

---

# Programming Games

---

Market Research

---

**Project Contorium**  
Emre Can Kucukoglu

---

In this informally written document, I try to introduce my research about programming games. Please concern that Information in this document are not well designed and ordered.

My first investigation about concept is related with the game **Omega (1989)**.

(<http://www.mobygames.com/game/omega> )

In this game, player has given basically a budget and wanted to design a tank. Player actually writes the codes of tank by himself. Moreover he can use various artificial intelligence scripts.

Later, I found another game **RobotWar (1970s)**. (<http://corewar.co.uk/robotwar/>)

Main task is approximately same with Omega: "to program a robot, that no other robot can destroy."

The main activity of the game was to write a computer program that would operate a robot. The robots did not have direct knowledge of the location or velocity of any of the other robots; they could only use radar pulses to deduce distance, and perhaps use clever programming techniques to deduce velocity. Robot's programming language is similar to BASIC. There were 34 registers that could be used as variables or for robots I/O functions. An example program is like that:

```
SCAN
  AIM +5 TO AIM           ; MOVE GUN
  AIM TO RADAR           ; SEND RADAR PULSE
LOOP
  IF RADAR < 0 GOSUB FIRE ; TEST RADAR
  GOTO SCAN
FIRE
  0 - RADAR TO SHOT      ; FIRE THE GUN
ENDSUB
```

Also, there are another games, namely RoboWar (1992), Color Robot Battle (1981) which aim also programming robots to fight with each other.

<http://en.wikipedia.org/wiki/RoboWar>

[http://en.wikipedia.org/wiki/Color\\_Robot\\_Battle](http://en.wikipedia.org/wiki/Color_Robot_Battle)

Another game Robocode (2001), has written in Java, and this game helps people learn to program in Java. Because a simple robot can be written in just a few minutes. However perfecting a bot can take months. There exist several leagues for RoboCode. This shows me that this concept programming games can draw attention.

There is another interesting programming game Colobot.

<http://www.ceebot.com/colobot/index-e.php>

Colobot (Colonize with Bots) is an educational real-time strategy video game featuring 3D graphics. The main feature of the game which makes it educational is the ability of the player to program his robots using a programming language similar to C++ or Java. The game has been recommended by the Polish Ministry of National Education as a teaching aid in learning of basic algorithms and object-oriented programming.

The programming language used in Colobot is C-BOT, syntactically similar to Java.

```
extern void object::FetchTitanium()
{
    object item; // declare variable

    item = radar(TitaniumOre); // find a piece of titanium ore
    goto(item.position); // go to the ore
    grab(); // pick up whatever is in front of the robot (presumably
the ore)

    item = radar(Converter); // find the ore converter
    goto(item.position); // relocate to the converter
    drop(); // drop the piece of ore
    move(-2.5); // back up 2.5 meters to allow the converter to start
processing the ore
}
```

There are different tournaments and leagues for the programming games. Usually a script is optimized for a special strategy. For instance, in Mouse Run, a Java class is written by a programmer, which will provide the logic for a mouse to navigate through a maze and compete with other mice to find and consume a cheese first. Programming games may be derived from almost any other type of game. For example, the World Computer Chess Championship consists of matches between programs written for the abstract strategy game of chess. Also, some non-computer games borrow elements of programming games; in the board game RoboRally, players arrange cards to "program" their pieces with a sequence of moves, causing moves to be made without the knowledge of one's opponents' preceding moves.

When I was looking for a good player developed AI game, between the multitude of robot games, I caught a sport-competition based game. I want to remind that our game is sport based game, too. I have found a very complicated game, it is actually a car racing. TORCS.

Firstly, I tried to learn concept of the game from its webpage.

<http://torcs.sourceforge.net/>

Unfortunately, I couldn't catch anything, so I decided to download it and look at it. After downloading the game, a little but 3d animated simulation window opened:



I tried to play the game, assuming it is an usual game. I could select a car, select a name, select race and start a race.

In addition, I have to add that graphics are very satisfying.



However I was not the captain of my car. I just watched the race and I realized that cars are initially uniform and has nothing special. The race was just a city tour. Then, I went back to website again to learn the game.

Documentation of the webpage, 3 main links exist: Practice Mode, Quick Start Guide and Robot Tutorial (C++). First two of them are just explain the GUI of game, but I have already experienced them. The golden key of game is actually explained in Robot Tutorial.

<http://www.berniw.org/tutorials/robot/tutorial.html>

Game installation and system hardware and software requirements are explained in installation link. Second link, robot tutorial firstly explains how to prepare your robot skeleton, mean files of our ai. After just 5-6 command from prompt, our basic robot (car) is ready as template files. Then, coding part has begun. For example, our car need a good drive function. Of course thanks to default constructor functions we are already driving. But for example when our car can't go forward anymore, what does it do in this case? We have to implement it.

Because we use the files which we already created from command line, basic relationships of game files and ai files are already done. After opening the files with a proper C++ IDE, we can easily see the objects like 'car' and its entities. And with our programming skills we can write our own steer.

```
static void
drive(int index, tCarElt* car, tSituation *s)
{
    float angle;
    const float SC = 1.0;
```

```

memset(&car->ctrl, 0, sizeof(tCarCtrl));

if (isStuck(car)) {
    angle = -RtTrackSideTgAngleL(&(car->_trkPos)) + car->_yaw;
    NORM_PI_PI(angle); // put the angle back in the range from -PI to PI
    car->ctrl.steer = angle / car->_steerLock;
    car->ctrl.gear = -1; // reverse gear
    car->ctrl.accelCmd = 0.3; // 30% accelerator pedal
    car->ctrl.brakeCmd = 0.0; // no brakes
} else {
    angle = RtTrackSideTgAngleL(&(car->_trkPos)) - car->_yaw;
    NORM_PI_PI(angle); // put the angle back in the range from -PI to PI
    angle -= SC*car->_trkPos.toMiddle/car->_trkPos.seg->width;
    car->ctrl.steer = angle / car->_steerLock;
    car->ctrl.gear = 1; // first gear
    car->ctrl.accelCmd = 0.3; // 30% accelerator pedal
    car->ctrl.brakeCmd = 0.0; // no brakes
}
}

```

For example, with this code, we say our car that if it is stuck and can't go forward then reverse the gear, don't use brakes and try to change your direction. But there is a problem: How can we understand that our car is stuck? Then we need isStuck function. By the way, I took these codes from website and try to implement basic robot with this tutorial, that's reason why codes are well-prepared, commented. ☺ Nevermind, let's look isStuck function:

```

/* check if the car is stuck */
bool isStuck(tCarElt* car)
{
    float angle = RtTrackSideTgAngleL(&(car->_trkPos)) - car->_yaw;
    NORM_PI_PI(angle);
    // angle smaller than 30 degrees?
    if (fabs(angle) < 30.0/180.0*PI) {
        stuck = 0;
        return false;
    }
    if (stuck < 100) {
        stuck++;
        return false;
    } else {
        return true;
    }
}
}

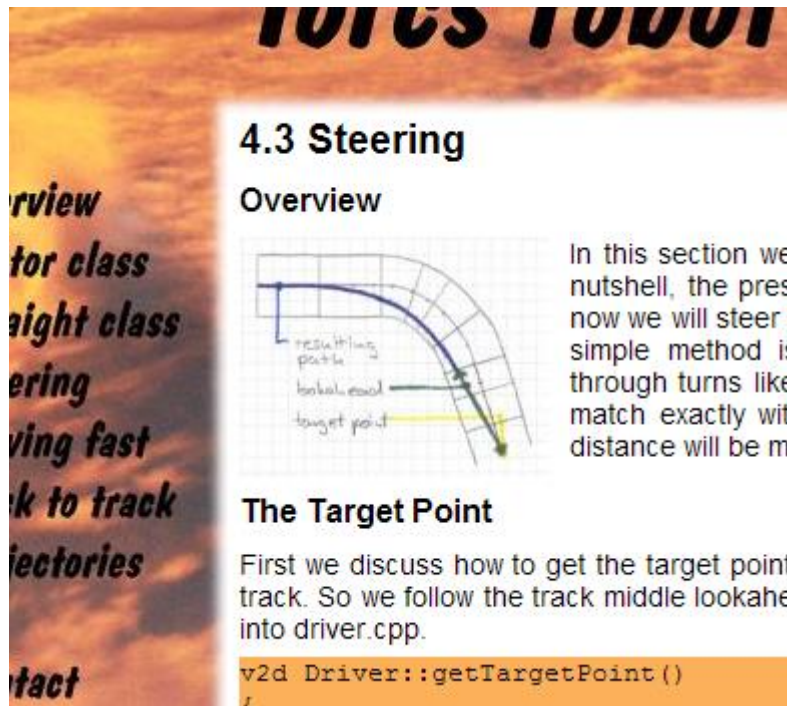
```

This code means that:

1. Init a counter to 0.
2. If the absolute value of the angle between the track and the car is smaller than a certain value, reset counter to 0 and return "not stuck".

3. If the counter is smaller than a certain limit, increase the counter and return "not stuck", else return "stuck".

Gameplaying part of TORCS is like that. We can implement pitstop class to improve our pit strategy, or we can write our own very very manual ☺ braking. Tutorial pages are very detailed. For example; even scientific part of calculations are explained when implementing steering strategy.



The image is a screenshot of a tutorial page for TORCS. The page has a dark, textured background on the left side with white text that is partially cut off. The main content area is white. At the top, the title "4.3 Steering" is displayed in a large, bold, black font. Below the title, the word "Overview" is written in a smaller black font. To the right of "Overview" is a diagram showing a curved track segment on a grid. A blue line represents the "resulting path", a green line represents the "look ahead", and a yellow dot represents the "target point". To the right of the diagram, there is a paragraph of text explaining the steering process. Below the diagram, the section "The Target Point" is introduced, followed by a paragraph of text. At the bottom of the screenshot, a code block is visible, showing the method signature `v2d Driver::getTargetPoint()`.

## 4.3 Steering

### Overview

In this section we nutshell, the pres now we will steer 1 simple method is through turns like match exactly with distance will be m

### The Target Point

First we discuss how to get the target point track. So we follow the track middle lookahe into driver.cpp.

```
v2d Driver::getTargetPoint ()
```

I try to research programming game concept and explain my experience and knowledge. These games are not very popular games, and they are really hard to play games. Most of them are open source licensed and none of them are sold by price. These games can be used as an educational purposes.

Also, there are webpages worth a look:

There is a wiki page: Programming Games Wiki. <http://www.programminggames.org>

There is also a page, which contains almost every programming games. Spesific information exist for every game in this page: <http://www.aiforge.net/game-links.htm>