Praktikum:
Khepera Roboter-Fussballspiel

Betreuer: Dr. G. Schrott
schrott@in.tum.de

Daniel Gull
gull@in.tum.de

Holger Jehle
jehle@in.tum.de

21. Oktober 2003

Überarbeitet von
Michael Braun & Philipp Hemmer

Inhaltsverzeichnis

1 Allgemein **3**
1.1 Praktikum . . . 3
1.2 Aufgabenstellung . . . 3
1.3 Robotik . . . 3

2 Die Umgebung **4**
2.1 Das Spielfeld . . . 4
2.2 Die Roboter . . . 4
2.3 Der Server . . . 7
2.4 Der Client . . . 7

3 Entwicklung **8**
3.1 Die Klasse CKhepera . . . 8
3.2 Erweiterte Fahrbefehle . . . 10
3.2.1 Erklärung technischer Eigenschaften . . . 10
3.2.2 Kleines Repetitorium Trigonometrie . . . 11
3.3 Anwendung der Erweiterung . . . 13
3.3.1 Code-Beispiel . . . 13
3.3.2 Programmieren . . . 13

4 Strategie **14**
4.1 Überlegungen . . . 15

1 Allgemein

1.1 Praktikum

Der Lehrstuhl Knoll bietet ein Praktikum an, dessen Ziel die Erarbeitung eines Programmes ist, das Roboter gegeneinander Fussball spielen lässt. Auf einer ca. 105 cm x 65 cm grossen Spielfläche spielen vier Khepera-Roboter mit einem Tennisball in zwei Teams gegeneinander „Fussball“.

1.2 Aufgabenstellung

Aufgabenstellung des Praktikums ist es, ein Programm zu entwerfen, welches das eigene Team so steuert, dass es durch geschickte Spielzüge und Teamplay der zwei Roboter zu einem interessanten Spiel, möglichst mit Torschuss und Sieg der eigenen Mannschaft kommt.
Dieses Programm wird auf einem Praktikumsrechner ausgeführt und kommuniziert mit dem Server. Auf diesem Weg können die Koordinaten der Roboter und des Balles in Erfahrung gebracht werden, sowie Befehle an den Roboter abgesetzt werden.
Da sowohl Server als auch Roboter keinen Einfluss auf das Spielgeschehen nehmen, muss die gesamte Logik in dem zu erarbeitenden Programm implementiert werden.

Dabei liegt der Schwerpunkt darauf, dass ein strategisch interessantes Spiel entstehen soll, also nicht alle Roboter gleichzeitig in Richtung Ball losrasen, sich dort gegenseitig blockieren und zu einem Deadlock führen, so dass das Spiel dann abgebrochen werden muss. Nach einem Neustart passiert dann im ungünstigen Fall sofort dasselbe, was das Abschlussturnier des Praktikums recht eintönig gestaltet.

1.3 Robotik

Interessant ist auch die Möglichkeit, während des Praktikums Erfahrung im Umgang mit Robotern und deren Steuerung zu sammeln.

So fällt z.B. auf, dass die Roboter trotz einwandfreier Berechnung des Fahrtweges manchmal nicht das erwartete Ziel erreichen.

Dies hat mehrere Gründe:

- die Ausgangswerte der Berechnung sind mit Messfehlern behaftet

- die Odometrie der kleinen Roboter ist nicht perfekt - das heisst die Ausführung der Befehle ist teils mit erheblichem Fehler durch Reibung und Schlupf behaftet.
- Übertragungsfehler vom Server zum Roboter auf dem Funkweg können dazu führen, dass einzelne Befehle übersehen werden, also nicht ausgeführt werden - es besteht der Verdacht, dass eingeschaltete Mobiltelefone im Praktikumsraum dieses Verhalten fördern.
- der Server kann nicht in jeder Situation alle Roboter erkennen. Oft ist dies nur für ein Einzelbild der Fall.

Diese Fehler müssen berücksichtigt werden, um dem Roboter vernünftige Spielzüge ausführen zu lassen.

2 Die Umgebung

2.1 Das Spielfeld

Das Spielfeld besteht aus einer schwarzen Grundfläche mit Abmessungen von ca. 105 cm x 65 cm. An beiden Enden befindet sich eine 30 cm breite Ausbuchtung die das Tor darstellt - eine kleine Absperrung verhindert dass die Roboter selbst in das Tor fahren können und sich damit dem „Blick“der über dem Spielfeld angebrachten Kamera entziehen.
Diese Kamera ist an den **Server** angeschlossen, welcher die Roboter anhand von Farbmarkierungen erkennt und zuordnet.
Ein gelber Tennisball dient als Ball, der von den Robotern unter der Absperrung hindurch ins Tor manövriert werden soll.

2.2 Die Roboter

Die Roboter tragen die Bezeichnung Khepera und sind von einem Hersteller der sich K-Team nennt.
Die Geräte haben eine zylindrische Abmessungen von ca. 5,5 cm Durchmesser und ca. 8 cm Höhe.
Sie wurden ursprünglich für die Forschung entwickelt, um Algorithmen, die in der Simulation entwickelt und getestet wurden, in der realen Welt auszuprobieren.

Die Roboter besitzen zwei Elektromotoren, die ihre Drehung über ein Getriebe auf die beiden Räder übertragen.[1] Kurvenfahrt wird durch verschiedene Drehgeschwindigkeiten der Räder erreicht - dasselbe Prinzip wie bei Kettenfahrzeugen. Dabei ist anzumerken, dass die Odometrie nicht sehr exakt ist - die gewünschte Bewegung entspricht aufgrund von Schlupf und Reibung nicht zwingend der tatsächlich ausgeführten.
Zusätzlich sind knapp über Fahrbahnhöhe an den Robotern in den möglichen Fortbewegungsrichtungen Infrarotsensoren an den Robotern angebracht, die Hindernisse erkennen können. Dies wird z.B. von den Befehlen *shoot* und *dribble* genutzt, um den Ball zu erkennen.[2]
Es ist möglich, verschiedene Aufsätze, sogenannte Turrets, auf den Khepera zu stecken, die mit Greifer oder Kamera ausgerüstet sind. Die Kameraversion (Vision Turret) besteht aus einer horizontalen Zeilen-Kamera (64 Pixel, dies entspricht einem 36 Grad Blickfeld), die abgestimmt auf die Lichtstärke ein Graustufenbild erzeugt.
Die von uns verwendeten Roboter sind zusätzlich zur Zeilenkamera mit erweiterter Energieversorgung in Form von Akkus und einem Funkmodem ausgestattet. Das folgende Bild zeigt die Fortbewegungseinheit mit den Sensoren und das zweite die Roboter auf dem Spielfeld.

[1] Die Räder werden von je einem Gleichstrommotor angetrieben. Pro Umdrehung werden 24 Impulse erzeugt. Die Untersetzung beträgt 25:1. Dies ermöglicht pro Radumdrehung 600 Impulse.

[2] Die Sensorik des Khepera besteht aus acht Infrarot-Abstandssensoren mit je einem Infrarot-Sender und Empfänger. Diese können sowohl das normale Umgebungslicht, als auch von Objekten reflektiertes Licht messen.

Die Orientierung über die Zeilenkamera ist jedoch stark fehleranfällig und wurde deshalb durch eine hochauflösende Kamera über dem Spielfeld ersetzt, die Kameras der Roboter selbst finden derzeit keine Verwendung mehr.
Die Roboter sind mit einem Motorola m68k Prozessor bestückt, auf dem ein Programm ausgeführt wird. Dieses empfängt Signale via Funkmodem vom Server und führt diese durch entsprechende BIOS-Aufrufe aus.

Dieses Programm befindet sich in einem flüchtigen, batteriegepufferten Speicher, was bedeutet, dass es nach einem Stromausfall (z.B. Batterie leer) neu auf den Roboter heruntergeladen werden muss. Dies geschieht über die serielle Schnittstelle des Servers und dem Programm *Kheplab*. Dies wird vom Tutor übernommen, und sollte ohne Einweisung nicht selbst versucht werden.

2.3 Der Server

Der Server besteht aus normaler x86 Hardware, die mit einer Video - Framegrabber - Karte aus dem Hause Matrox versehen ist (Typ: Meteor). An der seriellen Schnittstelle ist ein Funkmodem angeschlossen, das die Kommandos an die Roboter weitergibt. Die andere serielle Schnittstelle wird dazu verwendet, um über ein Kabel das Basisprogramm auf den Roboter selbst zu übertragen (s.o.). Die Hauptaufgabe des Systems besteht darin, über die Framegrabber Karte Bilder der Kamera über dem Spielfeld zu empfangen, diese auszuwerten und den Client-Programmen die so gewonnenen Daten über Roboter und Ball zur Verfügung zu stellen.
Das Auswerten der Kamerabilder ist eine rechenintensive Aufgabe, die leicht gestört werden kann. Liegen im „Sichtbereich"der Kamera helle Gegenstände, so können diese die Algorithmen der Bildauswertung zum Erkennen der Roboter stören, und es werden sehr ungenaue oder insgesamt falsche Positionskoordinanten zurückgegeben. Dies kann schon durch einen Eingriff mit der Hand ins Spielfeld verursacht werden, um z.B. den Roboter aus einer Ecke zu befreien. Es ist daher empfehlenswert, die Koordinaten einem Plausibilitätstest zu unterziehen.
Für die Auswertung der Bilddaten werden die „Halcon-Libraries"verwendet, die ursprünglich am Lehrstuhl Radig entwickelt wurden.
Der Server wurde zuletzt im Sommersemester 2003 im Rahmen eines Systementwicklungsprojektes überarbeitet. Dabei wurde vor allem die Roboter- und Ballerkennung stark verbessert, so dass nun auch bei schlechteren Lichtverhältnissen die Roboter sicher erkannt werden.
Die Ankopplung der Clients an den Server erfolgt via TCP/IP (Sockets). Die vom Client gesendeten Kommandos werden dann vom Server serialisiert und per Modem an die Roboter weitergeleitet.

2.4 Der Client

Hier geschieht die eigentliche Spielsteuerung - auf den Clients sollen Programme entwickelt werden, die sich über die Spielsituation Überblick verschaffen und dementsprechend die Roboter des eigenen Teams koordinieren. Für jedes Team steht ein Rechner (Client) zur Verfügung. Am Ende des Praktikums wird der beste Client über ein Turnier ermittelt.
Um ein Turnier fair austragen zu können, müssen sich die Clients an einige Regeln halten:

- Nur 2 Roboter steuern
  Auch wenn die Client-Schnittstelle theoretisch die Steuerung aller vier Roboter erlaubt, dürfen pro Client nur zwei Roboter angesprochen und mit Befehlen versorgt werden. Die Abfrage der Koordinaten aller vier Roboter ist selbstverständlich erlaubt.

- Nicht zu viele Befehle auf einmal senden
  Wie weiter unten nochmal genauer beschrieben dürfen nicht zu viele Befehle auf einmal abgesetzt werden. Dies behindert den gesamten Spielablauf.

- Der Status der Roboter und des Server
  Dieser kann anhand der Koordinaten ausgelesen werden (x = -1 und y = -1: Roboter wird nicht erkannt; x = -2 und y = -2: Server ist gestoppt). Besonders der Serverstatus sollte beachtet werden. Ausserdem sollten die Clients mit einer Pausefunktion ausgestattet sein.

3 Entwicklung

3.1 Die Klasse CKhepera

Um den Einstieg zu erleichtern wird ein Beispielprogramm zu Verfügung gestellt, das die Kommunikation mit dem Server übernimmt[3] und eine Beispielimplementierung der Basisfahrbefehle aufzeigt. Auch zum Austesten der Funktionen und der dazugehörigen Werte eignet sich das Programm sehr gut. Die benötigten Befehle sind in einer Klasse „CKhepera“gekapselt und werden als Methoden zur Verfügung gestellt:

- <k>amerkoordinaten anzeigen (verwendet: CKhepera::getBall() und CKhepera::getKoord())
  die Koordinaten des Balls und der Roboter werden ausgelesen und angezeigt. Dabei muss die getKoord-Funktion für jeden Roboter einzeln aufgerufen werden.

- <f>ahren: (verwendet CKhepera::gostraight(int length))
  der Roboter legt eine angegebene Wegstrecke (Werte zwischen -1000 und 1000 Einheiten) in die aktuelle Richtung zurück - das Spielfeld ist knapp 800 Einheiten lang, so dass die maximale Distanz ca. eine diagonale Durchkreuzung des Spielfeldes erlauben würde.

[3] An diesem Teil sollten während des Praktikums keine Änderungen notwendig sein

- <d>rehen: (verwendet CKhepera::rotate(int angle))
  dem Roboter wird ein Drehwinkel zwischen -360 und 360 Grad vorgegeben, um den er sich von der derzeitigen Position um die Hochachse drehen soll.

- <e>nginecontrol: (verwendet CKhepera::enginecontrol(int a, int b)
  den Fahrmotoren wird eine Drehgeschwindigkeit zwischen -200 und 200 vorgegeben - damit lässt sich z.B. ein Bogen oder enger Kreis um den Ball fahren oder zum Ball an die gewünschte Position fahren. Sinnvolle Werte liegen aber nur zwischen -20 und 20, da darunter bzw. darber sich die Geschwindigkeit nicht ändert. Diese Einheiten stehen in keinem Bezug zu den Positionseinheiten.

- <s>chuss: (verwendet CKhepera::shoot())
  bei diesem Befehl kommen erstmals die erwähnten Abstandssensoren des Roboters ins Spiel: Der Roboter beschleunigt stark, bis er auf ein Hindernis trifft - dann stoppt er. Dadurch soll es ihm möglich sein, den Ball zu schiessen oder einen Pass zu versuchen.

- dr<i>bbeln: (verwendet CKhepera::rollball(int length))
  Auch dieser Befehl verwendet die Abstandssensoren. Für den Einsatz dieses Befehls muss der Roboter wirklich nahe an den Ball positioniert werden, zudem muss der Roboter in der gewünschten Richtung hinter dem Ball stehen. Wird der Befehl nun mit einer Längenangabe gestartet, so schiebt der Roboter geleitet von seinen Sensoren den Ball über die angegebene Strecke vor sich her. Verliert er den Ball vorher, beendet sich der Befehl.

- <s>top: (verwendet CKhepera::stop())
  Verhält sich ein Roboter nicht wie gewünscht, so kann er gestoppt werden.

- <a>lle Roboter stoppen: (verwendet CKhepera::stop())
  Wie oben, nur dass damit alle im Spiel befindlichen Roboter angehalten werden - auch die des anderen Teams. Dies sollte normalerweise nicht notwendig sein.

Um das Verhalten der Roboter im Vorfeld etwas kennenzulernen, können diese Methoden mit einem spartanischen Menue angesprochen werden, das im Beispielprogramm enthalten ist (cmdLoop.cpp).

3.2 Erweiterte Fahrbefehle

3.2.1 Erklärung technischer Eigenschaften

Darauf aufbauend, ist es zweckmässig die Grundfunktionen des Client-Programmes um einige Fahrbefehle zu erweitern.

Sinnvoll könnte z.B. ein Befehl sein, der den Roboter zu einer bestimmten Koordinate fahren lässt. Ein sinnvolles Ziel könnte der Ball sein oder auch ein freier Punkt auf dem Feld, um einen gegnerischen Roboter zu umfahren. Wie bereits oben erwähnt, kann es aber passieren, dass der Roboter trotz korrekter Berechung und eindeutiger Angabe der Zielkoordinaten diese nicht exakt trifft, so dass Korrekturen der Bewegung nötig sind.
Dies kann man dadurch erreichen, dass regelmässig die von der Kamera gelieferten Koordinaten von Ball und Robotern überprüft werden, und ein Soll-Ist-Vergleich von erwarteten und tatsächlichen Werten durchgeführt wird. Auftretende Differenzen können dann frühzeitig korrigiert werden.
Dabei ist zu beachten, dass der Server ungefähr 5-8 Bilder pro Sekunde auswertet[4], und damit ca. alle 200 Millisekunden neue Koordinaten zur Verfügung stellt.

Es macht wenig Sinn, geringere Zeitabstände zum Abfragen der Kamerakoordinaten zu verwenden. Es bietet sich an, auch dieses Intervall zum Senden neuer Befehle einzuhalten.

An dieser Stelle ist eine technische Information angebracht: Die Funkverbindung vom Server auf den Roboter hat eine Bandbreite von 9600 Baud. Die Übertragung eines Befehls nimmt auf diesem Weg ca. 16 Millisekunden in Anspruch. Werden also ohne Verzögerung vom Client Kommandos abgesetzt, so führt dies zum einen dazu, dass diese auf dem Server gequeued und erst dann abgearbeitet werden, wenn Zeit dazu ist. Ausserdem kann dies dazu führen, dass die gesamte Verbindung zu den Robotern abstürzt.

Deshalb sollte darauf geachtet werden, dass zwischen dem Absetzen jedes Kommandos an einen Roboter kurze Pausen eingehalten werden - 100 Millisekunden haben sich als sinnvoll erwiesen[5].
Ausserdem muss dem Roboter auch erst die Möglichkeit gegeben werden einen Befehl komplett auszuführen. Wird ein Dreh- oder Fahrbefehl vor seiner vollständigen Ausführung durch einen anderen Befehl überschrieben[6],

[4] Stand Wintersemester 03/04 auf P2-350

[5] man 3 usleep

[6] der erste Befehl geht nicht verloren, er wird nur sehr schnell durch den nächsten abgelöst

dann ruckelt der Roboter. Nur der enginecontrol-Befehl lässt sich ohne Ruckeln überschreiben.

3.2.2 Kleines Repetitorium Trigonometrie

Nun aber zurück zu der Methode, die den Roboter an bestimmte Koordinaten dirigiert.

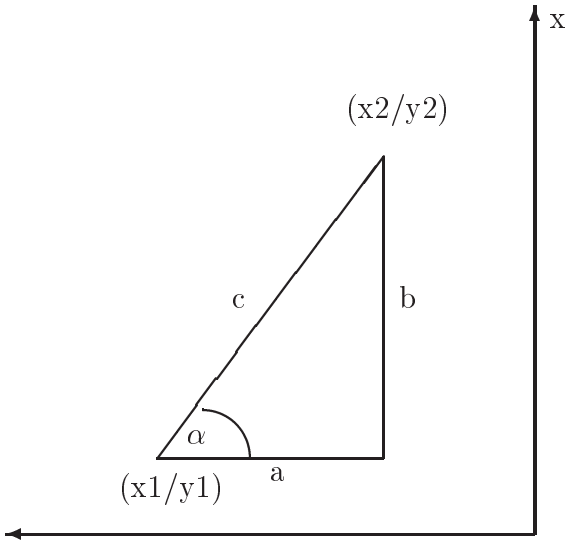
Die Erfahrung hat gezeigt, dass zu Beginn des Praktikums relativ viel Zeit darauf verwendet werden muss, sich trigonometrische Zusammenhänge und Grundkenntnisse wieder ins Gedächtnis zu rufen.

Deshalb kurz am Beispiel unserer gewünschten Methode eine Auffrischung:

Die eigenen Koordinaten des zu verwendenden Roboters lassen sich einfach mit der CKhepera::getKoord() Methode herausfinden.

Betrachten wir das Koordinatensystem im Spielfeld: Rechts unten befindet sich der Nullpunkt (ist auf dem Spielfeld beschriftet) mit x und y.

In diesem Koordinatensystem stellt man sich zwischen dem eigenen Standpunkt und dem Zielpunkt ein Dreieck vor.



Die Koordinaten der Punkt erhält man durch:

```
x1 = getKoord().x;
```

```
y1 = getKoord().y;
x2 = Coord.x;
y2 = Coord.y;
```

„Coord “ist ein der Methode übergebener struct mit den Zielkoordinanten.

Die Werte a, b und c lassen sich nun bestimmen:

```
a = y2 - y1;
b = x2 - x1;
c = sqrt ( a*a + b*b );
```

Der Wert c ergibt sich einfach aus der Anwendung des Satzes von Pythagoras.

Nun interessiert natürlich der Winkel alpha (in Grad), in den der Roboter gedreht werden muss, um zum Punkt x2, y2 zu „schauen “.

```
alpha = atan2 (b, a) * (180.0 / M_PI);
```

Die Multiplikation mit (180 / pi) muss erfolgen, da der Winkel sonst im Bogenmass errechnet wird, was für weitere Berechnungen unpraktisch ist.

Wie aus C/C++ Referenzmanuals zu entnehmen ist, liefert atan2 für die ersten beiden Quadranten positive Werte zwischen 0 und 180 Grad, für den dritten und vierten Quadranten negative Werte zwischen 0 und 180 Grad.

Der tatsächliche Winkel kann aber einfach ermittelt werden:

```
if (alpha <0)
       alpha = ( 360 - alpha );
```

Damit haben wir in alpha den Winkel relativ zur Y-Achse. Allerdings darf der aktuelle Stellwinkel des Roboters nicht vergessen werden, der ebenfalls mit der getKoord()-Funktion ausgelesen werden kann:

```
beta = ( alpha - getKoord().alpha);
```

In beta steht nun der relative Winkel zur aktuellen Drehposition. Nun sollte dieses Ergebnis noch optimiert werden. Dazu ein Beispiel: soll der Roboter vom Winkel 359 auf 0 gedreht werden, würde bei dieser Berechnung beta bei -359 liegen. Eine Drehung um 1 wäre in diesem Fall die bedeutend schnellere Alternative:

```
if (beta >180)
       beta = ( beta - 360 );
else if (beta <-180)
       beta = ( beta + 360 );
```

Nun haben wir alle wichtigen Informationen gesammelt, um zum gewünschten Punkt zu gelangen.

Wir müssen den Roboter nun in die gewünschte Richtung ausrichten (mit der CKhepera::rotate() Methode), dann warten, bis die Drehung vollzogen ist, und dann die errechnete Wegstrecke zurücklegen (mit CKhepera::gostraight()).

```
rotate(beta);
// 2 sec warten. Dies kann man verbessern, jede Sekunde zählt!
// z.B. nur solange warten,
// bis der gewünschte Drehwinkel erreicht ist
sleep(2);
gostraight(c);
```

Nun wäre es interessant, die Bewegung des Roboters anhand der Kamerakoordinaten zu verfolgen, und gegebenenfalls zu korrigieren.

Dies sei dem Leser überlassen.

3.3 Anwendung der Erweiterung

3.3.1 Code-Beispiel

Damit wäre bereits ein sehr einfacher Client entstanden:

```
int main () {
       CKhepera robo1(1);
       robo1.MoveToXY(robo1.getBall());
}
```

Mit diesen wenigen Zeilen sollte der Roboter 1 bereits zum Ball fahren.

3.3.2 Programmieren

Im khepera home-Verzeichnis /usr/proj/khepera[7] ist ein Verzeichnis „Client“vorzufinden, in dessen Unterverzeichnis „doc“auch diese Beschreibung zu finden ist.

Hier sind auch die Programm-Gerüste für die Programmieraufgaben vorzufinden:

Wie oben bereits erwähnt, wird die Kommunikation mit dem Server bereits

[7]Stand Wintersemester 03/04

komplett vom Gerüst zur Verfügung gestellt.
Um nicht durch den dafür notwendigen Code unnötig Verwirrung zu stiften, wurde das Gerüst so gestaltet, dass eine library dazugelinkt wird, die dann die Funktionalität bereitstellt.

Die für das Praktikum relevanten Dateien sind in den folgenden Verzeichnissen zu finden:

- src_demo_menu: Das oben beschriebene Programm. Es stellt die Basisfunktionalität über ein kleines Menü zur Verfügung.
- src_home_progging: Modifiziertes Makefile und leere CKhepera-Klasse. Hier wurde alles so zugeschnitten, dass zuhause programmiert werden kann. Damit ist es möglich, die Programme zuhause zu kompilieren und syntaktische Fehler zu finden. Testen lässt sich damit natürlich nicht da sich die leere Klasse zwar übersetzen lässt, aber keinerlei Funktionalität bietet. Bei den Programmierversuchen zuhause ist darauf zu achten, dass die Dateien CKhepera.cc und CKhepera.hh nicht verändert werden. Diese Veränderungen stehen auf dem Server dann nicht zur Verfügung (und werden auch nicht übernommen), folglich wird das entwickelte Programm auf dem Server nicht das Gewünschte leisten.
- src_client: Stellt den Startpunkt für die eigene Entwicklung dar. Das mitgelieferte Makefile enthält die notwendigen Pfade um die oben genannten Libraries anzubinden. Es übersetzt standardmässig eine Datei namens client.cc (.cc für C-Class). Soll die Datei, in der die main() - Funktion erwartet wird anders heissen, so ist hierfür der Eintrag „PROG := client“im Makefile zu ändern. Kann sich jemand mit der Endung .cc nicht anfreunden, so kann er dies mit dem Schlüssel „SUFFIX := cc“im Makefile seinen Wünschen entsprechend anpassen.
  Die Binaries landen, soweit nicht anders konfiguriert, im ../bin Verzeichnis - die Objekt-Files (*.o) in ../obj - auch dies kann nach Belieben verändert werden. Für sehr einfache Programme ist die explizite Verwendung von *.o Dateien nicht erforderlich, bei umfangreicheren Programmen aber empfehlenswert[8]. Für genauere Informationen sei auf die man-page von make verwiesen.

4 Strategie

[8]Beispiel ist bereits auskommentiert im Makefile enthalten

4.1 Überlegungen

Naheliegende Überlegungen sind nun, sich eine Klasse „strategie “oder ähnlich zu bauen, die das gesamte Spielvorgehen beobachtet, sinnvoll dirigiert und damit der Mannschaft zum Sieg verhilft.

Dabei sollte auf mögliche Verklemmungen geachtet und diese bereits im Vorfeld verhindert werden.

Zusätzlich könnte das Spielfeld bewertet werden, also je nach Stellung des Gegners in einen aggressiven oder passiven Modus geschaltet werden.

Weitere Überlegungen sind, das Programm in mehrere Threads zu zerlegen, um zeitgleich Spielfeld analysieren sowie Befehle absetzen zu können.

Viel Spass beim Entwickeln des Clients!