

## The “Big” Exercise

- You are given a  $\sim 1$  kSLOC, non-trivial, slightly real-worldish program.
- Your task is to test and improve it as specified in the exercises assigned to your group.
- Why such an exercise format?
  - More “Real World”: much of programming is maintenance of code *other* people have written.
  - You *must* read other people’s code to learn to program; otherwise its like a deaf person learning to speak.
  - We can demonstrate you worthwhile things to know on the side.
  - ... and it is easier for us to come up with good exercises...

## What *Attitudes* I Hope You'll Learn

- Fearlessness: there's much code you don't have to, or won't be able to understand.
  - The AI-code in this exercise, for example.
- Collaboration: with both past, current and future colleagues.
  - Past: Live with and learn from their code
  - Future: Write code that is easy for them to maintain

## ToroidToe

- Game of five-in-a-row on a  $6 \times 6$  grid wrapped onto a torus.
- The game is implemented in a simplistic WWW-server.
- *The Code* contains:
  - Networking: a UNIX daemon listening to a port, waiting for connections and requests
  - WWW: A little bit of URLs and HTTP — very typical string processing stuff.
  - AI: a rather respectable computer opponent.

ToroidToe - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:6666/.9F31B69.9.3.5

## ToroidToe, move 9

	O	O	O	X	
			O	X	
			X	O	O
	X	X		X	X
O	O		X		
	O	X			

I won!

[New game](#)

## Getting Started

- Download and unpackage as course guides suggest
- `make`
- Run in some machine  $X$  with `./toroidtoe`
- Point your browser to `http://X:6666/`
- Make moves by clicking on empty squares
  
- NOTE: No two processes can listen to the same port number (6666) in the same machine.
- NOTE: The port may remain reserved for a while after an unclean exit.

## net.c

- 121 lines, 80 SLOC
- A rather general-purpose UNIX network daemon event loop
  - Create a socket, bind it to a TCP/IP port, listen to the port, accept incoming connections.
  - Wait for incoming connections or data with `select`, pass the data to the given *callback*.

```
typedef int (*callback_t)(int fd, int n_bytes, char *bytes);  
void net_event_loop(callback_t cb);
```

## session.c

- 261 lines, 172 SLOC
- HTTP
  - `int session_cb(int fd, int n_bytes, char *bytes);`
  - Parse the HTTP request (to the extent necessary)
    - \* Serve GIF files from given directory by copying the file into the socket
    - \* Initial page
    - \* A move by the user
    - \* Other requests answered with “404 Not Found” page.

- Generate HTTP response (incl. HTML)
  - \* The playing area is a HTML TABLE containing hyperlinks which correspond to a move by the user

```
<TABLE><TR>  
<TD><A HREF="/.B40C89F.1.0.0"><IMG SRC="/pix/empty-white.gif"></A>  
<TD><A HREF="/.B40C89F.1.1.0"><IMG SRC="/pix/empty-white.gif"></A>  
...
```
- The URL's path contents *now*:
  - \* Session ID: a random number to distinguish players
  - \* Move number
  - \* X and Y of the previous move by user

- Most tasks affect this file
- You may, often must, change the format of URL path
- Useful resources:
  - \* <http://www.faqs.org/docs/htmltut/>
  - \* Especially <http://www.faqs.org/docs/htmltut/forms/>
  - \* <http://www.w3.org/Protocols/>
- Consider splitting the big function `session_cb`.

## main.c

- 88 lines, 54 SLOC
- The main program
  - Parse command line options with getopt
  - Call initialization functions of other modules
  - `net_event_loop(session_cb);`

## ai.c

- 656 lines, 461 SLOC
- A respectable computer opponent
- Except for lack of ASIC, techniques not very different from those that beat Kasparov
- This is a general overview: more information in literature. Few tasks affect this file.
- A recursive routine `alpha_beta_with_memory` searches the best moves for both sides up to a certain depth
- Threatening, forced or “natural” lines are searched even deeper

- A positional evaluation at the leaves of the search tree
  - Cross wins = 6000 pts, nought wins = -6000, drawn = 0
  - Uses a machine-generated pattern database in `patterns.h` (177150 lines!). For example the line

```
{ 1000, 2 }, /* '   Xxxx   ' */
```

tells that four crosses in a row with ample space around is worth 1000 points and it is a very forcing line that has to be checked by searching.

- A lossy hash table (*transposition table*) to avoid researching identical subtrees
- Another lossy hash table (`rote_learning_hash_table`) memorizes results of previous searches, so that if they occur in later searches, their worth is better known
- Positions are stored in hash tables in a compressed and canonified form

## Ok, Seriously...

...this exercise is artificial in the sense that you would use better existing solutions for `net.c` and parts of `session.c`, Apache for example.

But this would have made the exercise less thematic for the course.

## What *Knowledge* I Hope You'll Learn

- Typical industrial programming style (actually, much better 😊 )
- A typical program structure
- A little about sockets/network programming
- Event-loop and callback-based programming style
- A little about what goes under the hood of WWW
- String processing examples, maybe files
- A teaser into AI, algorithmics, etc.