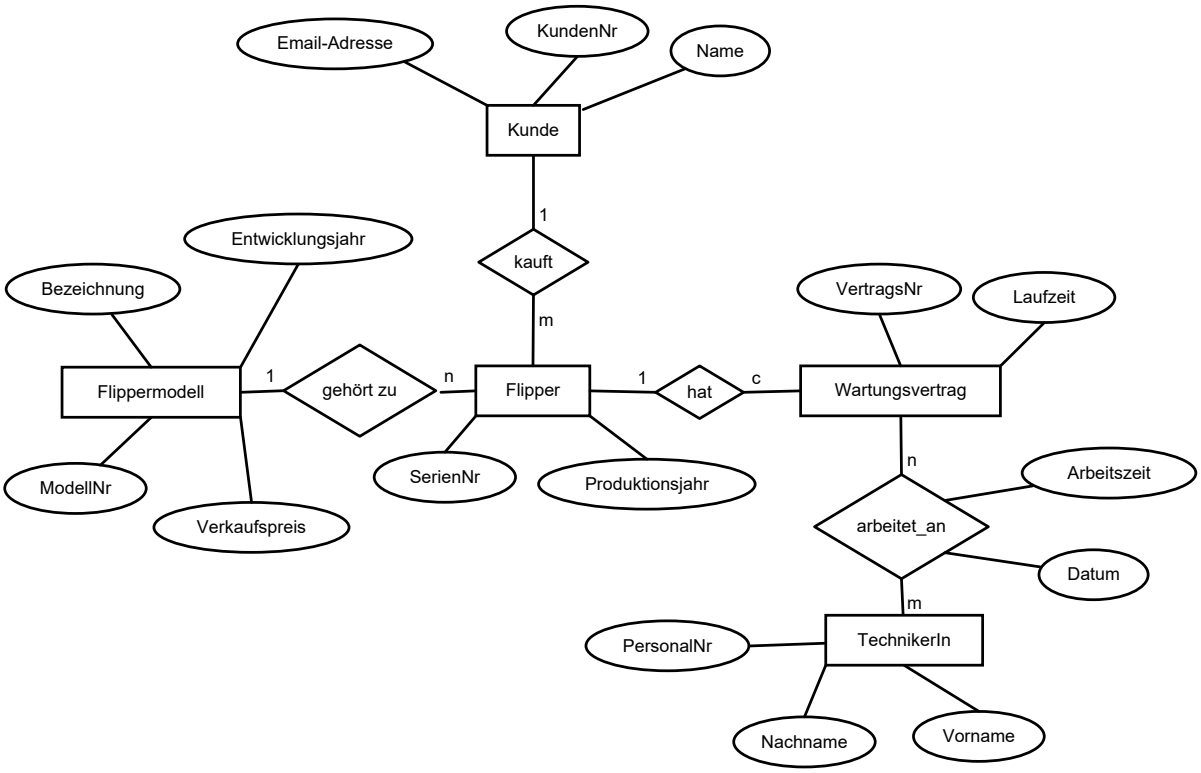

Abbildung 1: ER-Modell Wartungsvertrag

1.2 (14 Punkte)

Überführen Sie obiges ER-Diagramm in ein relationales Modell, das der dritten Normalform entspricht. Kennzeichnen Sie dabei Primärschlüssel durch Unterstreichen und Fremdschlüssel durch Voranstellen eines Pfeiles ↑.

Jedes Flippermodell besitzt eine eigene Software mit der Spiellogik. Fehler dieser Spiellogik werden durch neue Softwareversionen behoben. Um bei einem Download die Integrität sicherzustellen, wird zu der entsprechenden Version eine Prüfsumme ermittelt und in der entsprechenden Tabelle gespeichert. Ergänzend zum obigen ER-Modell (siehe Abbildung 1) soll die folgende Relation zur Anwendung kommen:

Software (VersionsNr, Pruefsumme,↑FlippermodellNr)

1.3 (5 Punkte)

Geben Sie eine SQL-Anweisung zur Erstellung dieser Tabelle unter Berücksichtigung der referentiellen Integrität an.

Die NGPM GmbH beschäftigt Softwareentwickelnde für die Programmierung der Spiellogik.
Die Softwareentwicklung ist für die einzelnen Flippermodelle in Projekten organisiert.
Hierbei werden jeweils die Stundenzahlen erfasst, die eine Mitarbeiterin oder ein Mitarbeiter an dem Projekt gearbeitet hat. Dabei können durchaus mehrere Eintragungen desselben Mitarbeiters oder derselben Mitarbeiterin zu ein und demselben Projekt existieren, falls es mehrere Zeitabschnitte gibt, in denen er oder sie an dem Projekt arbeitete.

In der Datenbank sind dazu unter anderem folgende Tabellen angelegt:

Mitarbeiter

PersonalNr	Nachname	Vorname	↑Abteilung
4001	Stern	Detlef	3
4020	Williams	Lara	1
4025	Jersey	Paula	2
…	…	…	…

Abteilung

AbtNr	AbtBezeichnung	↑AbtLeitung
1	Mechanik	4020
2	Elektrotechnik	3121
3	Softwareentwicklung	1302
…	…	…

Projekt

ProjNr	Flippermodell	↑ProjLeitung	ProjStart	ProjEnde
10	Soccer	1211	2022-05-11	2022-10-13
20	Sun Wars	4025	2020-06-12	2022-11-19
30	Zoo	4025	2021-10-12	
…	…	…	…	…

Bei laufenden Projekten ist noch kein Projektende eingetragen.

arbeitet_an

ErfassungID	↑Projekt	↑Mitarbeiter	Anzahl_Stunden
421	10	1240	34
422	10	1306	99
423	10	1306	178
424	20	1041	20
…	…	…	…

1.4 (5 Punkte)

Ermitteln Sie eine SQL-Abfrage, die für jedes Projekt, das bis zum 31.12.2022 fertiggestellt wurde, die Projektnummer, das Flippermodell sowie den Nachnamen und Vornamen der Projektleitung ausgibt.

1.5 (7 Punkte)

Ermitteln Sie mit Hilfe einer SQL-Abfrage die Gesamtzahl der Arbeitsstunden an den Projekten zum Flippermodell „Soccer“.

1.6 (9 Punkte)

Erstellen Sie eine SQL-Abfrage, die die Anzahl der Mitarbeitenden pro Projekt bestimmt für Projekte, die im Jahr 2022 sowohl gestartet als auch beendet wurden. Dabei sollen nur die Projekte mit mehr als 4 Mitarbeitenden sortiert nach der Anzahl der Mitarbeitenden angezeigt werden, wobei das Projekt mit der höchsten Anzahl zuerst erscheinen soll.

1.7 (6 Punkte)

Erklären Sie anhand eines selbst gewählten Beispiels den Unterschied der Tabellenverknüpfung mittels INNER JOIN bzw. LEFT JOIN.

1.8 (10 Punkte)

Erstellen Sie eine VIEW, welche den Vor- und Nachnamen, die Personalnummer und die Anzahl der laufenden Projekte aller Mitarbeitenden enthält, die an höchstens drei aktuell laufenden Projekten beteiligt sind.

Aufgabe 2 – Theoretische Informatik

Bei der Entwicklung eines Flipperautomaten kommen Konzepte der Theoretischen Informatik bei verschiedenen Aufgabenstellungen zum Einsatz.

2.1 (6 Punkte)

Geben Sie Definitionen für die Begriffe Alphabet, Wort und Sprache im Kontext der Theoretischen Informatik an.

2.2 (9 Punkte)

Geben Sie eine formale Definition eines deterministischen endlichen Automaten (DEA) an.

Beim Flipperautomaten beschreibt eine Mission eine Aufgabe für den Spieler, die Kugel über einen aus einzelnen Abschnitten bestehenden Weg zu lenken. Um die Mission zu erfüllen, muss die Kugel diese Abschnitte in ganz bestimmter Weise auf dem Spielfeld zurücklegen.

Gegeben ist der folgende deterministische endliche Automat, der eine Mission der Spiellogik abbildet:

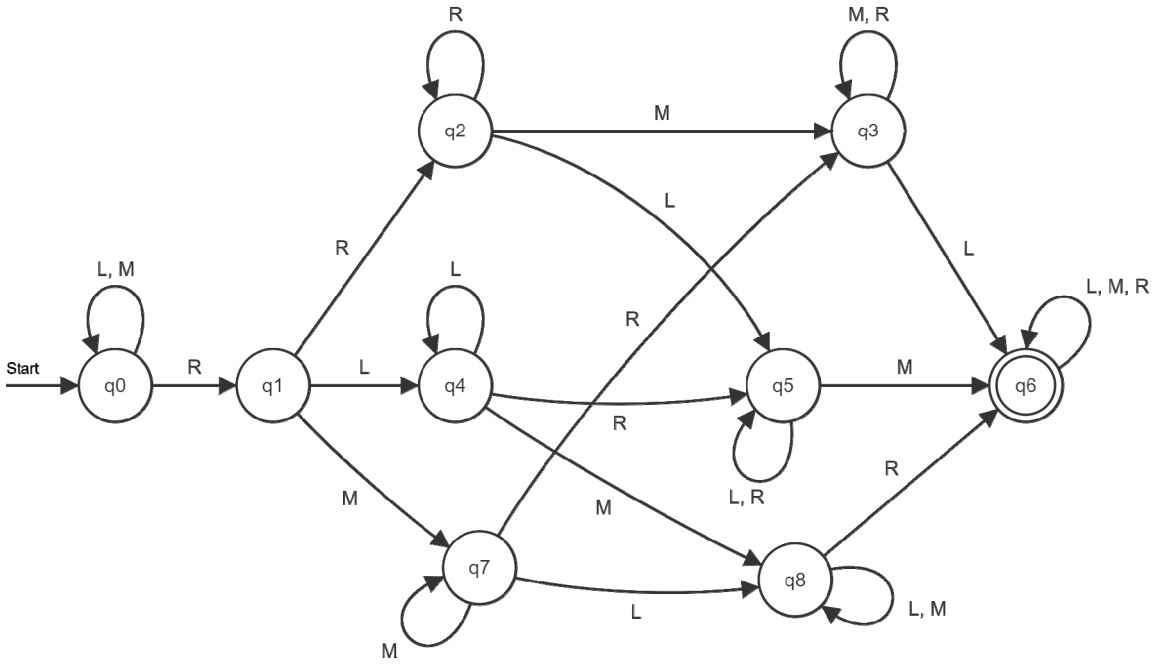

Abbildung 2: DEA Mission

L steht für Linke Rampe, M für Mittlere Rampe und R für Rechte Rampe.

2.3 (3 Punkte)

Geben Sie das Eingabealphabet, den Startzustand und die Menge der Endzustände des Automaten in Abbildung 2 an.

2.4 (6 Punkte)

Überprüfen Sie unter Angabe der durchlaufenen Zustände, ob die folgende Eingabe
L M R R R M R L zu einem Endzustand führt.

2.5 (4 Punkte)

Beschreiben Sie mit eigenen Worten, wann die Mission im Sinne des oben abgebildeten Automaten (siehe Abbildung 2) erfolgreich ist.

2.6 (10 Punkte)

Entwerfen Sie eine reguläre Grammatik, die diejenigen Wörter erzeugt, die den möglichen Abfolgen von Rampenläufen im Sinne des oben abgebildeten Automaten (siehe Abbildung 2) für eine erfolgreiche Mission entsprechen.

Abbildung 3 beschreibt eine andere Mission mit Hilfe eines nichtdeterministischen endlichen Automaten (NEA). Durch Erreichen eines Endzustandes ist die Mission beendet. L, M und R entsprechen dabei der Definition zu Aufgabe 2.3. Zusätzlich zu den Rampen werden jetzt auch sogenannte Targets betrachtet, also Ziele (Schalter) auf dem Spielfeld, die mit der Kugel angespielt werden müssen.
T1 und T2 bezeichnen im folgenden Automaten solche Targets.

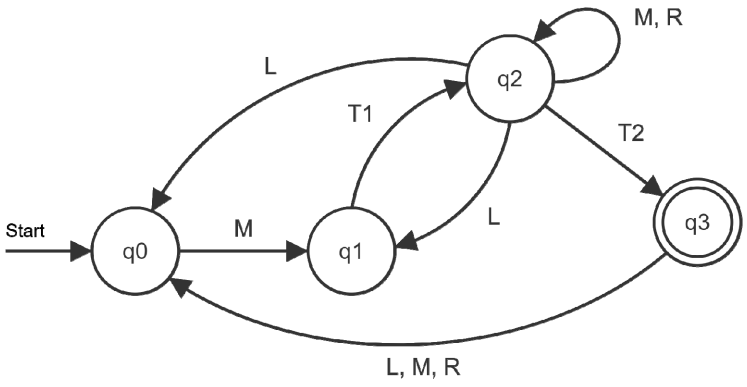

Abbildung 3: NEA weitere Mission

2.7 (4 Punkte)

Beschreiben Sie den Unterschied zwischen deterministischen und nichtdeterministischen endlichen Automaten.

2.8 (15 Punkte)

Erstellen Sie mithilfe der Potenzmengenkonstruktion einen vollständigen deterministischen endlichen Automaten, der die gleiche Sprache akzeptiert wie der Automat in Abbildung 3. Anzugeben sind sowohl die Übergangstabelle als auch der Graph des Automaten.

2.9 (4 Punkte)

Begründen Sie, dass die Menge der mit NEAs beschreibbaren Sprachen gleich der Menge der mit DEAs beschreibbaren Sprachen ist.

Sie haben die Idee für eine weitere spezielle Spielmission. Nachdem diese gestartet wurde, wird die linke Rampe (L) mindestens einmal durchlaufen. Anschließend wird die mittlere Rampe (M) mindestens zweimal durchlaufen. Um die Mission zu beenden, muss die linke Rampe (L) genauso oft durchlaufen werden, wie vor dem Durchlaufen der mittleren Rampe.

Dies kann durch die formale Sprache $S = \{L^n M^k L^n \mid n \geq 1, k \geq 2\}$ beschrieben werden.

2.10 (6 Punkte)

Untersuchen Sie, ob es einen deterministischen endlichen Automaten gibt, der die Sprache S akzeptiert.

2.11 (8 Punkte)

Erstellen Sie eine Grammatik, die die angegebene Sprache S erzeugt.

Aufgabe 3 – Dynamische Datenstrukturen

In der Entwicklungsabteilung gibt es eine Arbeitsgruppe, die sich mit der Programmierung der Spiellogik befasst. Dabei geht es u. a. um die Verwaltung von Ereignissen, das Aufsummieren von Punkten oder die Spielerverwaltung. Ereignisse treten immer dann auf, wenn die Kugel einen Sensor trifft. Dies ist beispielsweise beim Durchlaufen einer Rampe der Fall.
Sie sollen Vorschläge für einzelne Aspekte erarbeiten.

Für die Programmierung der Spiellogik stehen die dynamischen Datenstrukturen Stack und Queue zur Diskussion.

3.1 (4 Punkte)

Erklären Sie die Stack und Queue zugrundeliegenden Prinzipien.

Sie erinnern sich, dass zum Aufbau einer einfach verketteten Liste Containerobjekte verwendet werden können, die jeweils neben dem zu speichernden Objekt auch einen Verweis auf das nächste Containerobjekt enthalten.

3.2 (7 Punkte)

Geben Sie den Programmcode einer Containerklasse mit den beschriebenen Eigenschaften an. Dabei sollen neben den Attributen ein Konstruktor sowie get- und set-Methoden realisiert werden.

Listen können auch doppelt verkettet implementiert werden.

3.3 (4 Punkte)

Beschreiben Sie, welche Änderungen in der Containerklasse zur Realisierung einer doppelt verketteten Liste vorgenommen werden müssen.

Sie entscheiden sich, Ereignisse mithilfe der dynamischen Datenstruktur Queue zu verarbeiten. Die von der Kugel während des Spiels ausgelösten Ereignisse werden von der Klasse `FlipperEvent` (siehe Anlage 1) für die weitere Verarbeitung zur Verfügung gestellt.

Wenn die Kugel ein Ereignis auslöst, wird die Methode

```
private void ballEvent(FlipperEvent e)
```

der Klasse `Spiellogik` (siehe Anlage 2) aufgerufen.

3.4 (5 Punkte)

Beschreiben Sie die Bestandteile des Methodenkopfes der Methode `ballEvent`.

Die Methode `ballEvent` muss das `FlipperEvent e` nach verschiedenen Kriterien auswerten.

Handelt es sich um ein Event mit einer ID von 1 bis 99, so wird das `FlipperEvent e` zu der bereits initialisierten Queue `eventQueue` (siehe Anlage 3) hinzugefügt. Handelt es sich bei dem Event-Typ um einen Schalter („Switch“), so wird die Methode `switchLight` aufgerufen. Ist die Sensorgruppennummer 999, so wird die Methode `ballOut` aufgerufen.

3.5 (7 Punkte)

Implementieren Sie die Methode `ballEvent` mit den oben beschriebenen Anforderungen.

Im Programm werden zusätzlich Listen entsprechend der Vorgabe in Anlage 4 verwendet.

Exemplarisch soll zunächst die Methode `insert` der Klasse `List` betrachtet werden.

Gehen Sie davon aus, dass Zeiger `first`, `last` und `it` verwendet werden.

- `first` soll auf das erste Element der Liste zeigen oder auf null, falls die Liste leer ist.
- `last` soll auf das letzte Element der Liste zeigen oder auf null, falls die Liste leer ist.
- `it` (für Iterator) soll auf das aktuelle Element zeigen oder auf null, falls es kein aktuelles Element gibt.

3.6 (10 Punkte)

Stellen Sie für alle relevanten Fälle der Methode `insert` graphisch den Zustand der Liste sowie die Positionen der Zeiger `first`, `last` und `it` jeweils vor und nach dem Aufruf der Methode dar.

3.7 (10 Punkte)

Implementieren Sie die Methode `remove` der Klasse `List` gemäß der Beschreibung in Anlage 4. Sie können alle anderen Methoden der Klasse `List` und die oben beschriebenen Zeiger `first`, `last` und `it` als gegeben voraussetzen.

Verlässt der Ball das Spielfeld, ist das Spiel zu Ende und es werden die Ereignisse ausgewertet, um Bonuspunkte zu ermitteln. Da erfasst werden muss, ob bestimmte Sensorgruppen, beispielsweise die Schlagtürme (engl. Bumper), mehrfach getroffen wurden, sollen die in der Queue gespeicherten Ereignisse in eine Liste einsortiert werden.

Ein Spielverlauf wird durch die Queue `eventQueue` beschrieben, die beispielsweise folgendermaßen aussieht:

ID	sensorName	sensorType	sensorGroupNumber
14	Abschuss der Kugel	Plunger	1
21	Rampe rechts	Ramp	2
46	Schlagturm	Bumper	3
47	Schlagturm	Bumper	3
35	Schalter	Switch	4
21	Rampe rechts	Ramp	2
22	Rampe links	Ramp	2
23	Rampe mitte	Ramp	2
35	Schalter	Switch	4
19	Verlustbahn rechts	OutLaneRight	999

Die Ereignisse stehen in der angegebenen Reihenfolge (von oben nach unten) in der `eventQueue`. An der ersten Position steht immer das Ereignis „Abschuss der Kugel“. Der letzte Eintrag ist immer ein Sensor der Gruppe 999. Es gibt keinen weiteren Eintrag mit der sensorGroupNumber 999.

In der Methode `ballOut` sollen die Ereignisse der `eventQueue` sortiert in die Liste `eventList` eingefügt werden.

3.8 (12 Punkte)

Implementieren Sie die Methode `ballOut`, die das Einfügen aller Ereignisse der Queue `eventQueue` in die Liste `eventList` sortiert nach der SensorGroupNumber realisiert. Das Element mit der kleinsten SensorGroupNumber soll dabei am Anfang der Liste stehen.

Sie können alle Methoden der Klassen `Queue` und `List` (vgl. Anlagen 3 und 4) als gegeben voraussetzen und nutzen. Sie können davon ausgehen, dass die Liste bei Aufruf von `ballOut` leer ist.

In der Klasse `Spiellogik` (siehe Anlage 2) gibt es das Attribut `score` vom Typ `int`. In diesem Attribut wird der Punktestand erfasst, der auf dem Display des Flippers angezeigt wird. Auf Grundlage der Liste `eventList` sollen am Ende des Spiels die Bonuspunkte zum vorhandenen Punktestand in dem Attribut `score` addiert werden.

Bei den Bonuspunkten werden mehrfache Treffer von Schaltern derselben Gruppe gestuft nach Anzahl der Treffer mit Multiplikatoren versehen. Jeder Treffer wird mit 1000 Punkten belohnt. Unter 10 Treffern in einer Gruppe kommen keine weiteren Punkte hinzu. Werden in einer Gruppe 10 bis 19 Treffer erzielt, werden die Punkte mit dem Multiplikator 2, ab 20 bis 29 Treffern mit dem Multiplikator 3, ab 30 bis 39 Treffern mit dem Multiplikator 4 usw. multipliziert.

Beispiel:
Die Schlagtürme wurden 14-mal getroffen. Dann werden 14·1000·2 Bonuspunkte erzielt.

3.9 (16 Punkte)

Implementieren Sie die Methode `bonusScore`, die die Liste `eventList` verarbeitet und die entsprechenden Bonuswerte zum Attribut `score` addiert.

Sie können alle Methoden der Klasse `List` aus Anlage 4 als gegeben voraussetzen. Sie können eine nach der `sensorGroupNumber` aufsteigend sortierte Liste voraussetzen.

Anlagen

Anlage 1

Klasse FlipperEvent

```
public class FlipperEvent
{
    private int id;
    private String eventType;
    private String sensorName;
    private int sensorGroupNumber;

    public FlipperEvent (int id, String eT, String sN, int sGN)
    {
        this.id=id;
        this.eventType = eT;
        this.sensorName = sN;
        this.sensorGroupNumber = sGN;
    }

    public int getID() {return id;}

    public String getEventType() {return eventType;}

    public String getSensorName() {return sensorName;}

    public int getSensorGroupNumber() {return sensorGroupNumber;}
}
```

Anlage 2

Klasse Spiellogik

```
public class Spiellogik
{
  private Queue eventQueue;
  private List eventList;
  private int score;
  [...]

  private void ballEvent(FlipperEvent e) { … } // Aufgabe 3.5

  private void switchLight(FlipperEvent e) { … }

  private void ballOut() { … } // Aufgabe 3.8

  private void bonusScore(){ … } // Aufgabe 3.9

  [...]
}
```

Anlage 3

Dokumentation der Klasse Queue

Konstruktor	**Queue()** Eine leere Schlange wird erzeugt.
Anfrage	**boolean isEmpty()** Die Anfrage liefert den Wert true, wenn die Schlange keine Objekte enthält, sonst liefert sie den Wert false.
Auftrag	**void enqueue(Object pObject)** Das Objekt pObject wird an die Schlange angehängt. Falls pObject gleich null ist, bleibt die Schlange unverändert.
Auftrag	**void dequeue()** Das erste Objekt wird aus der Schlange entfernt. Falls die Schlange leer ist, wird sie nicht verändert.
Anfrage	**Object front()** Die Anfrage liefert das erste Objekt der Schlange. Die Schlange bleibt unverändert. Falls die Schlange leer ist, wird null zurückgegeben.

Anlage 4

Dokumentation der Klasse List

Konstruktor	**List()** Eine leere Liste wird erzeugt.
Anfrage	**boolean isEmpty()** Die Anfrage liefert den Wert true, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert false.
Anfrage	**boolean hasAccess()** Die Anfrage liefert den Wert true, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert false.
Auftrag	**void next()** Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt.
Auftrag	**void toFirst()** Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

Auftrag	**void toLast()** Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.
Anfrage	**Object getObject()** Falls es ein aktuelles Objekt gibt, wird das aktuelle Objekt zurückgegeben, andernfalls gibt die Anfrage den Wert null zurück.
Auftrag	**void setObject(Object pObject)** Falls es ein aktuelles Objekt gibt und pObject ungleich null ist, wird das aktuelle Objekt durch pObject ersetzt.
Auftrag	**void append(Object pObject)** Ein neues Objekt pObject wird am Ende der Liste angefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt pObject in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls pObject gleich null ist, bleibt die Liste unverändert.
Auftrag	**void insert(Object pObject)** Falls es ein aktuelles Objekt gibt, wird ein neues Objekt vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt, wird pObject in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt und die Liste nicht leer ist oder pObject gleich null ist, bleibt die Liste unverändert.
Auftrag	**void concat(List pList)** Die Liste pList wird an die Liste angehängt. Das aktuelle Objekt bleibt unverändert. Falls pList null oder eine leere Liste ist, bleibt die Liste unverändert.
Auftrag	**void remove()** Falls es ein aktuelles Objekt gibt, wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr. Wenn die Liste leer ist oder es kein aktuelles Objekt gibt, bleibt die Liste unverändert.

Materialgrundlage

Alle Materialien wurden selbst erstellt.

Zugelassene Hilfsmittel

- Wörterbuch der deutschen Rechtschreibung
- Graphikfähiger Taschenrechner (GTR) oder Computeralgebrasystem (CAS)
- Eine Nutzung von Computersystemen ist verpflichtend vorgesehen. Als Programme werden „Dia“ Version 0.97 oder höher und ein Texteditor eingesetzt.

Punktevergabe und Arbeitszeit

Inhaltliche Leistung	225 Punkte
Darstellungsleistung	15 Punkte
Gesamtpunktzahl	240 Punkte

Bearbeitungszeit	270 Minuten
zusätzliche Rüstzeit	30 Minuten