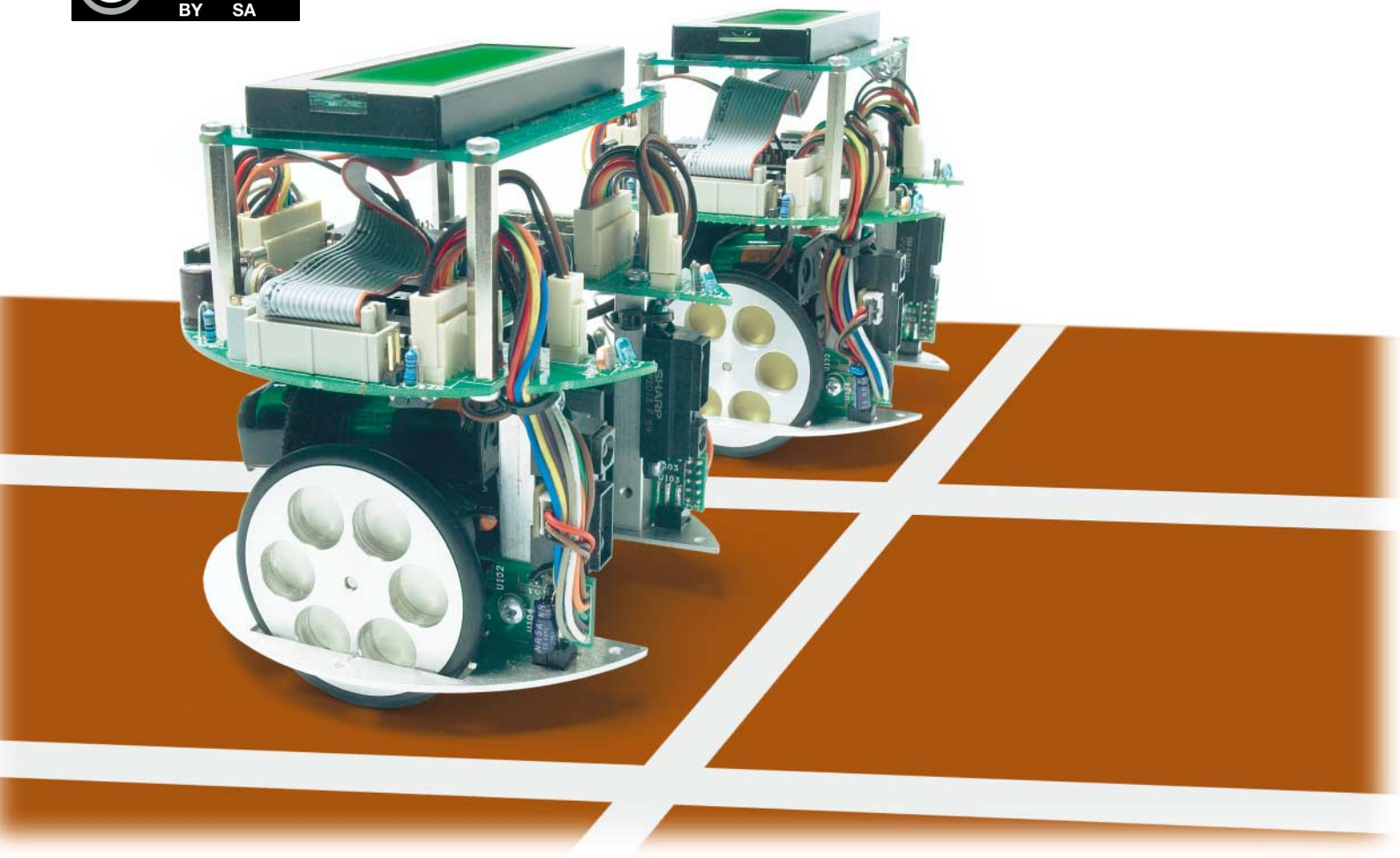


Benjamin Benz, Peter König

Hase und Igel

Roboterwettbewerb im Simulator c't-Sim

Vor einem halben Jahr hoben wir den c't-Bot aus der Taufe, einen Roboter, den man selbst aufbaut und programmiert. Jetzt schreibt die Redaktion das erste virtuelle Bot-Turnier aus: Wer dort siegen will, muss einen flotten Weg durch unbekanntes Gelände finden können. Einen realen c't-Bot braucht man dafür nicht – den gibt es zu gewinnen.

Zwei Rivalen stehen in den Startboxen. Vor ihnen liegt eine schwierige Aufgabe: Um in die nächste Runde zu kommen, muss der eine den unbekannten Irrgarten schneller durchqueren als der andere. Die Zuschauermenge tobt – vor den Bildschirmen. Der Wettkampf-Server gibt das Rennen frei. Fans, Bot-Eigner und Schiedsrichter verfolgen das Geschehen live übers Web. Solche Szenen dürften sich in der Woche vom 16. bis 20. Oktober tatsächlich abspielen: Wir veranstalten im Rahmen des c't-Bot-Projektes einen Programmierwettbewerb. Virtuelle Roboter sollen um die Wette in der Simulationsumgebung c't-Sim Labyrinthe durchqueren. Dabei liegt die Herausforderung nicht im Motor-Tuning oder darin, die stärksten Akkus an den Start zu bringen – gefragt sind vielmehr clevere Strategien.

Da es sich um einen virtuellen Wettbewerb handelt, befindet sich auch der Austragungsort im Cyberspace. Den gesamten Wettbewerb wickeln die c't-Server ab, die Darstellung übernimmt eine Webseite auf heise online. Die stellt Rennstrecke und Bots voraussichtlich in stark vereinfachter Aufsicht dar – die gewohnte 3D-Grafik des c't-Sim verursacht bei den erwarteten Zuschauerzahlen mehr Netzlast, als unsere Server vertragen können.

Dabei sein ist alles

Am Wettbewerb dürfen sowohl Einzelkämpfer als auch Teams teilnehmen, wobei jedes Team nur einen virtuellen Bot an den Start schicken darf. Wer sich der Konkurrenz stellen will, muss sich bis zum 8. September auf der Webseite zum Wettbewerb [1] registrieren. Dort stehen auch die Spielregeln und Wettbewerbsbedingungen zur Verfügung, diese akzeptiert man bei der Anmeldung per Checkbox. Fertigen Robotercode braucht man zu diesem Zeitpunkt noch nicht – er muss erst bis zum 6. Oktober bei uns eingehen.

Unser virtueller Roboter fährt auf zwei Rädern und einem Gleitpin. Für Orientierung im Raum sorgen je zwei nach vorne blickende Distanz- und Lichtfühler. Das Terrain unter dem Bot behalten Liniensensoren im Auge, Abgrundmelder warnen vor drohendem Absturz. Ein optischer Maussensor in der Grundplatte registriert jede Bewegung; zusätzlich überwacht der Bot die Drehung seiner Räder. Als Steuercode nimmt der Roboter C-Programme entgegen, Ansteuerungsroutinen für die Sensoren und Aktuatoren stellen wir als fertiges Framework auf unserer

Projektseite [2] zur Verfügung. Dort findet man auch den in Java geschriebenen Simulator c't-Sim, der Trainingsläufe abwickelt. Alle notwendigen Werkzeuge für eine Teilnahme am Roboterwettbewerb bis hin zur empfohlenen Entwicklungsumgebung Eclipse sind kostenlos erhältlich. Ihre Installation und Konfiguration beschreiben ausführliche Anleitungen auf der Projektseite zum c't-Bot.

Unser Programmierwettbewerb richtet sich besonders auch an jene Roboter-Enthusiasten, die bisher noch keinen eigenen Roboter zusammengelötet haben. Daher winken als Preise für die Teams auf den ersten drei Plätzen je ein c't-Bot-Bausatz in der höchsten Ausbaustufe, die zum Zeitpunkt des Finales erhältlich ist: komplett mit Display und Fernbedienung, wahrscheinlich mit Verschlussklappe vor dem Transportfach und eventuell bereits mit Funkmodul. Aus der Masse ihrer Artgenossen stechen die drei Preis-Bots durch ein besonderes Äußeres hervor – mehr wird noch nicht verraten.

Termine

bis 8. September:	Anmeldung der Teilnehmer und Teams
bis 6. Oktober:	Annahme von Code für Wettbewerbsbots
16. bis 20. Oktober:	Wettbewerb
20. Oktober:	Finale

Minotaurus lässt grüßen

Die Wettbewerbskurse bauen auf ein festes Quadratraster auf, dessen Felder Mauern oder Fallgruben enthalten können [3]. Solche Hindernisse erkennt der simulierte Bot problemlos mit seinen Distanz- und Abgrundsensoren. Eine durchgehende Mauer umgibt die Turnier-Parcours. Die einzige Unterbrechung bilden die grünen Zielfelder in der Mitte der oberen Wand. Die Startfelder an der unteren Wand tragen ebenfalls Codefarben: Der rote Startblock befindet sich an der linken, der blaue an der rechten Hälfte der Wand. Da der Bot mit Hilfe seiner Liniensensoren Bodenfarben unterscheiden kann, identifiziert er diese neuralgischen Punkte des Parcours eindeutig. So kann er bestimmen, ob er aus der rechten oder der linken Position heraus startet. Ein helleres Feld vor der Startposition bestimmt die Ausrichtung des Bots bei Beginn des Rennens.

Die Kantenlänge eines Grundraster-Quadrats beträgt das Zweifache des Bot-Durchmessers. Jeder Durchgang der Wettbewerbslabyrinthe ist mindestens zwei Felder breit. Das gilt unabhängig davon, ob die Hindernisse am Rand des befahrbaren Feldes aus Mauern oder Fallgruben bestehen. Die Bots können sich damit den Weg ins Ziel nicht gegenseitig blockieren und finden auch sonst genügend Raum zum Manövrieren vor. Die Distanzsensoren liefern unterhalb von etwa zehn Zentimetern keine zuverlässigen Werte mehr, daher sollten Bots auf der Rennstrecke ausreichend Abstand von Wänden halten.

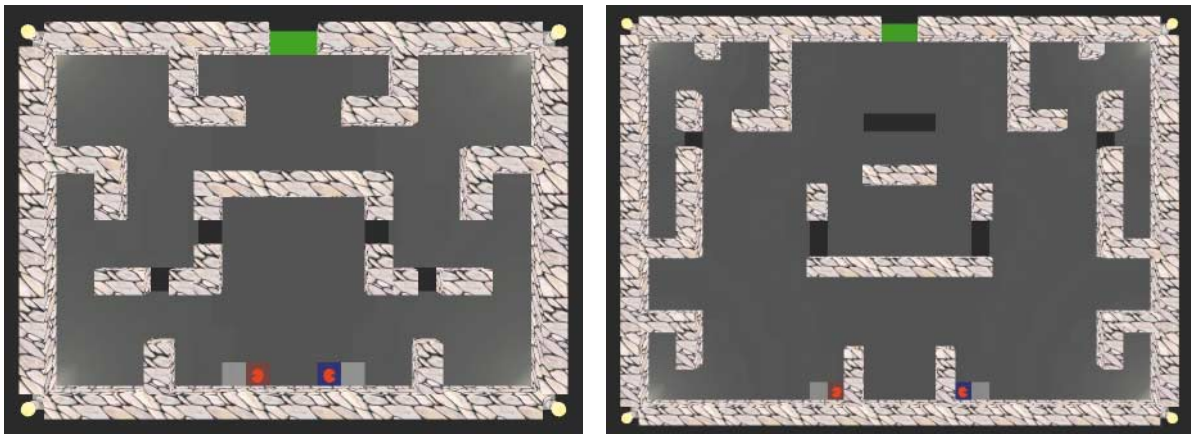
Die Größe des Labyrinths variiert im Wettbewerb von Runde zu Runde. Vier beleuchtete Säulen, die wie Leuchttürme zur Orientierung dienen können, markieren die Ecken. Der Bot kann die Lampen mit Hilfe seiner zwei Lichtsensoren grob anpeilen. Für beide Bots ist der Weg ins Ziel genau gleich weit, denn der Parcours ist symmetrisch. Darüber hinaus fällt der Weg links- oder rechtsherum an der Wand entlang deutlich länger aus als ein Weg durch die Mitte. Ein Bot, der diese Abkürzung zuverlässig findet, hat gute Aussichten, das Match für sich zu entscheiden. Es gewinnt derjenige, dessen Zentrum sich als erstes über einem der Zielfelder befindet. Sollte der unwahrscheinliche Fall eintreten, dass beide Bots zum exakt gleichen Zeitpunkt ins Ziel einlaufen, entscheidet das Los.

Wer mit dem Gleitpin oder mindestens einem Rad in einen der lauernden Abgründe hineingerät, bleibt stecken – das Rennen ist für ihn gelaufen. Rempeleien mit Kontrahenten und Wänden sind zwar erlaubt, bergen aber die Gefahr, dass der Bot sich dabei festfährt. Die Uhr läuft dabei weiter – erreicht keiner der Bots das Ziel, endet das Rennen nach einer festgelegten Zeitspanne. Wer bei Ablauf des Countdowns näher am Ziel ist, gewinnt das Duell.

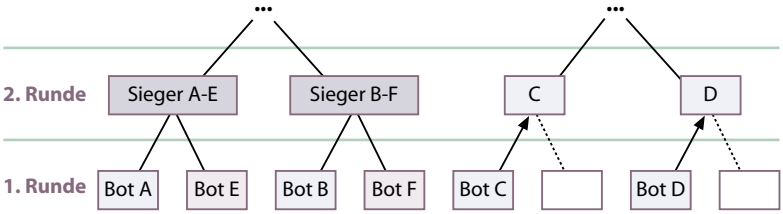
Virtueller Wettlauf

Damit alles fair abläuft, starten die Bots jeweils auf einem separaten System. Die Beschränkungen der echten Bot-Hardware sind für die Dauer des Turniers teilweise außer Kraft gesetzt. Um den virtuellen Bots die Reise durch das unbekannte Labyrinth zu erleichtern, dürfen sie alle Ressourcen ihres Wirtsrechners nutzen. Das heißt, ihnen stehen rund 256 MByte RAM und die Rechenleistung einer gängigen CPU zur Verfügung. Der Wettkampf-Server, eine spezielle Version des c't-Sim, läuft auf einem dritten System. Er setzt die Bots zu Beginn eines Matches an den Startpunkten ab, gibt den Ring frei, begrenzt unerbittlich die Zeit, die den Kontrahenten für die Suche nach dem Ziel bleibt, und bestimmt mit unbestechlicher Präzision den Sieger.

Das Turnier wird rundenweise nach dem K.-o.-System abgewickelt; jeweils der Gewinner eines Duells qualifiziert sich für die nächste Ausscheidung. Nur unter den vier besten werden alle Plätze ausgespielt. Die Begegnungen der ersten Runde werden ausgelost. Treten nicht genügend Teams an, um alle Zweikämpfe des ersten Durchgangs zu füllen, erhalten möglichst viele Bots ein Freilos, das sie automatisch für die nächste Runde qualifiziert (siehe Abbildung S. 212). Im Extremfall findet somit in der ersten Runde des Turniers nur ein einziges Duell statt – dafür sind alle anschließenden Durchgänge voll besetzt. In ihnen hat jeder Teilnehmer pro Runde ein Match zu bestehen, der Server setzt die Bots dabei auf jeder Stufe in einem neuen Labyrinth aus. Denn wer beim c't-Sim-Wettbewerb bestehen will, muss flexibel sein und mit möglichst unterschiedlichen Irrgärten klarkommen.



Die Wettkampfarenen sind symmetrisch, das Ziel liegt immer zentral in der oberen Wand. Der Weg an der Wand entlang ist länger als die Alternative durch die Mitte.



Die Begegnungen der ersten Runde werden ausgelost, wobei möglichst viele Bots ein Freilos erhalten, das sie für die nächste Runde qualifiziert.

Was tun?

Seit c't-Ausgabe 2/06 erschienen in dichter Folge Artikel zum c't-Bot, sie sind auf der Projektseite [2] kostenlos zu lesen. Sie beschreiben, wie man einen c't-Bot in C von Grund auf selbst programmieren kann, sodass er beispielsweise Wänden folgt oder seine Umgebung nach Lichtquellen absucht. Verschiedene Verhaltensmodule und Steuerroutinen für simulierte und reale c't-Bots wurden vorgestellt, die sich mit etwas Geschick erweitern oder zu ganz neuen Programmen kombinieren lassen. Eine Fülle von Tipps, Anleitungen und Anregungen liefern die Projektseite, ein Leserforum und eine Mailingliste (Anmeldung unter www.heise.de/bin/newsletter/listinfo/ct-bot-entwickler), sodass Neulingen noch rechtzeitig für den Wettbewerb der Einstieg in die Materie gelingen kann.

Zum Durchqueren eines Labyrinths existieren verschiedene Strategien. Liegen Start und Ziel beide an oder in einer Außenwand wie bei unseren Wettbewerbskursen, so führt der so genannte Höhlenforscher-Algorithmus sicher, aber nicht unbedingt schnell ans Ziel. Eine Implementierung für solches Wandern an der Wand entlang wurde in c't bereits vorgestellt [4]. Natürlich kann man das Renntraining damit beginnen, den vorgefertigten Wandfolger auf Schnelligkeit zu trimmen oder zu versuchen, ihn Sackgassen frühzeitig erkennen und abkürzen zu lassen.

Mehr Erfolg versprechen allerdings Strategien, die Ausschau nach einem kürzeren Weg halten. So lässt sich die Symmetrie des Labyrinths bei der Navigation ausnutzen. Loggt ein Bot die eigene Position mit Hilfe von Odometrie oder dem Maussensor mit [5], findet er möglicherweise den schnellen Weg durch die Mitte des Spielfelds, anstatt sich an den Hindernissen des rauen Randes aufzuhalten.

Messwerte der Distanzsensoren am c't-Bot

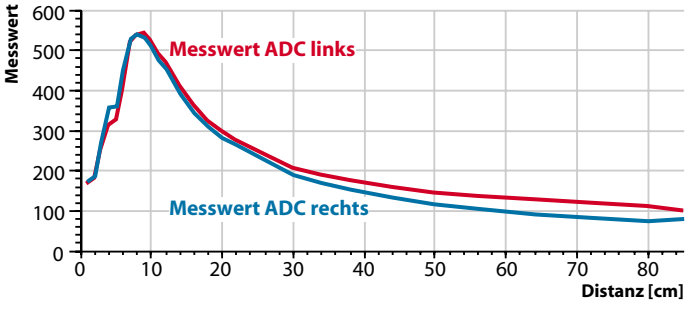
Distanz (cm)	Messwert links	Messwert rechts
1	167	173
2	182	187
3	252	270
4	312	357
5	326	360
6	400	450
7	520	530
8	538	540
9	543	531
10	525	511
11	493	475
12	470	452
14	412	391
16	364	345
18	323	310
20	298	281
22	276	265
26	240	225
30	206	191
34	189	170
38	175	153
44	160	133
50	145	118
56	135	106
64	128	92
72	119	83
80	110	76
85	100	80

Sensorflimmern

Clever verhält sich ein Renn-Roboter, der zwischen Wänden und anderen Bots zu unterscheiden weiß. Zuverlässige Daten über die eigene Bewegung liefern Maussensor und Rad-Encoder – vergleicht er diese Informationen mit der Distanzänderung, welche die Abstandssensoren melden, kann ein Bot ein bewegliches Hindernis von einem festen unterscheiden. Aber Achtung: Die Abstandsmesser besitzen in der Wettbewerbsversion des c't-Sim Kennlinien, die an jene der realen GP2D12-Sensoren angenähert sind (siehe Abbildung unten). Die Messwerte gaukeln dem simulierten Bot jetzt einen rapide wachsenden Abstand vor, wenn er zu nah an ein Hindernis heranfährt. Diesen Effekt muss die Bot-Logik wiederum herausrechnen.

Die Sensoren liefern bei gleicher Distanz zum nächsten Objekt leicht unterschiedliche Werte zurück – eine Folge davon, dass beide um 180 Grad gegeneinander verdreht in den Bot eingebaut werden. Als Referenz für die Sensormodellierung im c't-Sim diente ein c't-Bot mit stabilisierter Spannungsversorgung für die Distanzmesser. Dazu wurden parallel je ein 100nF-Kondensator, ein Tantalelko 10µF und ein Elko mit 100µF direkt an die Betriebsspannungspins des Sensors gelötet. Die Tabelle links zeigt einige Messwerte bei dieser Konfiguration.

Die entscheidenden wertvollen Sekunden kann ein Bot gewinnen, der seine Distanzsensoren auch zur Zielerkennung einsetzt. Denn die Zielfelder liegen in einer Lücke der Außenwand – die Abstandssensoren eines Bots, der direkt darauf zusteuert, sollten eine unendliche Distanz zum nächsten Hindernis melden.



Messungen: Torsten Evers

Im Wettbewerb liefern die Distanzsensoren Werte, die der Kennlinie originaler Sensoren angenähert sind – viele Messungen fallen dadurch zweideutig aus.

Kommt man den Wänden nicht zu nahe, lassen sich die Messwerte der Abstandssensoren auch benutzen, um eine einfache Umgebungskarte zu erstellen. Da die festen Hindernisse der c't-Sim-Welten derzeit streng dem Grundraster folgen, reichen gelegentliche prüfende Blicke, um zu einer groben Skizze der Umgebung zu kommen, auf der die eigene Position analog zur odometrisch gemessenen Bewegung verschoben werden kann. Die Hardware-Beschränkungen des echten Bot spielen beim c't-Sim-Wettbewerb keine Rolle. Daher sind Rechenaufwand und Speicherbedarf von Kartografie-Algorithmen weit weniger kritisch als beim Betrieb eines echten Bot.

Eine Patentlösung für unsere Aufgabe gibt es wahrscheinlich nicht – die Auswahl einer cleveren Strategie und deren effiziente Implementation stellen die Herausforderungen unseres Wettbewerbs dar. Die vorangegangenen c't-Artikel der Roboterserie stellen so gut wie alle notwendigen Verhaltensbausteine für einen erfolgreichen Rennbot vor. Wie gut der eigene Algorithmus funktioniert, zeigt ein Testrennen mit dem Wandfolger [4], dessen Steuerungsroutinen die aktuelle Version des c't-Bot-Codes im CVS enthält. Mit dieser Methode kommt ein Bot zwar sicher durch das Labyrinth, wenn auf seinem Kurs keine Fallgrube lauert. Den kürzesten Weg findet er in den Wettbewerbsparcours aber niemals. Darüber hinaus ist der Algorithmus nicht auf Geschwindigkeit optimiert. Ob er sich noch neben einer Wand befindet, prüft der Bot jedes Mal, wenn er sich um seinen eigenen Durchmesser nach vorne bewegt hat – pro Rasterfeld der Karte etwa zweimal, eine Umsicht, die aufhält.

Trainingslager

In Trainingsläufen sollte jeder Wettbewerbsteilnehmer in der Lage sein, diesen Testbot zu schlagen. Solche Proberennen gegen den Wandfolger oder ältere Versionen des eigenen Codes führt die aktuelle Version des c't-Sim durch. Ein Schiedsrichter (Klasse Judge.java) gibt das Rennen erst frei, wenn zwei Roboter auf den Startfeldern im Simulator angetreten sind. Der Kampfrichter misst auch die Zeit und überwacht den Zieleinlauf. Wer den

c't-Sim weiterhin in herkömmlicher Weise (etwa mit einem einzelnen Bot) benutzen will, entfernt in der Datei ct-sim.xml einfach folgende Zeile:

```
<parameter name="judge" value=
    "ctSim.Model.Rules.LabyrinthJudge"/>
```

Im Training sollte ein Bot mit möglichst vielen verschiedenen, ihm unbekannten Labyrinthen klarkommen, um eine Chance gegen die Konkurrenten zu haben. Natürlich könnte man jetzt mit einem Texteditor verschiedene Wettkampfarenen, die unseren Regeln entsprechen, von Hand tippen [3] und die Bots darin auf die Bewährungsprobe stellen. Einfacher geht es jedoch, wenn die Spielfelder automatisch generiert werden. Dafür haben wir einen Labyrinth-Erzeuger programmiert. Ein Klick auf den Menüpunkt Welt/Generieren… erzeugt jedes Mal einen neuen wettbewerbstauglichen Parcours im c't-Sim. Möchte man mit dieser Rennstrecke intensiver trainieren, kann man sie über Welt/Speichern als… unter einem beliebigen Dateinamen als XML speichern.

Doping-Kontrolle

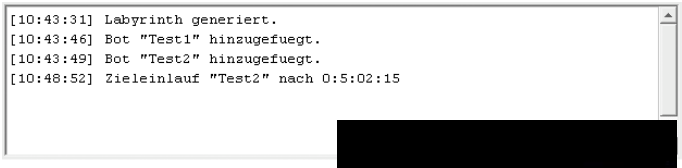
Jedem Bot stehen nur die Daten der eigenen Sensoren zur Verfügung – Hilfe von außen würde als unsportliches Verhalten disqualifiziert. Damit einerseits kein Bot schummelt und beispielsweise die Netzwerkverbindung des Rivalen stört und andererseits Chancengleichheit herrscht, müssen alle Wettkampf-Bots das Referenz-Framework benutzen. Dieses kümmert sich um die komplette Kommunikation mit dem c't-Sim. Der eigene Code darf nicht selbstständig Netzwerkkommandos oder Systemaufrufe absetzen – die Kommunikation mit Simulator und Wirtsrechner erledigt das vorgefertigte c't-Bot-Framework ohnehin. Für alle Routinen, die für die Intelligenz und das Verhalten des Roboters verantwortlich zeichnen, gelten keine Beschränkungen. Wer möchte, kann die gesamte Datei botlogik.c umschreiben. Damit die Bots reibungslos arbeiten, sollte der eigene Code eine Routine `bot_behave()` zur Verfügung stellen, innerhalb derer er seine Arbeit erledigt. Er muss diese Routine auch wieder verlassen, damit der Rest des Hauptprogramms noch zum Zuge kommt. Wo genau die Grenzen des Erlaubten liegen, regeln die verbindlichen Teilnahmebedingungen auf der Wettbewerbsseite [1].

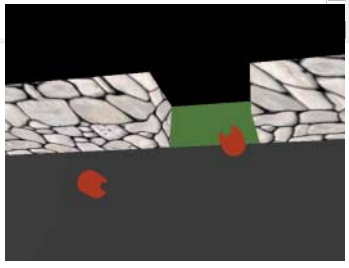
Als Wettbewerbsserver kommt die jetzt aktuelle Version des c't-Sim zum Einsatz. Wir behalten uns allerdings vor, bis zum Anmeldeschluss zum Wettbewerb (am 8. September) noch kleinere Änderungen an dieser Umgebung vorzunehmen, über die wir angemeldete Teilnehmer per Mail informieren. Damit nach diesem Termin trotzdem weiter am Simulator gefeilt werden kann, haben wir für den Wettbewerbsserver im Code-Archiv einen eigenen Entwicklungszweig angelegt. Wie man den jeweils neuesten Simulator aus diesem Entwicklungszweig gezielt aus dem CVS auslesen und installieren kann, beschreibt der Kasten rechts.

Die Wettbewerbsbeiträge der Teilnehmer erwarten wir in Form von Patches. Ein Patch ist eine Textdatei, die alle Unterschiede zwischen dem Referenz-Framework für den Roboter im CVS und der lokalen Kopie des Codes auf dem eigenen Rechner enthält. Wie man an den Code aus dem CVS kommt und Patches erstellt, beschreiben ausführliche Anleitungen auf der Projektseite. Für eine stabile Wettbewerbsversion des c't-Bot haben wir in dessen Programmarchiv ebenfalls einen gesonderten Zweig abgespalten.

Die Patches mit ihrem Wettbewerbscode laden registrierte Teilnehmer über unsere Wettbewerbsseite im Web [1] in eine Datenbank. Einmal am Tag starten die c't-Server einen automatischen Build-Prozess, die Rückmeldungen des Compilers erhalten die Teams per E-Mail. Sollte die vor Warnungen und Fehlern wimmeln, ist das kein Beinbruch – bis zum Annahmeschluss am 6. Oktober kann jedes Team beliebig oft neue Programmversionen für seinen Bot einschicken. Allerdings löschen auch fehlgeschlagene Compiler-Läufe bisher vorhandene Bot-Binaries, sodass Einsendungen in letzter Minute ein Wagnis darstellen. Bringt dann ein vergessenes Semikolon den Übersetzer auf dem c't-Server ins Stolpern, nimmt das Team mangels ausführbarer Datei nicht am Turnier teil. Fertig kompilierte virtuelle Bots können wir aus Sicherheitsgründen leider nicht annehmen.

Tipps und aktuelle Hinweise zum Ablauf des c't-Sim-Wettbewerbs erscheinen auf der Projektseite [2], im Forum und auf der Mailingliste. Pünktlich zum Anmeldeschluss für das Turnier unterbrechen wir die Entwicklung des c't-Wettbewerbssimulators; bis dahin freuen wir uns wie immer über Anregungen, Erweiterungspatches und Fehlerberichte. (pek)

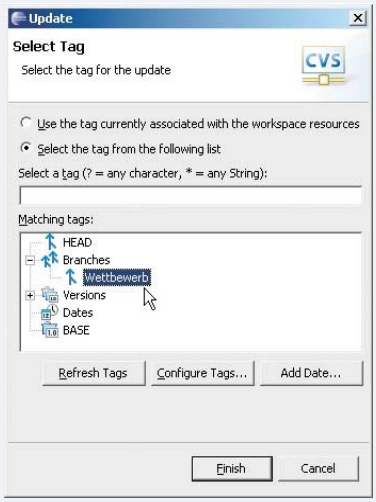




Bei den Trainingsläufen Bot gegen Bot sorgt der c't-Sim für einen korrekten Start, misst die Zeit und kürt den Gewinner – jenen Bot, dessen Zentrum sich zuerst über einem der Zielfelder befindet.

Abgezweigt

Die Turnierversion des c't-Sim steckt in einem neuen Zweig des CVS-Code-Archivs, der über das Tag „Wettbewerb" markiert ist. Wir empfehlen, für den Wettbewerbssimulator das Modul ct-sim aus dem CVS erneut auszulesen und in die Entwicklungsumgebung Eclipse als separates Projekt in den Workspace zu importieren. Im Kontextmenü, das ein Rechtsklick auf den Projektnamen zum Vorschein bringt, wählen Sie Team/Switch to Another Branch or Version. Im folgenden Dialog markieren Sie die Option Select the tag from the following list. Ein Mausklick auf das Pluszeichen klappt eine Baumdarstellung der Branches auf, in der Sie die Marke „Wettbewerb" auswählen und anschließend den Vorgang mit Finish beenden. Auf dem Laufenden halten Sie diese Codevariante mit dem CVS-Kommando update: Es ersetzt die lokale Programmkopie durch die neueste Revision innerhalb des Zweigs. Das Referenz-Framework für den c't-Bot erhalten Sie auf die gleiche Weise, das Tag des Zweigs lautet hier ebenfalls „Wettbewerb".



Mit Hilfe der grafischen Anzeige verschiedener Entwicklungszweige fällt in Eclipse der Versionenwechsel leicht.

Literatur

[1] Webseite zum c't-Sim-Wettbewerb: www.heise.de/ct/ftp/projekte/ct-bot/sim-contest/
[2] Webseite zum c't-Bot-Projekt: www.heise.de/ct/ftp/projekte/ct-bot
[3] Benjamin Benz, Genesis, c't-Sim: Weltenbau und Netzwerkzuschauer, c't 11/06, S. 214
[4] Torsten Evers, Ausgang gesucht!, Komplexe Verhalten für den c't-Bot selbst entwickelt, c't 10/06, S. 236
[5] Torsten Evers, Wo bin ich?, Positionsbestimmung für den c't-Bot, c't 13/06, S. 226

Soft-Link 0615210 ct