About CyberTiggyr Team's Submission to ICFP's Annual Programming Contest (2003)

Gene Michael Stover   Pavel Repin

created Tuesday, 1 July 2003
updated Wednesday, 2 July 2003

Contents

1 The Team **1**

2 Goals **2**

3 Our Approach to the Problem **2**
3.1 Path-Picking 2
3.2 Trace-Making 2

4 Results **4**

5 What Could Be Improved **4**

6 What Went Right **5**

7 An Asinine Observation **5**

8 List of all Files **5**

9 What is the ICFP Programming Contest? **8**

1 The Team

The team consisted of Gene Michael Stover and Pavel Repin.

2 Goals

Gene had participated in two other ICFP programming contests & had learned that a lot of entries are disqualified because of erroneous output − including Gene's own entries. So this year, our goal was to create a correct program. Even if its output was not optimum, we wanted it to be correct & to pass all phases of the judging.

3 Our Approach to the Problem

Neither of us had any practical knowledge of path-finding algorithms, so we decided that, even if we could implement one in the period of the contest, we didn't know that it could run in a small enough time.[1]

So we decided that a human should pick the route, & the program should create the driving instructions to follow the route. The program would try to make good use of acceleration & breaking to reduce the drive time.

3.1 Path-Picking

First, we had to pick the path. We thought it would be feasible for a person to edit the track file, which is a great big text file. The human could mark the key points on the path with characters that were not already reserved in the task description. That would be pretty much every character except *b*, *g*, *r*, *w*, the dot, the asterisk, & the bang (*!*). Upon trying the technique, we realized that the files were too big for a human to edit.

Our second attempt at path-picking was TrackEditor. It displays the graphical image of the track & allows a human to click on the key points of the path. The program writes all those points to a text file.[2] This became part of our submission.

3.2 Trace-Making

The program that makes the trace files is Turtle.

We considered a lot of algorithms for picking a path from a list of points. After seeing what others did, I realize that we never considered a lot of really excellent ideas that could have been implemented in time. We decided to go with a simple one that we were sure we could implement in time. It goes like this:

1. Given
    (a) a list of points to visit,

[1] After seeing the write-ups of other teams about their approaches, I know that a path-finding algorithm can run in less time than the contest's period. Still not sure we could have implemented it in time.

[2] Gene still thinks the program should have written to standard output so we could pipe it into the Turtle program.

   (b) a starting position (the first point in the list),
   (c) a starting state (direction & speed), &
   (d) a target-breaking-speed.

2. The point where we are is $current - point$. The point we want to reach is $next - point$.

3. While the list of points ain't empty
   (a) Until you reach $next - point$
      i. If you could decelerate to the $target-breaking-speed$ before you reached $next-point$, choose from ($accelerate-left$, $accelerate-right$) the "better" instruction & output it.
      ii. Otherwise (couldn't decelerate in time), output a brake instruction.
   (b) $current - point \Leftarrow next - point$.
   (c) $next - point \Leftarrow nextpointfromlist$.
   (d) Pop from list of points.

4. You're reached the finish line. Go for a beer.

Notice that the algorithm doesn't know anything about the track. It doesn't know about walls or collisions. It just drives the points. The point-picker (two billion-node neural networks, in our case) is responsible for picking a path that Peter Piper won't crash into a wall.

One technique we considered for choosing between an $accelerate-left$ & an $accelerate-right$ was to determine the angle we'd have to turn when we reached the next point, so we could decelerate (or accelerate) to a speed so that one left or one right instruction would left us pointing in at the next point. Instead of dusting-off our old trigonometry books, we went with a simpler approach. It's in ./src/Turtle/Turtle.cpp. We consider each move we could make & get the distance it would leave us from the next point. We choose whichever move would leave us closer. Here's that chunk of code:

```
multimap<Unit, string> best_action;
best_action.insert(make_pair(Point::dist2(compute_step("l",
p, v, d), path[next_p]), "l"));
best_action.insert(make_pair(Point::dist2(compute_step("r",
p, v, d), path[next_p]), "r"));
best_action.insert(make_pair(Point::dist2(compute_step("a",
p, v, d), path[next_p]), "a"));
best_action.insert(make_pair(Point::dist2(compute_step("al",
p, v, d), path[next_p]), "al"));
best_action.insert(make_pair(Point::dist2(compute_step("ar",
p, v, d), path[next_p]), "ar"));
```

score	track
20546	1 Simple
29993	2 Hairpins
37430	3 Sepang
40896	4 EatYouAlive
12635	5 Car
48383	6 IcfpContest
9239	7 Gothenburg
35461	8 ManyWays
32125	9 PhilAndSimon
266708	**Total**

Figure 1: The number of steps our trace files require to finish each track & in total.

```
min_dist2 = best_action.begin()->first;
// The real output here.
printf("%s.", best_action.begin()->second.c_str());
```

4 Results

Our scores for each track & in total are in Figure 1.

5 What Could Be Improved

After seeing what other teams did, I wish we had considered more possibilities. I don't know how we could ensure better open-mindedness other than taking smartness pills (or maybe some other drug) or getting together sooner. Originally, Gene planned to participate in the contest alone.[3] Much of the design work was done over dinner on Friday night with a non-programming friend.[4] Pavel didn't join the team, didn't even know about the contest, until Saturday afternoon. I think it would have been better if we had been ready for the contest & had met, face-to-face, to read the problem description & form our approach as soon as the contest started.

Possibly, it would have been better if both team members used the same development platform. As it happened, one of us used Lisp & Bourne on Unix, & the other used C++ & C-shrap on Macrosopht Winders.

We're both getting too old to stay awake all night, hacking.

[3]After the fact, I'm pretty sure I couldn't have done it alone.

[4]And I value your input, Jay. Very helpful stuff.

6 What Went Right

We planned & designed, & we followed the plan & the design.[5]

We tested as we went. To have written the whole kit-&-kaboodle at once & debug later would have taken too much time.

Possibly most important, it was a lot of fun clicking on those paths in TrackEditor & then seeing the car drive the traces produced by Turtle. The spiraling spin-outs into orbit were a hoot late at night.[6] Finding a path through track 8 “Many Ways” was a fun challenge for our two, billion-node neural neural networks, but track 9 “Phil & Simon” was difficult enough that I wouldn’t want to do it again. Took hours. In fact, Gene passed out from exhaustion in the wee hours of Monday morning, & Pavel stayed awake & somehow found a viable path through Simon’s ear canal.

Somewhere in the world, while someone ran an artificially intelligent path-finding program, Pavel held a ruler up to the screen to find the longest straight-away he could in the choose-your-path program.

7 An Asinine Observation

We couldn’t help but notice, & can’t help but mention here, that what the contest’s task description calls “velocity” is really “speed”.

Velocity is a vector. Speed is the magnitude of velocity; speed is a scalar.

8 List of all Files

Here is a list of pretty much all the files we created. Some with comments.

- ./Makefile – builds many of the programs, but not TrackEditor or Turtle; it’s a Unix makefile & those are Winders programs
- ./README
- ./cybertiggyr.tar.gz – the file we uploaded to ICFP as our submission
- `./doc/` – some files provided by the ICFP contest (contestants will recognize these). All of these files should be available on the ICFP 2003 contest Web site
  - ./doc/Een.dump – a sample output of simulator
  - ./doc/Een.trc – a sample “trace” file, for the Een track
  - ./doc/problem.pdf – the problem description. It went through several versions; I believe this is version 1.

[5]Probably not a claim to fame, since we’re both long-time career programmers, so we should do things like that by habit.

[6]Lots of things can be really funny late at night: *Grand Theft Auto 3* and Jay Leno are but two examples.

- `./dumps/` – the *.dump files produced by our simulator when we ran it on the `*.trc` files produced by Turtle for the official tracks. Also, the `*.points` files which were inputs to Turtle.
  - ./dumps/1_Simple.dump
  - ./dumps/2_Hairpins.dump
  - ./dumps/3_Sepang.dump
  - ./dumps/4_EatYouAlive.dump
  - ./dumps/5_Car.dump
  - ./dumps/6_IcfpContest.dump
  - ./dumps/7_Gothenburg.dump
  - ./dumps/8_ManyWays.dump
  - ./dumps/9_PhilAndSimon.dump
  - ./dumps/1_Simple.points
  - ./dumps/2_Hairpins.points
  - ./dumps/3_Sepang.points
  - ./dumps/4_EatYouAlive.points
  - ./dumps/5_Car.points
  - ./dumps/6_IcfpContest.points
  - ./dumps/7_Gothenburg.points
  - ./dumps/8_ManyWays.points
  - ./dumps/9_PhilAndSimon.points
- ./src/
  - ./src/demo0050.sh – test the simulator on a hard-coded track map (the internal data structure for a track).
  - ./src/demo0051.sh – run timetrial on the `Een` trace & track files. The idea is to compare the output with ./doc/Een.dumpo verify the `timetrial` program & the simulator.
  - ./src/print-fp.c – reads numbers in "real" arithmetic system & prints them in ICFP 2003 "fixed point" system.
  - ./src/print-key-points.sh – Part of an earlier idea for letting the human specify the path in the `*.trk` file itself. This program would read that file & print the "key points" from it.
  - ./src/recons-track.sh – reads a track file into an internal data structure, then prints a new track file. It's to test the track-reading function.
  - ./src/runall.sh – run timetrial on all the trace files in `traces/`

  - ./src/sim.lisp – race car simulator. It's for validating the `*.trc` files produced by Turtle.
  - ./src/table-arith.sh – print a table of arithmetic facts in the ICFP 2003 "fixed point" system.
  - ./src/table-friction.sh – print a table of speeds & frictions (because friction depends on speed)
  - ./src/table-turns.sh – print a table of speeds & turning angles (because turning angle depends on speed)
  - ./src/test0020.sh – test `timetrial` by running it on `Een.trc` & `Een.trk` & comparing the output to `Een.dump`
  - ./src/timetrial.sh – run the simulator, given a `*.trc` & a `*.trk` file
  - ./src/track.lisp – functions for reading track files & querying a track for the terrain attributes. Used by the simulator.
  - ./src/TrackEditor – display a track so a human can select the path on it, point by point
    1. ./src/TrackEditor/App.ico
    2. ./src/TrackEditor/AssemblyInfo.cs
    3. ./src/TrackEditor/TrackEditor.csproj
    4. ./src/TrackEditor/TrackEditor.cs
    5. ./src/TrackEditor/TrackEditor.resx
  - ./src/Turtle/ – read a file of points (which describes a path) & produce a `*.trc` file.
    * ./src/Turtle/Physics.cpp
    * ./src/Turtle/Physics.h
    * ./src/Turtle/Point.h
    * ./src/Turtle/Turtle.cpp
    * ./src/Turtle/Turtle.vcproj
    * ./src/Turtle/Unit.cpp
    * ./src/Turtle/Unit.h

- ./traces/
  - ./traces/1_Simple.trc
  - ./traces/2_Hairpins.trc
  - ./traces/3_Sepang.trc
  - ./traces/4_EatYouAlive.trc
  - ./traces/5_Car.trc
  - ./traces/6_IcfpContest.trc
  - ./traces/7_Gothenburg.trc
  - ./traces/8_ManyWays.trc

  - ./traces/9_PhilAndSimon.trc
- ./tracks/
  - ./tracks/1_Simple.trk
  - ./tracks/2_Hairpins.trk
  - ./tracks/3_Sepang.trk
  - ./tracks/4_EatYouAlive.trk
  - ./tracks/5_Car.trk
  - ./tracks/6_IcfpContest.trk
  - ./tracks/7_Gothenburg.trk
  - ./tracks/8_ManyWays.trk
  - ./tracks/9_PhilAndSimon.trk
  - ./tracks/Een.trk
  - ./tracks/X_Rally.trk
  - ./tracks/TrackViewer.class
  - ./tracks/TrackViewer.java
  - ./tracks/TrackViewerWindow.class
  - ./tracks/TrackViewerWindow.java
  - ./tracks/track1.gif
  - ./tracks/track2.gif
  - ./tracks/track3.gif
  - ./tracks/track4.gif
  - ./tracks/track5.gif
  - ./tracks/track6.gif
  - ./tracks/track7.gif
  - ./tracks/track8.gif
  - ./tracks/track9.gif
  - ./tracks/trackeen.gif
  - ./tracks/trackxrally.gif
- ./make-dist – creates `./cybertiggyr.tar.gz`, our submission file

9 What is the ICFP Programming Contest?

It's an annual programming contest hosted by the International Conference for Functional Programming. This year (2003 C.E.), their contest was the ICFP programming contest for 2003, which probably seems logical to most people.

The contest page is http://www.dtek.chalmers.se/groups/icfpcontest/index.html. It contains the description of the programming task[7] (or see a local copy here).

[7] http://www.dtek.chalmers.se/groups/icfpcontest/task.html