KOOPERATIVE STEUERUNG VON MODELLVERSUCHSFAHRZEUGEN

ENTWICKLUNG EINES INTELLIGENTEN FAHRENTSCHEIDERS

Softwareentwicklungspraktikum
Sommersemester 2008

Validierung zum System

E C A R

Auftraggeber
Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Prof. Dr.-Ing. Lars Wolf
Mühlenpfordstrasse 23
38106 Braunschweig

Betreuer: Kai Homeier, Carina Flämig
Phasenverantwortlicher: Wira Kakar

Auftragnehmer

Name	E - Mail
Wira Kakar	wirak@web.de
Chen Zhiwei	czwnii@hotmail.com
Rayan Merched El Masri	rayan_masri@hotmail.com
Ekrem Enes Duman	dumanenes@hotmail.de
Günter Dusch	g.dusch@Googlemail.com

Braunschweig, 09.07.2008

Versionsübersicht

Version	Datum	Autor	Status	Kommentar
1.0	07.07.08	Alle Auftragsnehmer		abgeschlossen
2.0	10.07.08	Alle Auftragsnehmer		Verbesserungen: Code hervorgehoben, Rechtschreib- und Grammatikfehler korrigiert, Layout optimiert

I n h a l t s v e r z e i c h n i s

1 EINLEITUNG 5

2 TESTPLAN 6

2.1 ZU TESTENDE KOMPONENTEN 6
2.2 ZU TESTENDE FUNKTIONEN 6
2.3 NICHT ZU TESTENDE FUNKTIONEN 7
2.4 VORGEHEN 7
2.5 TESTUMGEBUNG 8

3 TESTDURCHFÜHRUNG 9

3.1 TESTFALL /T1/: BILDERKENNUNG 9
3.1.1 /T130/ SEARCHBALL 9
3.2 TESTFALL /T2/: POSITIONIERUNG 14
3.2.1 /T202/ POSITION 14
3.2.2 /T210/ DIVIDE 15
3.2.3 /T211/ VISITEDWAY 18
3.2.4 /T212/ LINEOFSIGHT 20
3.2.5 /T213/ ANGLE 29
3.2.6 /T214/ MAPEXPLORED 31

3.3 TESTFALL /T3/: FAHRSTRATEGIE 34
3.3.1 /T300/ GOALCELL 34
3.3.2 /T301/ DECIDEANGLE 36
3.3.3 /T302/ DECIDESEGMENT 38
3.3.4 /T305/ ISNEW 40

3.4 TESTFALL /T4/: GESAMTTEST 43

4 ZUSAMMENFASSUNG 48

A b b i l d u n g s v e r z e i c h n i s

Abbildung 1: Der Ball wird aus einer 1,5 m Entfernung erkannt……..……...9

Abbildung 2:Der Ball wird aus einer 1,2 m Entfernung erkannt…….……....10

Abbildung 3:Der Ball wird aus einer 0,9 m Entfernung erkannt……..……...11

Abbildung 4: Ball wird aus einer 1,8 m Entfernung nicht erkannt 12

Abbildung 5: Falsches Ergebnis 13

Abbildung 6: Ein Labyrinth ohne Hindernisse 20

Abbildung 7: Ein Labyrinth mit Hindernisse 24

Abbildung 8: Bestimmung des Winkels mit der Methode decideAngle.....36

Abbildung 9: Die unbesuchten Zellen zwischen (0,0) und (6,7) im 1.Fall 41

Abbildung 10: Die unbesuchte Zellen zwischen (0,0) und (6,7) im 2.Fall 42

Abbildung 11: : Labyrinth- Test 1.....43

Abbildung 12: : Labyrinth- Test 2.....44

Abbildung 13: : Labyrinth- Test 3.....45

Abbildung 14: : Labyrinth- Test 4.....46

Abbildung 15: : Labyrinth- Test 5.....47

1 Einleitung

Softwaretests sind analytische, dynamische und vor allem unverzichtbare Maßnahmen zur Qualitätssteigerung von Softwaresystemen und zur Sicherstellung der Funktionsfähigkeit unter eventuell unerwarteten, aber im täglichen Gebrauch durchaus vorkommenden Bedingungen. Ziel ist, dass das Softwaresystem den Anforderungen des Nutzers genügt, wobei der Programmierer während der Implementierungsphase nicht alle Konfliktpotenziale vorausahnen und den Code entsprechend anpassen kann. Somit ist die Validierung unverzichtbar.
Eine Untersuchung der Software ECAR findet nach der Testfallspezifikation statt. Dabei kontrolliert man die zu benutzenden Eingabewerte und die erwarteten Ausgabewerte, d.h. die Testdokumentation zum Softwaresystem ECAR wird Ist- und Sollwerte der einzelnen Komponenten und Methoden auf Übereinstimmung prüfen. Dies ist ein manueller Vorgang.
Außerdem findet der Einsatz einer Simulationsumgebung statt, um unter Anderem auch das funktionsfähige Zusammenspiel der diversen einzelnen Komponenten mit diesem Test gewährleisten zu können.

2 Testplan

2.1 Zu testende Komponenten

- **/T1/** Bilderkennung
- **/T2/** Positionierung
- **/T3/** Fahrstrategie
- **/T4/** Gesamttest

2.2 Zu testende Funktionen

- **/F130/** searchBall
- **/F202/** position
- **/F210/** divide
- **/F211/** visitedWay
- **/F212/** lineOfSight
- **/F213/** angle
- **/F214/** mapExplored
- **/F300/** goalCell
- **/F301/** decideAngle
- **/F302/** decideSegment
- **/F305/** isNew

2.3 Nicht zu testende Funktionen

- **/F134/** getBallFound
- **/F135/** setPicMess
- **/F136/** getPicMess
- **/F137/** getCenter
- **/F200/** getCurrent
- **/F201/** setCurrent
- **/F203/** saveVisitedPoint
- **/F204/** getOld
- **/F205/** setOld
- **/F212/** setVisited
- **/F213/** getVisited
- **/F214/** setDirectWay
- **/F215/** getDirectWay
- **/F303/** getGoal
- **/F306/** getMaxi

2.4 Vorgehen

Zuerst werden die zu untersuchenden Methoden auf korrekte Ausgabewerte geprüft, indem die Eingabewerte bestimmt werden und getestet wird, ob die erwarteten Werte tatsächlich der Ausgabe entsprechen.

Nachdem dieses beschriebene Testvorgehen in Bezug auf die relevanten einzelnen Methoden abgeschlossen ist, wird der Gesamttest durchgeführt, sodass das erfolgreiche Zusammenspiel der einzelnen Methoden als Ganzes untersucht werden kann. Hierfür ist die Verwendung des Simulators vorgesehen.

2.5 Testumgebung

Die Testumgebung ist primär eine vom Institut IBR breitgestellte Simulationsumgebung, die das Verhalten des Roboters und den jeweiligen einzelnen Komponenten, die zum Roboter gehören, in einem einstellbaren Labyrinth bezogen auf den entwickelten Programmiercode des Softwaresystems ECAR simuliert. Somit kann eindeutig festgestellt werden, ob das Softwaresystem in dem Sinne funktionsfähig ist, dass alle Ziele und Aufgaben erfolgreich erledigt werden, d.h. dass das Labyrinth angemessen abgesucht, der Roboter sich adäquat fortbewegt und der Ball gefunden wird. Es ist jedoch noch festzuhalten, dass die Simulationsumgebung eine stark vereinfachte Form der Realität darstellt und somit teilweise Störfaktoren unberücksichtigt bleiben.

3 Testdurchführung

3.1 Testfall /T1/: Bilderkennung

3.1.1 /T130/: searchBall

Die Methode `searchBall` wendet den HoughCircle-Algorithmus aus der OpenCV-Bibliothek auf die Bilder an. Wird ein kreisförmiges Objekt in einem Bild gefunden, so wird der Algorithmus erneut angewendet. Wenn die Mittelpunkte der beide Kreise miteinander übereinstimmen (Toleranz zwischen -3 Pixel und +3 Pixel) wird der Wert `ballFound` auf `true` gesetzt.

Fall 1:

Testrahmen:

Entfernung 1.5 Meter

Arbeitsraum mit Störvariablen (Tische und Stühle).

Ausgabe:

Ball Found: 1

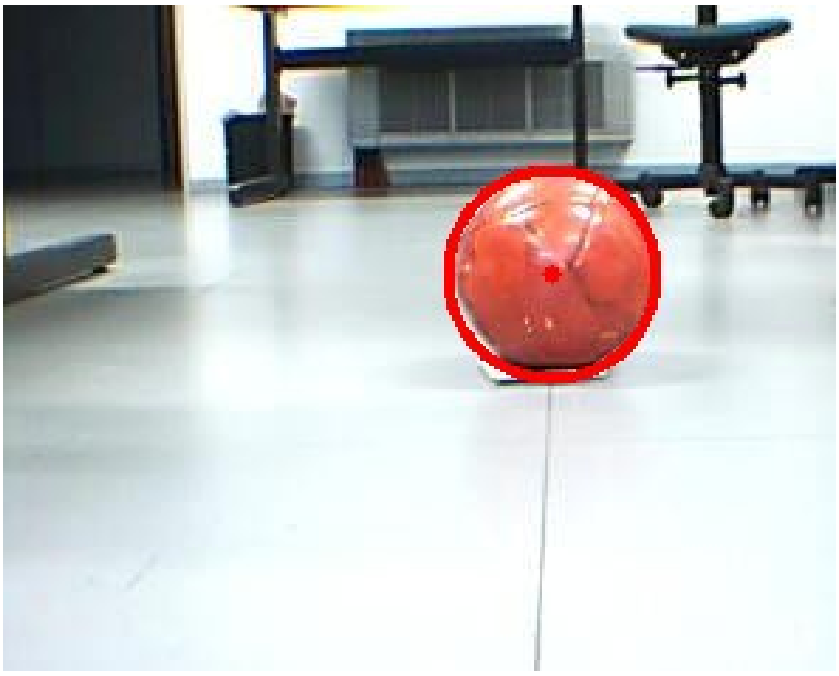

Abbildung 1: Der Ball wird aus einer 1.5 m Entfernung erkannt

Ergebnis: Der Ball wird erkannt

Fall 2:

Testrahmen:

Entfernung 1.2 Meter

Arbeitsraum mit Störvariablen (Tische und Stühle).

Ausgabe:

Ball Found: 1

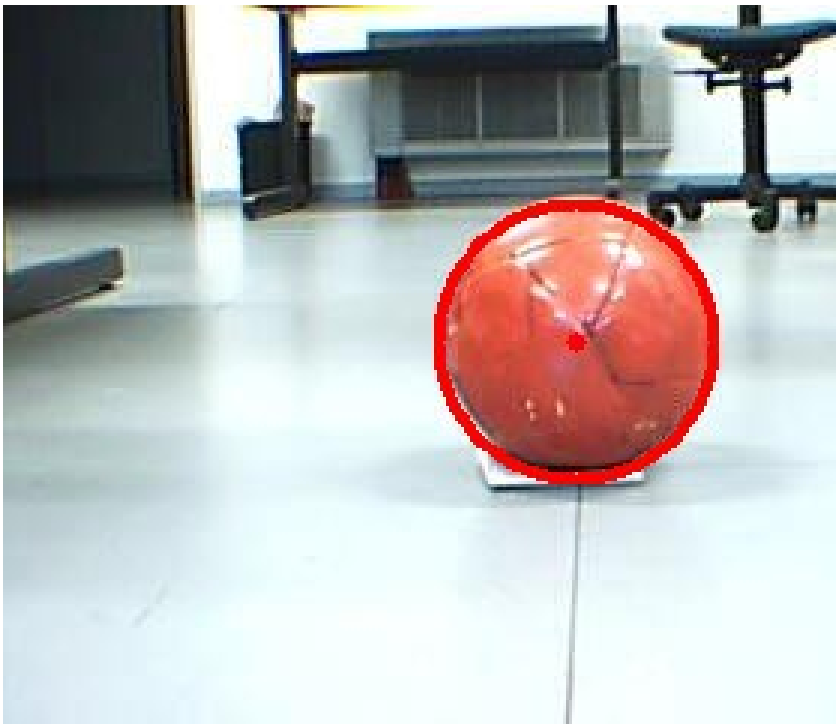

Abbildung 2: Der Ball wird aus einer 1.2 m Entfernung erkannt

Ergebnis: Der Ball wird erkannt.

Fall 3:

Testrahmen:

Entfernung 0.9 Meter

Arbeitsraum mit Störvariablen (Tische und Stühle).

Ausgabe:

Ball Found: 1

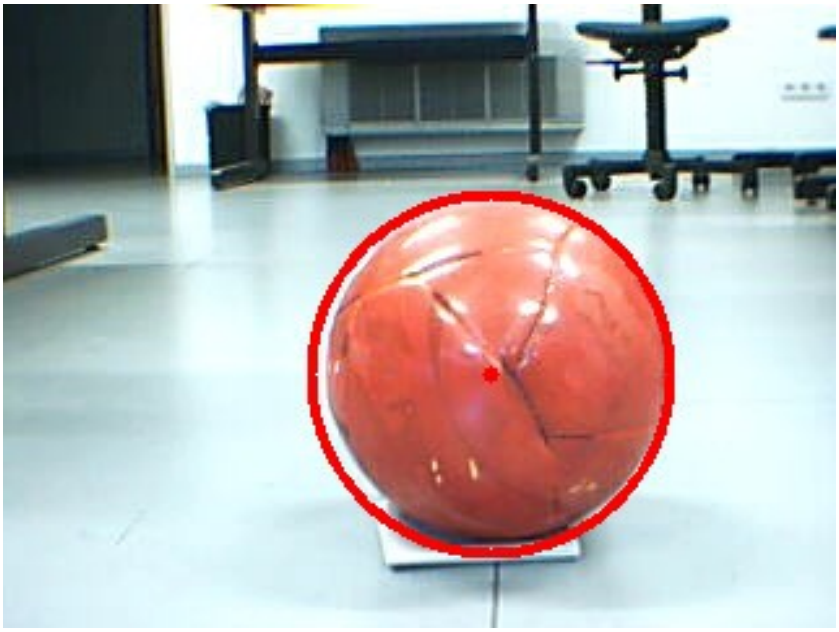

Abbildung 3: Der Ball wird aus einer 0.9 m Entfernung erkannt

Ergebnis: Der Ball wird erkannt.

Fall 4:

Testrahmen:

Entfernung 1.8 Meter

Arbeitsraum mit Störvariablen (Tische und Stühle).

Ausgabe:

Ball Found: 0

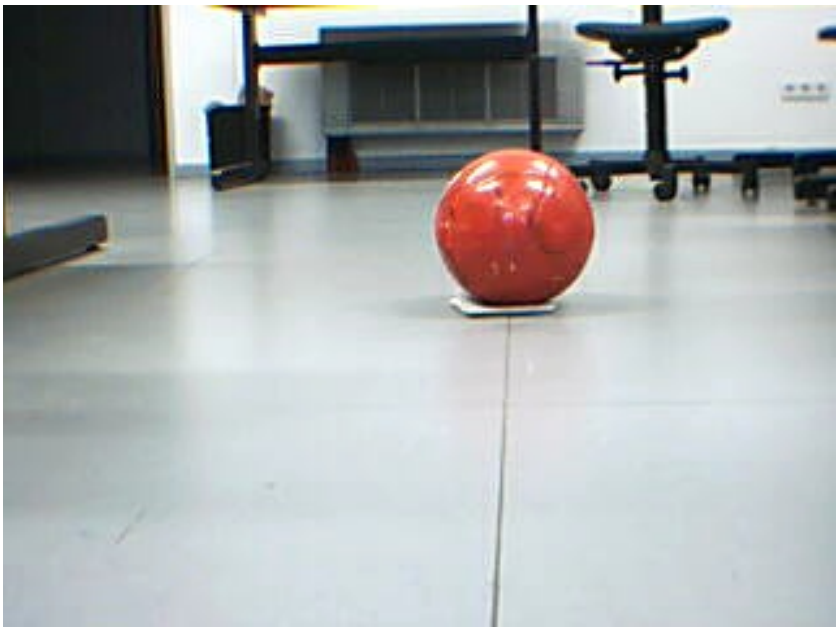

Abbildung 4: Der Ball wird aus einer 1.8 m Entfernung nicht erkannt

Ergebnis: Der Ball wird nicht erkannt.

Fazit: Der Ball wird erkannt aus einer Entfernung bis zu 1.5 Meter.

Fall 5:

Testrahmen:

Entfernung ca. 1.3 Meter

Arbeitsraum mit Störvariablen (Tische und Stühle).

Ausgabe:

Ball Found: 1

Abbildung 5: Falsches Ergebnis

Ergebnis: Ein kreisförmiges Objekt wird fälschlicherweise als Ball erkannt.

Lösung:

Solche falschen Ergebnisse werden vermieden, indem der Roboter die Bilder bis zum Erreichen des Balls (Entfernung von ca. 30 cm) analysiert. Während der Fahrt verschwinden diese Störvariablen aus dem Bild und der Wert der Variablen `ballFound` wird wieder auf `false` gesetzt. Die Suche wird fortgesetzt.

3.2 Testfall /T2/: Positionierung

3.2.1 /T202/ position

Beschreibung:

In dieser Methode wird die aktuelle Position vom Fahrzeug berechnet. Der Parameter `part` ist die Distanz zwischen dem alten Punkt und dem aktuellen Punkt. Zuerst wird die Steigung des Wegs zwischen dem alten Punkt und dem aktuellen Punkt berechnet. Dann können wir die aktuelle Position berechnen.

1. Fall: Wir fahren von dem Punkt (0,0) zu dem Zielpunkt (9,10) in einem Labyrinth. Die Distanz zwischen dem Punkt (0,0) und dem Punkt des Fahrzeug ist 25cm.

Eingaben:

`part`: 1.5m

Programmstück:

```
Position pos;
void Position:: position(double part)
{
       double beta= Labyrinth::angle(getOld(0), getOld(1),
       Decision::getGoal(0),Decision::getGoal(1));
       current[0]= int(part/cell*cos(beta))+old[0];
       current[1]= int(part/cell*sin(beta))+old[1];
        cout<<"Aktuelle x-Koordinate: "<< current[0]<<endl;
     cout<<"Aktuelle y-Koordinate: "<< current[1]<<endl;

}
```

Ausgabe:

Aktuelle x-Koordinate: 4

Aktuelle y-Koordinate: 4

Ergebnis: Test ist erfolgreich.

3.2.2 /T210/ divide

Beschreibung:

In unserem Algorithmus müssen wir unser Labyrinth in Zellen 25 x 25 cm teilen. Dies wird durch die Methode `divide(int labyrinthLength, int labyrinthWidth, int cell)` erfüllt. In dieser Methode wird angenommen, dass der Rest der Division auch eine Zelle ist, wenn es mehr als die Hälfte der Zelle beträgt. Dazu wird hier geprüft, ob es richtige Werte für die verschiedenen Fälle berechnet.

1.Fall

Eingaben:

`labyrinthlength`: 500cm

`labyrinthwidth` : 500cm

`cell` : 25cm

Programmstück:

```
Labyrinth lab;
int lableng=500;
int labwid= 500;
int cell=25;
lab.divide(lableng,labwid,cell);
cout<<"Zeilenanzahl: "<<lab.getMaxi(0)<<endl;
cout<<"Spaltenanzahl: "<<lab.getMaxi(1)<<endl;
```

Ausgabe:

Zeilenanzahl: 20

Spaltenanzahl: 20

Ergebnis: Test ist erfolgreich.

2.Fall

Eingaben:

`labyrinthlength`: 510cm

`labyrinthwidth` : 500cm

`cell` : 25cm

Programmstück:

```
Labyrinth lab;
int lableng=500;
int labwid= 500;
int cell=25;
lab.divide(lableng,labwid,cell);
cout<<"Zeilenanzahl: "<<lab.getMaxi(0)<<endl;
cout<<"Spaltenanzahl: "<<lab.getMaxi(1)<<endl;
```

Ausgabe:

Zeilenanzahl: 20

Spaltenanzahl: 20

Ergebnis: Test ist erfolgreich.

3.Fall

Eingaben:

`labyrinthlength`: 520cm

`labyrinthwidth` : 500cm

`cell` : 25cm

Programmstück:

```
Labyrinth lab;
int lableng=500;
int labwid= 500;
int cell=25;
lab.divide(lableng,labwid,cell);
cout<<"Zeilenanzahl: "<<lab.getMaxi(0)<<endl;
cout<<"Spaltenanzahl: "<<lab.getMaxi(1)<<endl;
```

Ausgabe:

Zeilenanzahl: 21

Spaltenanzahl: 20

Ergebnis: Test ist erfolgreich.

3.2.3 /T211/ visitedWay

Beschreibung:

In dieser Methode wird der zurückgelegte Weg berechnet und als besucht markiert. Zunächst werden alle Punkte auf dem zurückgelegten Weg berechnet und dann werden die Zellen, die diese Punkte enthalten, als `visited` markiert.

1.Fall

Wir fahren von dem Punkt (0,0) zu dem Punkt (6,7) in einem Labyrinth mit 200x200cm

Eingaben:

`labyrinthlength`: 200cm

`labyrinthwidth` : 200cm

`cell` : 25cm

Programmstück:

```
dec.goalCell(pos.getCurrent(0),pos.getCurrent(1));
dec.goalCell(pos.getCurrent(0),pos.getCurrent(1));
eCar.turn(dec.decideAngle(pos.getCurrent(0),pos.getCurrent(1)));
eCar.waitForStop();
while((eCar.getSensors()->readSensor( 0 )<50) && !sr.getBallFound()
&& dec.decideSegment()>part)
{
     eCar.drive(pos.cell);
     eCar.waitForStop();
     part=part+pos.cell;
}
pos.position(part);
lab.visitedWay();
for(int i=0;i<8;i++)
{
     for(int j=0;j<8;j++)
```

```
            cout<<lab.getVisited(i,j)<<"  ";
      cout<<endl;
}
```

Ausgabe:

S 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 **F**
0 0 0 0 0 0 0 0

S= Startposition, F= Finalposition, 1= Visited, 0= Unvisited

Ergebnis: Test ist erfolgreich.

3.2.4 /T212/ LineOfSight

Beschreibung:
Wenn es bei der Fahrt zwischen beiden Zellen ein Hindernis gibt, dann setzt diese Methode alle Zellen auf diesem Weg bis zu dem Hindernis und bis zur Zielzelle die Attribute `directWayy[][]` auf `false`.

1. Fall: Es gibt kein Hindernis zwischen den Zellen.

Eingaben: Startposition und Zielposition. Kein Hindernis dazwischen.

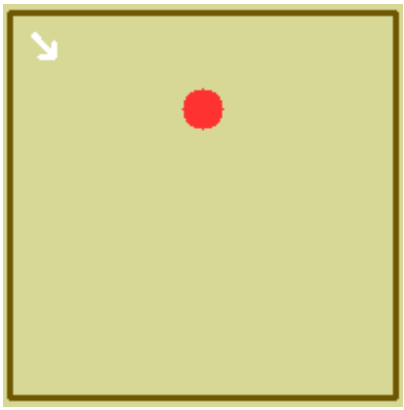

Abbildung 6: Ein Labyrinth ohne Hindernisse

Programmstück:

```
lab.divide(lableng,labwid,cell);
pos.setCurrent(0,0);
pos.setCurrent(1,0);
pos.saveVisitedPoint();
lab.setVisited(pos.getCurrent(0),pos.getCurrent(1),true);
dec.goalCell(pos.getCurrent(0),pos.getCurrent(1));
eCar.turn(dec.decideAngle(pos.getCurrent(0),pos.getCurrent(1)));
eCar.waitForStop();
IplImage* picture,*picture2;
picture2=(eCar.getCamera()->getTopViewImage());
```

```
cvSaveImage("picture2.png",picture2);
while((eCar.getSensors()->readSensor(        0        )<50)        &&
dec.decideSegment()>part)
{
      eCar.drive(pos.cell);
      eCar.waitForStop();
      part=part+pos.cell;
}
pos.position(part);
lab.visitedWay();
lab.lineOfSight();
for(int i=0;i<64;i++)
      if(lab.getDirectWay(i,62))
      cout<<"von Zelle "<<i<<" gibt es einen direkten Weg zu der
Zelle 62”<<endl;
```

Ausgabe:

von Zelle 0 gibt's einen direkten Weg zu der Zelle 62
von Zelle 1 gibt's einen direkten Weg zu der Zelle 62
von Zelle 2 gibt's einen direkten Weg zu der Zelle 62
von Zelle 3 gibt's einen direkten Weg zu der Zelle 62
von Zelle 4 gibt's einen direkten Weg zu der Zelle 62
von Zelle 5 gibt's einen direkten Weg zu der Zelle 62
von Zelle 6 gibt's einen direkten Weg zu der Zelle 62
von Zelle 7 gibt's einen direkten Weg zu der Zelle 62
von Zelle 8 gibt's einen direkten Weg zu der Zelle 62
von Zelle 9 gibt's einen direkten Weg zu der Zelle 62
von Zelle 10 gibt's einen direkten Weg zu der Zelle 62
von Zelle 11 gibt's einen direkten Weg zu der Zelle 62
von Zelle 12 gibt's einen direkten Weg zu der Zelle 62
von Zelle 13 gibt's einen direkten Weg zu der Zelle 62
von Zelle 14 gibt's einen direkten Weg zu der Zelle 62
von Zelle 15 gibt's einen direkten Weg zu der Zelle 62
von Zelle 16 gibt's einen direkten Weg zu der Zelle 62
von Zelle 17 gibt's einen direkten Weg zu der Zelle 62

von Zelle 18 gibt's einen direkten Weg zu der Zelle 62
von Zelle 19 gibt's einen direkten Weg zu der Zelle 62
von Zelle 20 gibt's einen direkten Weg zu der Zelle 62
von Zelle 21 gibt's einen direkten Weg zu der Zelle 62
von Zelle 22 gibt's einen direkten Weg zu der Zelle 62
von Zelle 23 gibt's einen direkten Weg zu der Zelle 62
von Zelle 24 gibt's einen direkten Weg zu der Zelle 62
von Zelle 25 gibt's einen direkten Weg zu der Zelle 62
von Zelle 26 gibt's einen direkten Weg zu der Zelle 62
von Zelle 27 gibt's einen direkten Weg zu der Zelle 62
von Zelle 28 gibt's einen direkten Weg zu der Zelle 62
von Zelle 29 gibt's einen direkten Weg zu der Zelle 62
von Zelle 30 gibt's einen direkten Weg zu der Zelle 62
von Zelle 31 gibt's einen direkten Weg zu der Zelle 62
von Zelle 32 gibt's einen direkten Weg zu der Zelle 62
von Zelle 33 gibt's einen direkten Weg zu der Zelle 62
von Zelle 34 gibt's einen direkten Weg zu der Zelle 62
von Zelle 35 gibt's einen direkten Weg zu der Zelle 62
von Zelle 36 gibt's einen direkten Weg zu der Zelle 62
von Zelle 37 gibt's einen direkten Weg zu der Zelle 62
von Zelle 38 gibt's einen direkten Weg zu der Zelle 62
von Zelle 39 gibt's einen direkten Weg zu der Zelle 62
von Zelle 40 gibt's einen direkten Weg zu der Zelle 62
von Zelle 41 gibt's einen direkten Weg zu der Zelle 62
von Zelle 42 gibt's einen direkten Weg zu der Zelle 62
von Zelle 43 gibt's einen direkten Weg zu der Zelle 62
von Zelle 44 gibt's einen direkten Weg zu der Zelle 62
von Zelle 45 gibt's einen direkten Weg zu der Zelle 62
von Zelle 46 gibt's einen direkten Weg zu der Zelle 62
von Zelle 47 gibt's einen direkten Weg zu der Zelle 62
von Zelle 48 gibt's einen direkten Weg zu der Zelle 62
von Zelle 49 gibt's einen direkten Weg zu der Zelle 62
von Zelle 50 gibt's einen direkten Weg zu der Zelle 62
von Zelle 51 gibt's einen direkten Weg zu der Zelle 62

von Zelle 52 gibt's einen direkten Weg zu der Zelle 62
von Zelle 53 gibt's einen direkten Weg zu der Zelle 62
von Zelle 54 gibt's einen direkten Weg zu der Zelle 62
von Zelle 55 gibt's einen direkten Weg zu der Zelle 62
von Zelle 56 gibt's einen direkten Weg zu der Zelle 62
von Zelle 57 gibt's einen direkten Weg zu der Zelle 62
von Zelle 58 gibt's einen direkten Weg zu der Zelle 62
von Zelle 59 gibt's einen direkten Weg zu der Zelle 62
von Zelle 60 gibt's einen direkten Weg zu der Zelle 62
von Zelle 61 gibt's einen direkten Weg zu der Zelle 62
von Zelle 62 gibt's einen direkten Weg zu der Zelle 62
von Zelle 63 gibt's einen direkten Weg zu der Zelle 62

Ergebnis: Der Test ist erfolgreich. Am Anfang wurde die Start- und Zielposition initialisiert, und da es von jeder Zelle zu jeder anderen Zelle einen direkten Weg gibt, ist die Ausgabe richtig.

2. Fall: Es gibt ein Hindernis zwischen Start- und Zielzelle.

Eingaben: Startposition und Zielposition, sowie ein Hindernis zwischen beiden im Punkt (3,4).

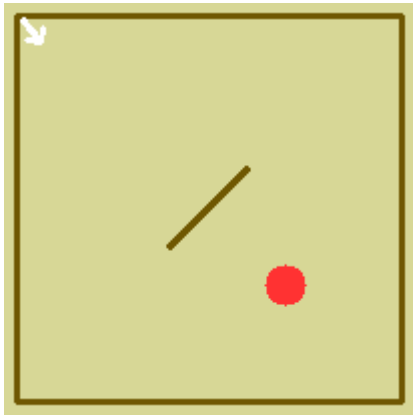

Abbildung 7: Ein Labyrinth mit Hindernis

Programmstück:

```
lab.divide(lableng,labwid,cell);
pos.setCurrent(0,0);
pos.setCurrent(1,0);
pos.saveVisitedPoint();
lab.setVisited(pos.getCurrent(0),pos.getCurrent(1),true);
dec.goalCell(pos.getCurrent(0),pos.getCurrent(1));
eCar.turn(dec.decideAngle(pos.getCurrent(0),pos.getCurrent(1)));
eCar.waitForStop();
IplImage* picture,*picture2;
picture2=(eCar.getCamera()->getTopViewImage());
cvSaveImage("picture2.png",picture2);
while((eCar.getSensors()->readSensor( 0 )<50))
{
      eCar.drive(pos.cell);
      eCar.waitForStop();
      part=part+pos.cell;
}
pos.position(part);
```

```
lab.visitedWay();
lab.lineOfSight();
for(int i=0;i<64;i++){
     if(lab.getDirectWay(i,62)){
          cout<<"von Zelle "<<i<<" gibt's einen direkten Weg zu der
Zelle 62”<<endl;
     else
          cout<<"von Zelle "<<i<<" gibt's keinen direkten Weg zu
der Zelle 62”<<endl;

     }
}
```

Ausgabe:

Das ist nur die Ausgabe für die Richtung von der Startposition zur Zielposition:

von Zelle 0 gibt's **keinen** direkten Weg zu der Zelle 62
von Zelle 1 gibt's einen direkten Weg zu der Zelle 62
von Zelle 2 gibt's einen direkten Weg zu der Zelle 62
von Zelle 3 gibt's einen direkten Weg zu der Zelle 62
von Zelle 4 gibt's einen direkten Weg zu der Zelle 62
von Zelle 5 gibt's einen direkten Weg zu der Zelle 62
von Zelle 6 gibt's einen direkten Weg zu der Zelle 62
von Zelle 7 gibt's einen direkten Weg zu der Zelle 62
von Zelle 8 gibt's **keinen** direkten Weg zu der Zelle 62
von Zelle 9 gibt's **keinen** direkten Weg zu der Zelle 62
von Zelle 10 gibt's einen direkten Weg zu der Zelle 62
von Zelle 11 gibt's einen direkten Weg zu der Zelle 62
von Zelle 12 gibt's einen direkten Weg zu der Zelle 62
von Zelle 13 gibt's einen direkten Weg zu der Zelle 62
von Zelle 14 gibt's einen direkten Weg zu der Zelle 62
von Zelle 15 gibt's einen direkten Weg zu der Zelle 62
von Zelle 16 gibt's einen direkten Weg zu der Zelle 62
von Zelle 17 gibt's **keinen** direkten Weg zu der Zelle 62
von Zelle 18 gibt's **keinen** direkten Weg zu der Zelle 62

von Zelle 19 gibt's einen direkten Weg zu der Zelle 62
von Zelle 20 gibt's einen direkten Weg zu der Zelle 62
von Zelle 21 gibt's einen direkten Weg zu der Zelle 62
von Zelle 22 gibt's einen direkten Weg zu der Zelle 62
von Zelle 23 gibt's einen direkten Weg zu der Zelle 62
von Zelle 24 gibt's einen direkten Weg zu der Zelle 62
von Zelle 25 gibt's einen direkten Weg zu der Zelle 62
von Zelle 26 gibt's einen direkten Weg zu der Zelle 62
von Zelle 27 gibt's einen direkten Weg zu der Zelle 62
von Zelle 28 gibt's einen direkten Weg zu der Zelle 62
von Zelle 29 gibt's einen direkten Weg zu der Zelle 62
von Zelle 30 gibt's einen direkten Weg zu der Zelle 62
von Zelle 31 gibt's einen direkten Weg zu der Zelle 62
von Zelle 32 gibt's einen direkten Weg zu der Zelle 62
von Zelle 33 gibt's einen direkten Weg zu der Zelle 62
von Zelle 34 gibt's einen direkten Weg zu der Zelle 62
von Zelle 35 gibt's einen direkten Weg zu der Zelle 62
von Zelle 36 gibt's einen direkten Weg zu der Zelle 62
von Zelle 37 gibt's einen direkten Weg zu der Zelle 62
von Zelle 38 gibt's einen direkten Weg zu der Zelle 62
von Zelle 39 gibt's einen direkten Weg zu der Zelle 62
von Zelle 40 gibt's einen direkten Weg zu der Zelle 62
von Zelle 41 gibt's einen direkten Weg zu der Zelle 62
von Zelle 42 gibt's einen direkten Weg zu der Zelle 62
von Zelle 43 gibt's einen direkten Weg zu der Zelle 62
von Zelle 44 gibt's einen direkten Weg zu der Zelle 62
von Zelle 45 gibt's einen direkten Weg zu der Zelle 62
von Zelle 46 gibt's einen direkten Weg zu der Zelle 62
von Zelle 47 gibt's einen direkten Weg zu der Zelle 62
von Zelle 48 gibt's einen direkten Weg zu der Zelle 62
von Zelle 49 gibt's einen direkten Weg zu der Zelle 62
von Zelle 50 gibt's einen direkten Weg zu der Zelle 62
von Zelle 51 gibt's einen direkten Weg zu der Zelle 62
von Zelle 52 gibt's einen direkten Weg zu der Zelle 62

von Zelle 53 gibt's einen direkten Weg zu der Zelle 62
von Zelle 54 gibt's einen direkten Weg zu der Zelle 62
von Zelle 55 gibt's einen direkten Weg zu der Zelle 62
von Zelle 56 gibt's einen direkten Weg zu der Zelle 62
von Zelle 57 gibt's einen direkten Weg zu der Zelle 62
von Zelle 58 gibt's einen direkten Weg zu der Zelle 62
von Zelle 59 gibt's einen direkten Weg zu der Zelle 62
von Zelle 60 gibt's einen direkten Weg zu der Zelle 62
von Zelle 61 gibt's einen direkten Weg zu der Zelle 62
von Zelle 62 gibt's einen direkten Weg zu der Zelle 62
von Zelle 63 gibt's einen direkten Weg zu der Zelle 62

Diese Ausgabe zeigt wie `lineOfSight` funktioniert und was es speichert:

directWay[0][26] 0 directWay[26][0] 0
directWay[0][34] 0 directWay[34][0] 0
directWay[0][35] 0 directWay[35][0] 0
directWay[0][43] 0 directWay[43][0] 0
directWay[0][44] 0 directWay[44][0] 0
directWay[0][52] 0 directWay[52][0] 0
directWay[0][53] 0 directWay[53][0] 0
directWay[0][61] 0 directWay[61][0] 0
directWay[0][62] 0 directWay[62][0] 0
directWay[8][26] 0 directWay[26][8] 0
directWay[8][34] 0 directWay[34][8] 0
directWay[8][35] 0 directWay[35][8] 0
directWay[8][43] 0 directWay[43][8] 0
directWay[8][44] 0 directWay[44][8] 0
directWay[8][52] 0 directWay[52][8] 0
directWay[8][53] 0 directWay[53][8] 0
directWay[8][61] 0 directWay[61][8] 0
directWay[8][62] 0 directWay[62][8] 0
directWay[9][26] 0 directWay[26][9] 0
directWay[9][34] 0 directWay[34][9] 0

directWay[9][35] 0 directWay[35][9] 0
directWay[9][43] 0 directWay[43][9] 0
directWay[9][44] 0 directWay[44][9] 0
directWay[9][52] 0 directWay[52][9] 0
directWay[9][53] 0 directWay[53][9] 0
directWay[9][61] 0 directWay[61][9] 0
directWay[9][62] 0 directWay[62][9] 0
directWay[17][26] 0 directWay[26][17] 0
directWay[17][34] 0 directWay[34][17] 0
directWay[17][35] 0 directWay[35][17] 0
directWay[17][43] 0 directWay[43][17] 0
directWay[17][44] 0 directWay[44][17] 0
directWay[17][52] 0 directWay[52][17] 0
directWay[17][53] 0 directWay[53][17] 0
directWay[17][61] 0 directWay[61][17] 0
directWay[17][62] 0 directWay[62][17] 0
directWay[18][26] 0 directWay[26][18] 0
directWay[18][34] 0 directWay[34][18] 0
directWay[18][35] 0 directWay[35][18] 0
directWay[18][43] 0 directWay[43][18] 0
directWay[18][44] 0 directWay[44][18] 0
directWay[18][52] 0 directWay[52][18] 0
directWay[18][53] 0 directWay[53][18] 0
directWay[18][61] 0 directWay[61][18] 0
directWay[18][62] 0 directWay[62][18] 0

* 0 steht für `false`. Also gibt es keinen direkten Weg.

3.2.5 /T213/ angle

Beschreibung:
Diese Methode berechnet den Arkustangens zwischen dem letzten Punkt und dem aktuellen. Am Anfang ist `angle` gleich 0.

1. Fall: Wir haben zwei Punkte: aktuelle Position: (0,0) und Zielposition: (6,7))

Eingaben:

S 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 **Z** 0

Programmstück:

```
Labyrinth lab;
cout<<lab.angle(0,0,6,7)<<endl;
```

Ausgabe: 0.86217

Ergebnis: Der Test ist erfolgreich. Der Winkel wird im Bogenmaß zurückgegeben und beträgt ca. 49°.

2. Fall: Wir haben zwei Punkte: aktuelle Position: (6,7) und Zielposition: (0,0)

Eingaben:

Z 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 **S** 0

Programmstück:

```
Labyrinth lab;
cout<<lab.angle(6,7,0,0)<<endl;
```

Ausgabe: -2.27942

Ergebnis: Der Test ist erfolgreich. Der Winkel wird im Bogenmaß zurückgegeben und beträgt in Gegenrichtung ca. -131°.

3.2.6 /T214/ mapExplored

Beschreibung:

Diese Methode setzt `search` auf den boolschen Wert `true` und gibt es als Rückgabewert zurück, wenn das Labyrinth nicht komplett abgesucht worden ist. Andernfalls wird `search false` zugewiesen und die Suche in der main-Methode beendet.

1.Fall:

Labyrinth ist vollständig abgesucht worden. `mapExplored` setzt die Attribute von `search` auf `false` und gibt diese zurück.

Eingaben: Ein vollständig abgesuchtes Labyrinth.

1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1

Programmstück:

```
bool Labyrinth:: mapExplored()
{
for(int i=0;i<8;i++)
           for(int j=0;j<8;j++)
                lab.setVisited(i,j,true);
for(int i=0;i<8;i++)
{
     for(int j=0;j<8;j++)
          cout<<lab.getVisited(i,j)<<"  ";
     cout<<endl;
```

```
}
cout<<lab.mapExplored()<<endl;
return search;
```

Ausgabe: 0. In der Funktion `mapExplored` geben wir als Rückgabewert für `search true` zurück, wenn es noch Zelle(n) gibt, die unbesucht ist(sind). Die 0 steht hier dafür, dass alle Zellen schon besucht wurden, damit `search false` (0) zurück geben kann.

Ergebnis: Der Test ist erfolgreich.

2.Fall:

Labyrinth hat noch unbesuchte Zellen. `mapExplored` muss die Attribute `search` auf `true` setzen und zurückgeben.

Eingaben: Ein Labyrinth, das zwei unbesuchte Zellen ((4,0) und (1,7)) hat.

```
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
```

Programmstück:

```
for(int i=0;i<8;i++)
           for(int j=0;j<8;j++)
                lab.setVisited(i,j,true);
lab.setVisited(4,0,false);
lab.setVisited(1,7,false);
for(int i=0;i<8;i++)
{
     for(int j=0;j<8;j++)
          cout<<lab.getVisited(i,j)<<"  ";
     cout<<endl;
}
cout<<lab.mapExplored()<<endl;
```

Ausgabe: 1

Ergebnis: Der Test ist erfolgreich.

3.3 Testfall /T3/: Fahrstrategie

3.3.1 /T300/: goalCell

Es wird angenommen, dass die Suche beginnt, d.h. alle Zellen sind unbesucht und dem Softwaresystem ist noch nicht bekannt wo sich die Hindernisse im Labyrinth befinden. Das Labyrinth ist in diesem Fall 3m x 3m groß und in 0.25m x 0.25m Zellen unterteilt `divide(300, 300, 25)`, d.h. das Labyrinth ist virtuell in 12 x 12 Zellen unterteilt. Es wird ebenfalls angenommen, dass die Startposition der Zelle (0,0) entspricht. Um die Methode `goalCell(int x, int y)` zu testen, muss nur `goalCell(0,0)` in der main-Methode eingeben werden. Die Methode markiert eine Zelle genau dann als Ziel (goal), wenn der Weg, der die zwei Punkte verbindet, die größte Anzahl von unbesuchten Zellen enthält und auf diesem Weg kein Hindernis liegt.
Am Anfang sind alle Zellen unbesucht und keine Hindernisse gespeichert.

Fall 1: Da sich das Fahrzeug in der Zelle (0,0) befindet, ist der längste Weg von dem Startpunkt die Zelle (11,11).

Eingabe: `goalCell(0,0)`

Ausgabe: goalCell: (10,11)

Das Ergebnis ist die Zelle (10,11) - eine Zelle neben dem Punkt (11,11) - weil beide Wege die gleiche Anzahl unbesuchter Zellen enthalten.

Fall 2: Mit der Methode `setVisited(int a, int b, bool c)` setze die Zellen das obere Dreieck auf `visited`, das neue Ziel von (0,0) ist (11,8).

Eingabe: `goalCell(0,0)`

Ausgabe: Goal(0,0): (11,8)

S	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	**Z**	0	0	1

Legende:

1: besuchte Zellen 0: unbesuchte Zellen

S: Startpunkt Z: Ziel

Fall 3:

Die gleichen Bedingungen wie in Fall 2 aber mit Startpunkt (1,0)

Eingabe: `goalCell(1,0)`

Ausgabe: Goal(1,0): (11,9). Das sind alle Nullen, die das obere Dreieck von unten begrenzen.

1	1	1	1	1	1	1	1	1	1	1
S	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1
0	0	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	1	1

0 0 0 0 0 0 0 0 0 **Z** 0 1

3.3.2 /T301/: decideAngle

Die Methode `decideAngle(int x, int y)` berechnet den Winkel, um den sich das Fahrzeug bei einer Richtungsänderung drehen muss.

Das Fahrzeug befindet sich in der Zelle (2,2) [Punkt 1 in Abbildung 8] mit dem Winkel null. Die Methode `goalCell()` bestimmt, dass die Zelle (8,9) das Ziel ist [Punkt 2 in Abbildung 8]. Der Roboter dreht sich um den Winkel von ca. 50° (beta) und fährt bis Zelle (7,8) (hat eine Wand als Hindernis erkannt). Das neue Ziel ist (0,5) [Punkt 3 in Abbildung 8]. Der Winkel von (7,8) zur Zelle (0,5) beträgt ca. -156° (alpha). Der zu drehende Winkel ist die Differenz der beiden Winkeln, also -206°(beta-alpha).

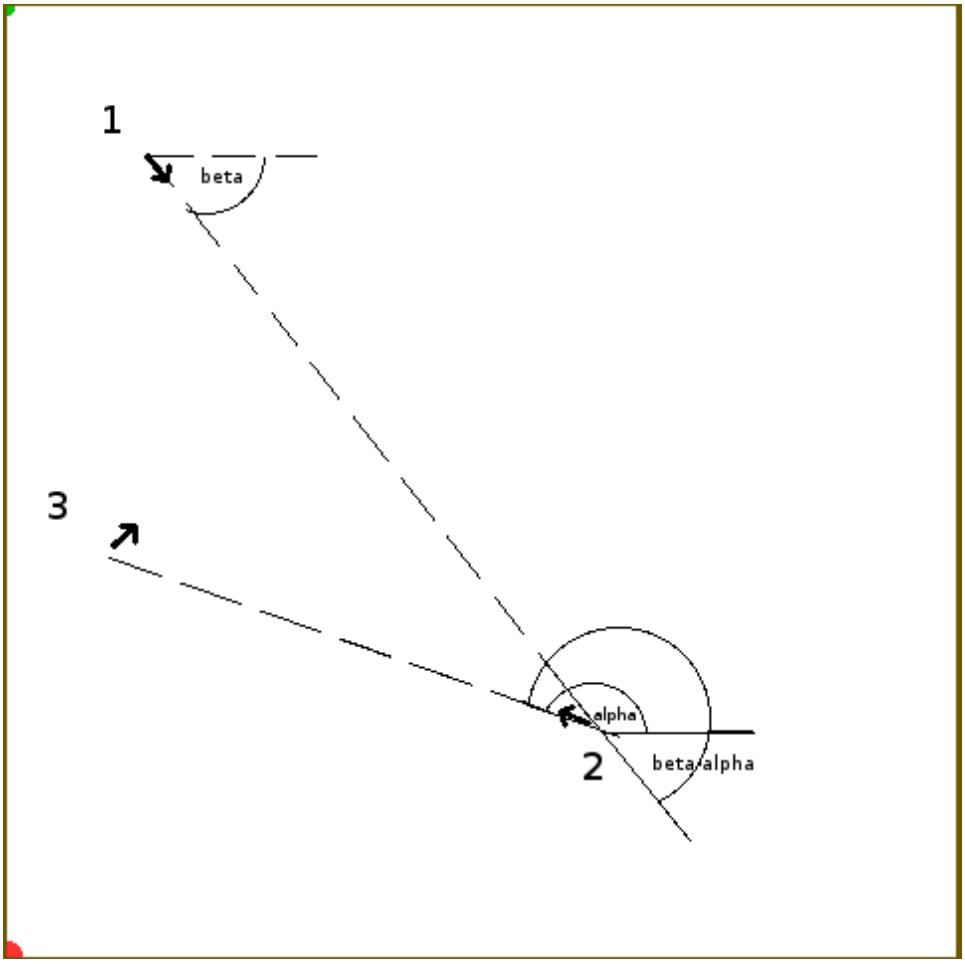

Abbildung 8 : Bestimmung des Winkels mit der Methode decideAngle

Ausgabe:

```
old angle:  0
new angle:  49.3987
turn:  49.3987
Position: (2, 2)
Goal  (8, 9)
old angle:  50.1944
new angle:  -156.801
turn:  -206.996
Position: (7, 8)
Goal  (0, 5)
```

3.3.3 /T302/: decideSegment

Die Methode `decideSegment()` berechnet die Länge der Strecke zwischen zwei Punkten, dem Standort und dem Ziel. Der Roboter fährt anschließend diesen Weg.

Der Abstand wird wie folgt berechnet:

Standort :(xS, yS)
Ziel : (xZ, yZ)
Der Abstand ist gleich der Wurzel aus $(xS-xZ)^2 + (yS-yZ)^2$.
Das Labyrinth ist virtuell in 0.25m x 0.25m Zellen geteilt, daher wird der berechnete Abstand mit 0.25m multiplizieren.

Abstand= [sqrt((xS-xZ)*(xS-xZ) + (yS-yZ)*(yS-yZ))]*0.25

Anhand des folgenden Beispiels wird die Methode getestet:

Der Roboter fährt mehrere Punkte ab. Angefangen wird bei der Zelle (0,0) über diverse andere (siehe Ausgabe) und endet bei (1,7).
Die Ausgabe verdeutlicht den Standort (Position), Ziel (Goal) und den Abstand zwischen den beiden Punkten (Segment Length).

Ausgabe:

```
Position: (0 , 0)
  Goal  (8, 9)
  Length:  3.0104 m

  Position: (6 , 7)
  Goal  (9, 0)
  Length:  1.90394 m

  Position: (6 , 6)
```

Goal (9, 1)
Length: 1.45774 m

Position: (8 , 3)
Goal (0, 9)
Length: 2.5 m

Position: (7 , 4)
Goal (0, 9)
Length: 2.15058 m

Position: (2, 8)
Goal (9, 0)
Length: 2.65754 m

Position: (6, 3)
Goal (1, 0)
Length: 1.45774 m

Position: (3, 1)
Goal (0, 9)
Length: 2.136 m

Position: (1, 7)
Goal (9, 0)
Length: 2.65754 m

3.3.4 /F305/: isNew

Die Methode `isNew` überprüft ob die Zellen, die sich zwischen zwei anderen Zellen befinden bereits besucht worden sind oder nicht. Ist eine Zelle unbesucht, wird der Zähler um eins erhöht.

Fall1:
Alle Zellen sind unbesucht.
Zum Testen wie viele unbesuchte Zellen zwischen (0,0) und (6,7) existieren, wird in der Ausgabe die Anzahl und die Koordinaten der unbesuchten Zellen zu finden sein.

Eingabe: `dec.isNew(0,0,6,7)`

Ausgabe:

Zwischen (0,0) und (6,7): **13** unbesuchte Zellen. Das sind:

(0,0)
(0,1)
(1,1)
(1,2)
(2,2)
(2,3)
(3,3)
(3,4)
(4,4)
(4,5)
(5,5)
(5,6)
(6,7)

Die betroffenen Zellen werden in [Abbildung 2] schwarz markiert. (S: Startzelle, Z: Ziel).

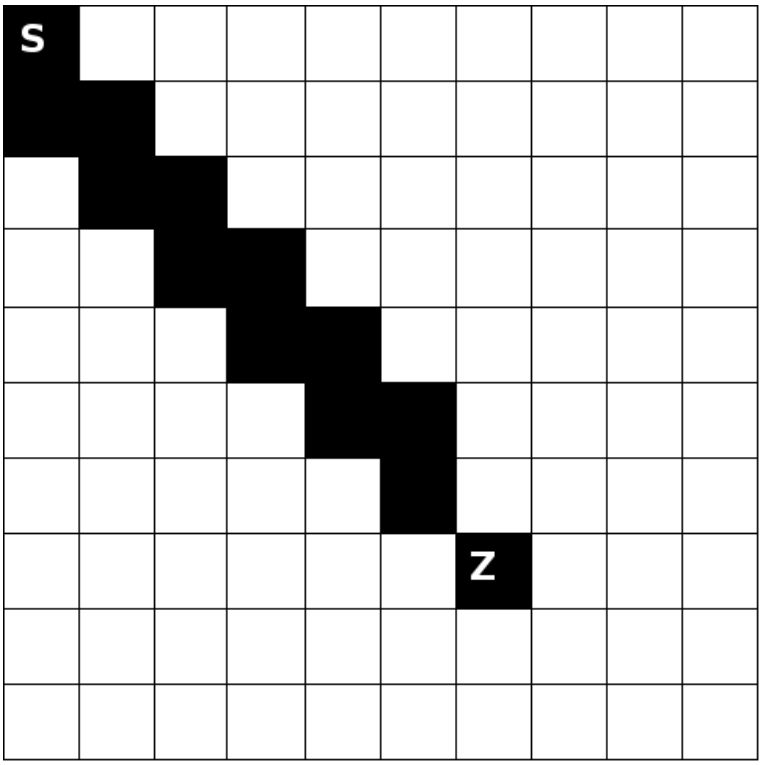

Abbildung 9: Die unbesuchten Zellen zwischen (0,0) und (6,7) im 1.Fall

Fall 2: Setze für die gleiche Start- und Zielzellen die Zellen (2,2), (3,3) und (4,4) auf `visited=true`.

Eingabe:

```
lab.setVisited(2,2,true); // setze die Zellen auf visited=true
lab.setVisited(3,3,true); // setze die Zellen auf visited=true
lab.setVisited(4,4,true); // setze die Zellen auf visited=true
dec.isNew(0,0,6,7);
```

Ausgabe:

Zwischen (0,0) und (6,7): **10** unbesuchte Zellen. Das sind:

(0,0)
(0,1)
(1,1)
(1,2)
(2,3)
(3,4)
(4,5)
(5,5)
(5,6)
(6,7)

Die betroffene Zellen werden in [Abbildung 3] in schwarz markiert. (S: Startzelle, Z: Ziel).

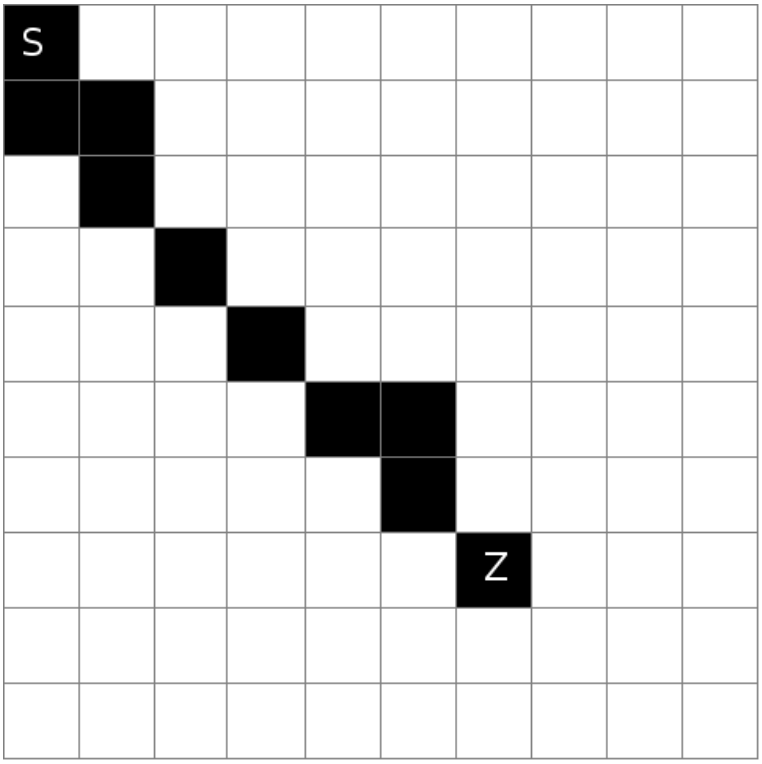

Abbildung 3: Die unbesuchte Zellen zwischen (0,0) und (6,7) im 2.Fall

3.4 Testfall /T400/: Gesamttest

Test 1:

Testumgebung: Simulator (Simulated Roboter)

Labyrinthgröße 5m x 5m
Startposition: (0,0)
Ballposition: (16,10) [3,7m, 2,5m]

Hindernisse: siehe Abbildung 11.

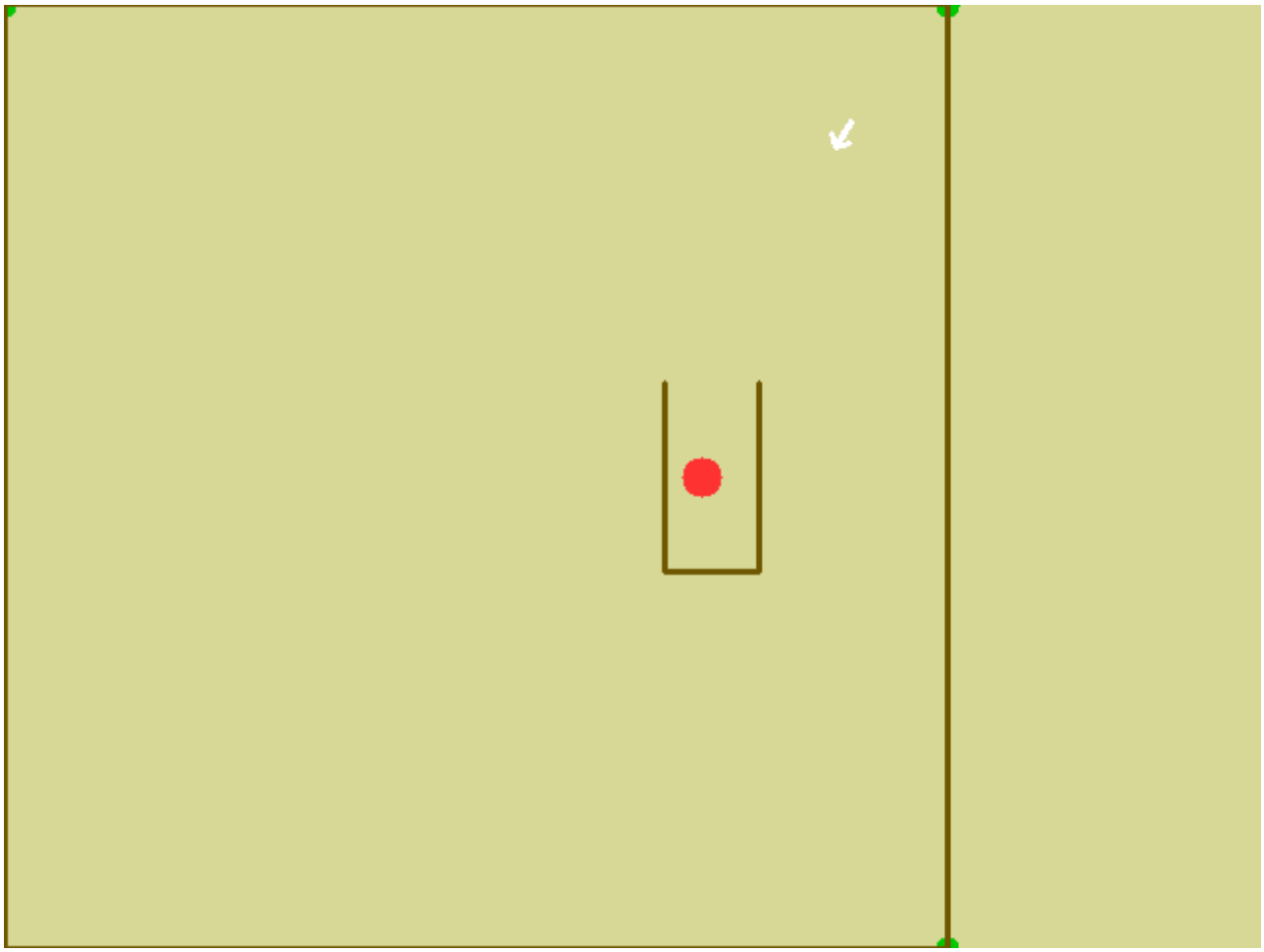

Abbildung 11: Labyrinth- Test 1

Ergebnis: Der Ball wird gefunden.

Test 2:

Testumgebung: Simulator (Simulated Roboter)

Labyrinthgrösse 5m x 5m
Startposition: (0,0)
Ballposition:(14,14) [3,3m, 3,3m]
Hindernisse: siehe Abbildung 12

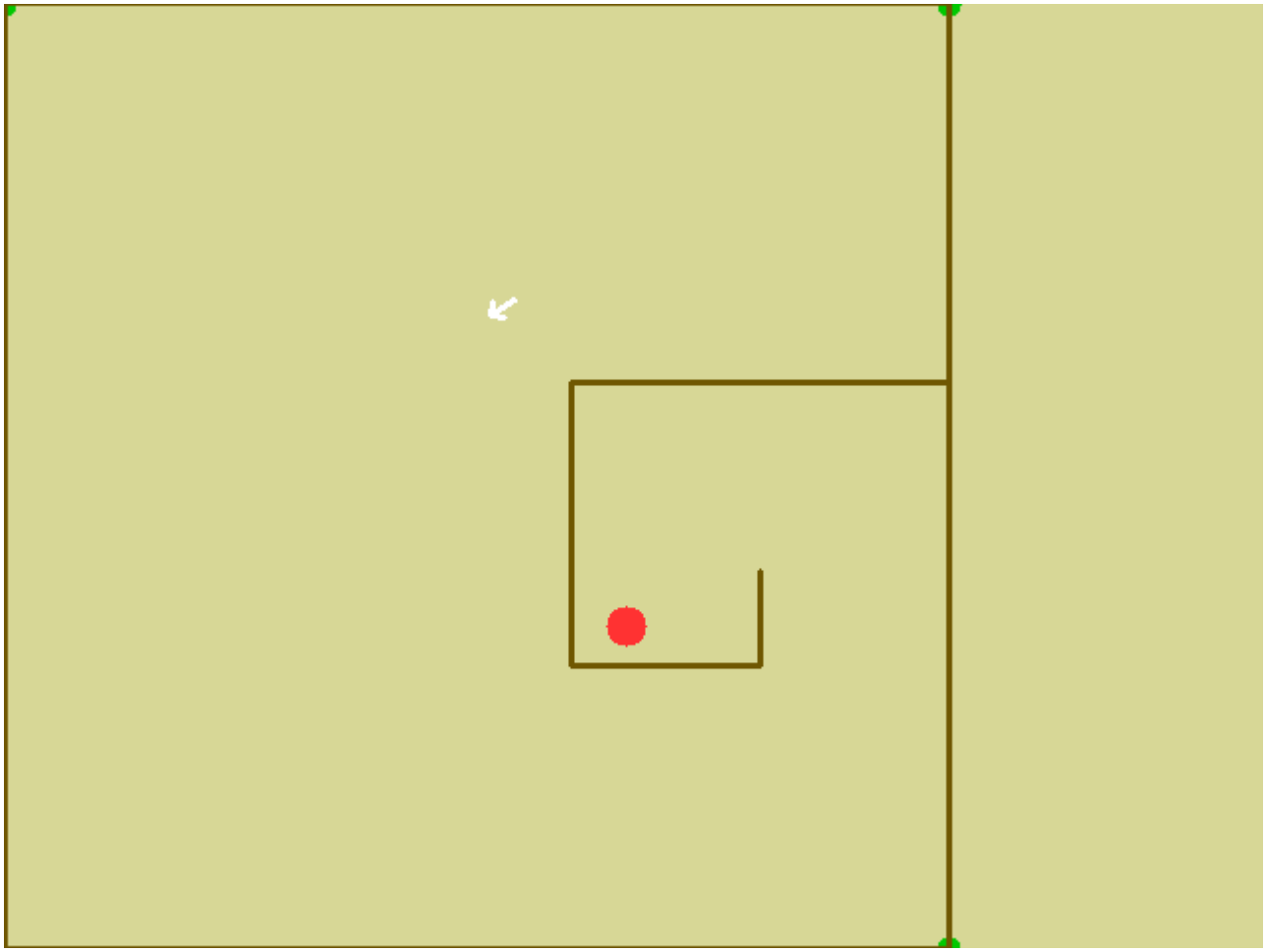

Abbildung 12: Labyrinth- Test 2

Ergebnis: Der Ball wird gefunden.

Test 3:

Testumgebung: Simulator (Simulated Roboter)

Labyrinthgröße 5m x 5m
Startposition: (0,0)
Ballposition: (14,14) [3,3m, 3,3m]
Hindernisse: siehe Abbildung 13

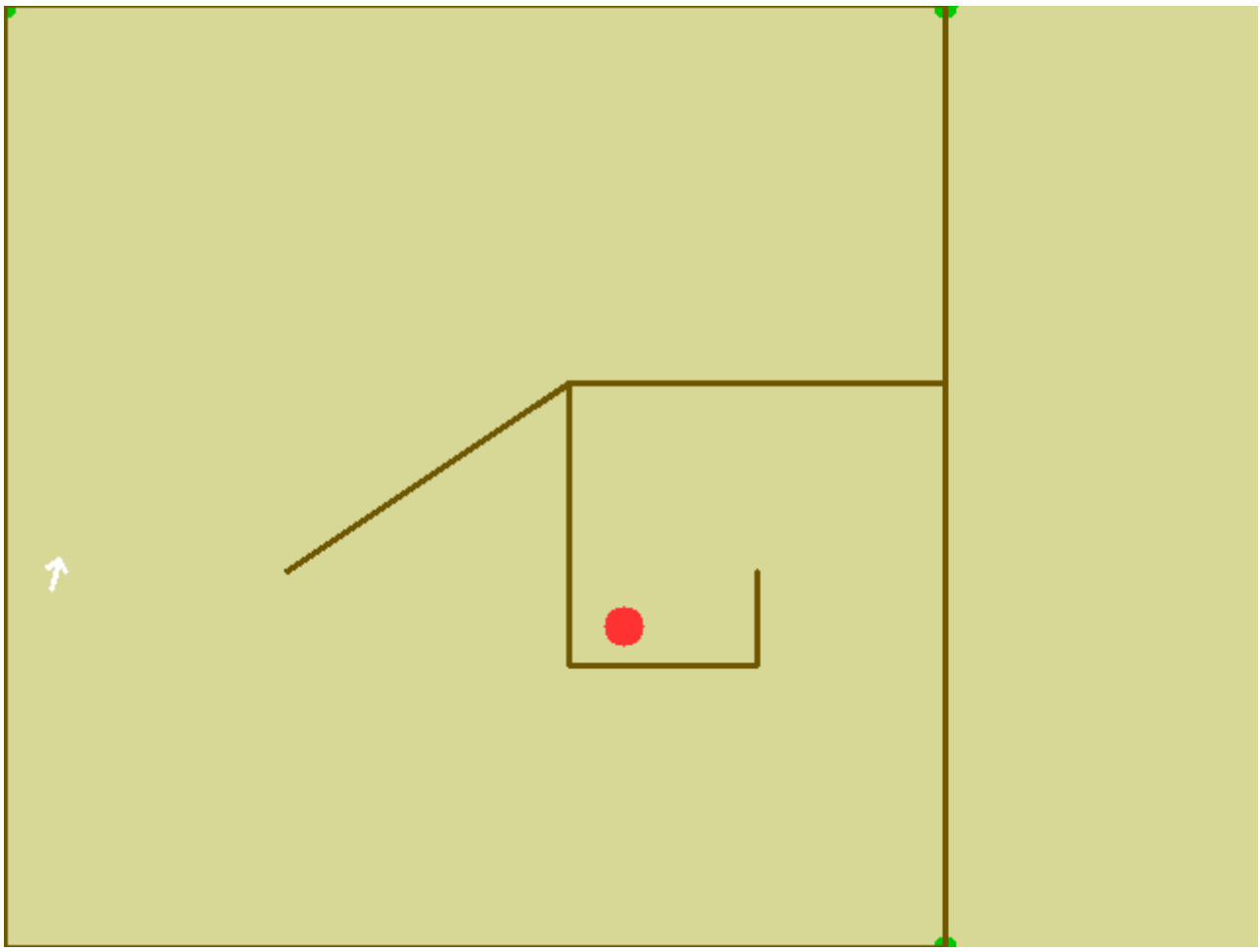

Abbildung 13: Labyrinth- Test 3

Ergebnis: Der Ball wird gefunden.

Test 4:

Testumgebung: Simulator (Simulated Roboter)

Labyrinthgrösse 5m x 5m
Startposition: (0,0)
Ballposition: (9,7) [2,3m, 1,8m]
Hindernisse: siehe Abbildung 14

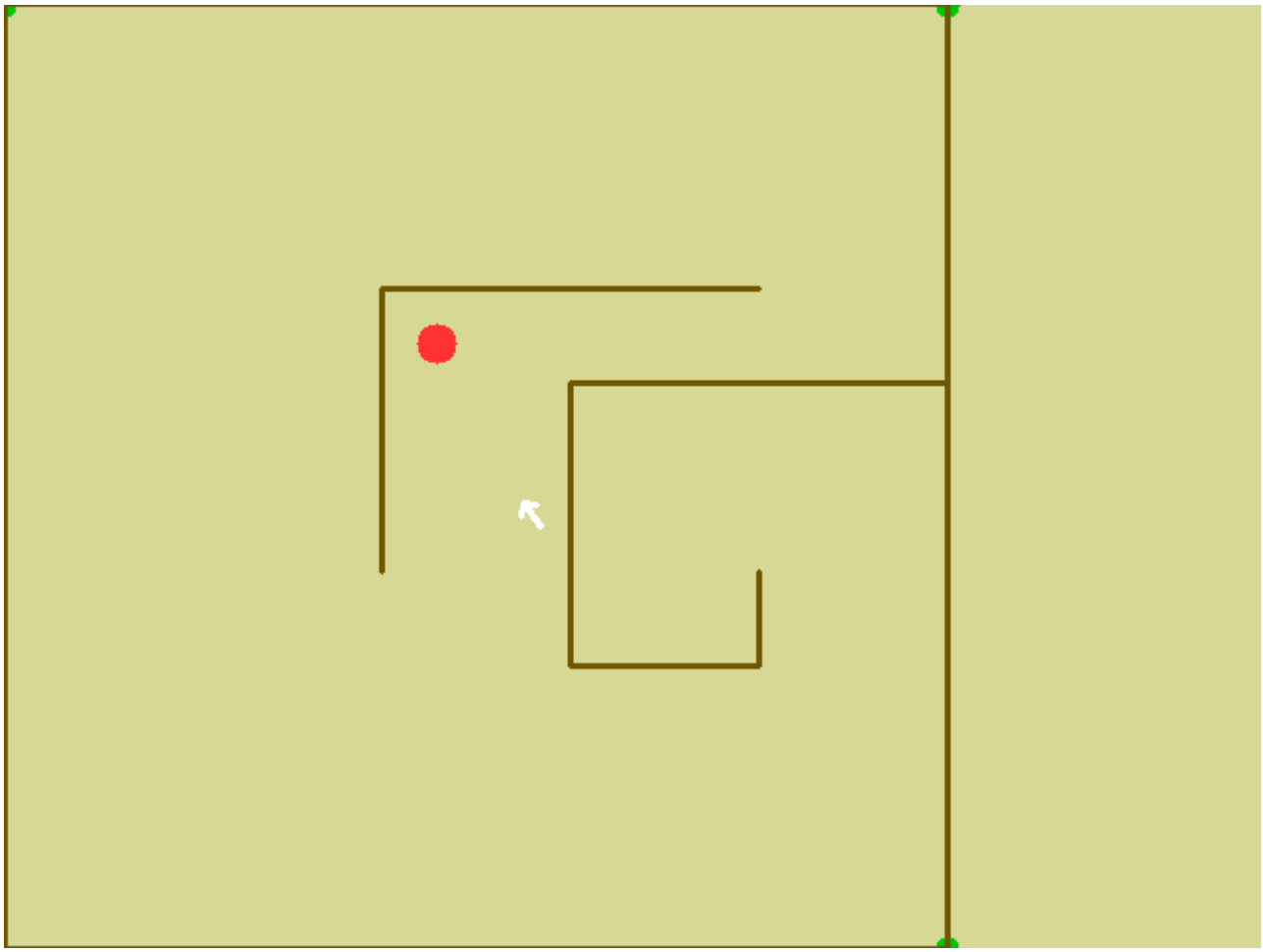

Abbildung 14: Labyrinth- Test 4

Ergebnis: Der Ball wird gefunden.

Test 5:

Testumgebung: Simulator (Simulated Roboter)

Labyrinthgrösse 5m x 5m
Startposition: (0,0)
Ballposition: (9,12) [2,3m, 2,8m]
Hindernisse: siehe Abbildung 15

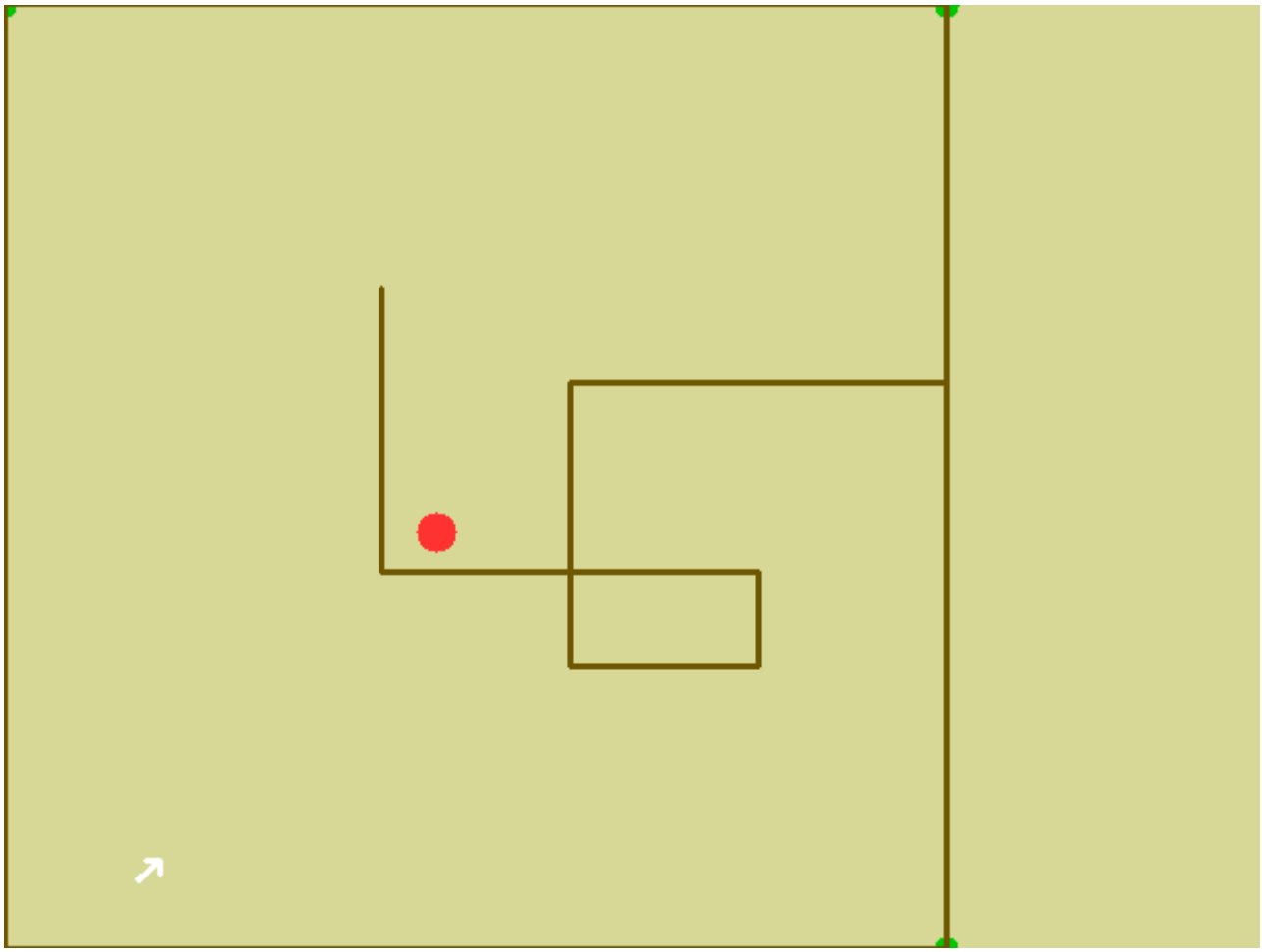

Abbildung 15: Labyrinth- Test 5

Ergebnis: Der Ball wird gefunden.

4 Zusammenfassung

Alles in Allem ist das Softwaresystem ECAR funktionsfähig und bereit zur Übergabe an den Auftraggeber. Das bestellte Produkt weist die notwendigen Attribute und Fähigkeiten auf, die verlangt werden, um die Lösung einer Suchen-und-Finden-Problematik zu liefern. Ausschließlich der Aspekt der Positionsbestimmung durch Kreuzpeilung anhand von aufgestellten Baken wurde nicht in geplanter Form umgesetzt. Die Gründe, die hierfür anzuführen sind, sind Zeitmangel und die sich gebotene Chance eine unkompliziertere Methode anzuwenden, die im Großen und Ganzen eine zeitlich effektivere Suche ermöglicht. Die Softwaretests verdeutlichen, dass die Ballerkennung bis zu einer Entfernung von 1,5 Metern sicher, fehlerlos und verlässlich abläuft. Lediglich bei größeren Distanzen und bei unerwünschten kreisförmigen Objekten in der Umgebung kann es zu einer - im ersten Fall – Nichterkennung oder - wie im zu letzt genannten - zu einer fehlerhaften Erkennung führen. Schwerwiegende negative Konsequenzen ergeben sich daraus nicht, denn eine Ballerkennung bis zu 1,5 Metern ist völlig ausreichend für eine erfolgreiche Suchprozedur und irritierende kreisförmige Objekte, die zur Identifikation falscher Objekte führen können, befinden sich nicht in dem tatsächlichen Einsatzgebiet des Roboters.