Complex Strategy AI Research & Education
What is the State of the Art?
Board Game AI
Computer Game AI
What exactly is “complex”?
Complexity Criteria

- more than 2 players
- hidden Information
- large number of turns or real-time
- simultaneous hidden moves
- large situation space
- complex decision space
- randomness
How to develop AI for complex games?
Available Software

● commercial games
  - no programming access
  - short lifecycle

● open-source games
  - e.g. “Freeciv”
  - usually way too complex
  - no specific AI support
Create your own!
Requirements

- easy debugging & testing
- performance evaluation
  - simulation of multiple games
- comparison to competitors
  - requires a central website or server
- availability of AI modules
- game records for analyses
  - required for most learning techniques
The CoSAIR Approach

- based on existing browser-game
  - balanced game, mature code
- using mod_perl web server
- people can upload, execute, test, benchmark and compare their bots
- access to all CPAN AI::* modules
What is the game about?
Planets
Battles
Battles
Battles
Battles
Battles
How to develop AI for COSAIR?
A bot is a Perl class with methods for each decision.
package Cosair::AI::Bot::BrainBug;

use base ('Cosair::AI::Bot');
require Cosair::AI::OwnPlanet;

# called whenever queue runs empty
sub decide_planet_items {
    my ($self, $planet) = @_;  
    if ($planet->{can_build('starbase')) {
        return 'starbase';
    } else {
        return 'cruiser';
    }

}
This is executed each turn by the game engine
Bot Execution Code 1

# create a sandbox copy of the game state
my $control = create_control($bot_name);

$db->begin_work();

eval {
    my $bot = Cosair::AI::Bot->new($control);

    # operate exclusively on $control
    $bot->calculate_turn();

    # check actions and write to database
    process_actions($control);
}
if ($@) {
    # error in bot code or illegal action
    $db->rollback();
    log_error($bot_name, $@);
} else {
    # everything ok, commit actions
    $db->commit();
}

# log debug messages in any case
log_debug_messages($control);
Letting people execute code on my server???
Code Monitoring

• monitor code in regular intervals to avoid
  – infinite loops
  – memory leaks

• consider:
  – process time, not real time
  – process might grab several MB/s
use Time::HiRes qw(setitimer ITIMER_VIRTUAL);
use Devel::Mallinfo qw(mallinfo);

my $seconds_running = 0;
my $memory_offset = mallinfo()->{uordblks};

# return allocated memory in MB
sub allocated_memory {
    return
        (mallinfo()->{uordblks} - $memory_offset)
            / (1024*1024);
}
local $SIG{VTALRM} = sub {
    # infinite loop check
    $seconds_running += 1;
    die "exceeded time limit of 10s!\n" if ($seconds_running >= 10);

    # memory leak check, use MB as unit
    die "exceeded memory limit of 64MB!\n" if (allocated_memory() > 64);
};
# check each second of process time
setitimer(ITIMER_VIRTUAL, 1.0, 1.0);

# execute monitored bot code
eval $bot_code;

# stop checks
setitimer(ITIMER_VIRTUAL, 0);

if ($@) {
    # handle exceptions
}

Code Restrictions

- prevent execution of malicious code:
  - game information spying
  - game data manipulation
  - filesystem access
  - uncontrolled module usage
use Safe;
Opcode Restrictions

- control compiler-internal Perl opcodes
  - perldoc Opcode
- in our scenario:
  - allow basic perl constructs
  - allow “require”, “bless”, references...
  - deny any I/O
  - deny all system interactions
sub get_safe {
    my $safe = Safe->new('Cosair::AI');
    $safe->permit_only(qw(
        :base_core :base_mem :base_loop
        :base_math :base_orig require
    ));
    $safe->deny(qw(
        entertry leavetry
        dbmopen dbmclose
    ));
    return $safe;
}
Safe Compartments II

my $safe = get_safe();
# only non-lexical variables can be shared
our $result;
$safe->share('$$result');

$safe->reval(q[
  # only restricted opcodes compile
  $$result = 42;
]);
die "error in safe: @$" if ($@);

print $$result, "\n";
This is easy for simple code. How about using modules?
Safes & Modules

• A “require” in a safe
  - always refers to the "main" namespace
  - always compiles the module
    • including all of its dependencies
    • with the opcode restrictions

→ bypasses the namespace limitation!
Module Usage Summary

• modules with denied opcodes (case 1)
  – have to be required outside the safe
  – must not be required in the safe
  – thus have to be in the safe's namespace
  – have to be used in the safe without the namespace prefix

• modules without denied opcodes (case 2)
  – can be required in the safe
  – can be used with their full namespace
Importing Functions I

• importing functions inside the safe:

```perl
use Namespace::Module qw(sub_a sub_b);

$safe = Safe->new('Namespace'); $safe->reval(q[
  # module is compiled and accessible
  # but subs are not yet imported
  Module->import('sub_a', 'sub_b');
]);
```

→ does not work without changes
Importing Functions II

- reason: @ISA is evaluated at runtime and Exporter::import() is not visible

```perl
package Namespace::Module;
use Exporter qw(import);
our @EXPORT_OK = qw(sub_a sub_b);

→ importing in the safe works now!
```
That was easy.
- Too easy?
Inheritance

- runtime evaluation of @ISA makes inheritance very messy
- each class in the inheritance tree has to be modified for usage in the safe
Inheritance Code

```perl
require Namespace::BaseClass;
require Namespace::DerivedClass;

$safe->reval(q[
  my $object = DerivedClass->new();
  # adapt @ISA locally
  # relative to safe's namespace
  local @DerivedClass::ISA = qw(BaseClass);
  # now inherited methods can be called
  ...
]);
```

→ has to be done recursively!
External Classes I

• good news:
  – references blessed outside the safe can be used in the safe when shared

• bad news:
  – inheritance does not work of course

• even worse news:
  – @ISA cannot be adapted, it's out of scope!

• worst news of all:
  – most classes I need use inheritance
External Classes II

• possible solution:
  1. derive the class in question
  2. make the new class be an exporter
  3. export all class and object methods
  4. write a new wrapper class in the safe's namespace
  5. import all methods from the derived class
  6. hope that it works for your module...
And you use it exactly like this?
Practical Problem

• the safe is inefficient for trusted bots
• situation: evaluate an untrusted bot
  – playing a series of 500 games
  – with 500 turns in average
  – with 5 trusted bots as opponents
  – 1,500,000 safe executions
• the safe prohibits the use of some external classes
Practical Solution

- source code transformation on upload:
  - strip namespace prefix everywhere
  - replace “use” with import calls
  - remove “require” lines

- people can code as if there was no safe
- bots can be executed
  - in safe (with transformation)
  - outside safe (without transformation) and use all desired modules
That was it?
Ready, set, go?
Open Problems

• Security risks:
  – some crashable opcodes still allowed (sprintf, sort, ...)
  – instant memory mass-allocation still possible
• module usage restrictions (needs wrapping in safe's namespace)
• SQLite commands abusable (filesystem access)
Wishlist

• Safe with monitoring capabilities
  
  \$\text{compartment} \rightarrow \text{set\_time\_limit}(10.0)
  
  \$\text{compartment} \rightarrow \text{set\_memory\_limit}(64.0)

• Safe with more differentiated namespace visibility
What else?
Simulations

- schedule simulation runs
  - with defined priority
- daemon processes calculate individual games
  - uses nice
  - multi-processor capable
Other Languages

• OO architecture allows any classes with the specified methods to act as a bot

  ➔ use Inline;
  ▶ Safe compatibility?
  ▶ inheritance problems?
  ▶ external modules inaccessible

  ➔ only suitable for simple bots in education
Graphics from:

- http://www.rabbitooth.com/
- http://www.wallpapersweb.com/
- http://www.3dworldclub.com/
- http://www.imageshack.us/
- http://www.luckycastles.co.uk/
- http://www.st-v-sw.net/
The End

Thank you!
Any questions?