Game Development In Perl

Abram Hindle

Victoria Perl Mongers

abez@abez.ca

November 24, 2003

This Presentation

- What am I going to cover?
  - Arguments for using interpreted dynamic languages for game development.
  - Patterns Of Perl and Games
  - Libraries and Tools for Perl Game Development
  - Survey of Games implemented in Perl

This Presentation

- What am I not going to cover?
  - In depth MikMod
  - In depth SDL
  - In depth XS
  - Win32 support - There is some SDL_perl support
  - OSX support - There is some SDL_perl support
  - Anything In depth :-)

Our Problem

- What are problems in non-commercial game development?
  - Completion
  - Complexity
  - Garbage Collection
  - Integration of Mini or Interpreted Languages to allow for user extensible objects and AI.
  - Low level languages are used to solve very high level problems. (C/C++)
  - User extensibility.
  - Content is more important than Code.. Code is easier.

A Solution

- We could use interpreted / Dynamic Languages. But what are the disadvantages?
  - Slow
  - Poor Hardware Support
  - Difficulties getting ”low level” access.
  - Poor I/O
  - Lack benefits of static typing.

A Solution

- What are the advantages of interpreted and dynamic languages in relation to games.
  - Easy to program
  - Easy to change
  - Can be limited to a sub-domain (sand-boxed)
  - Generate Code on the fly (genetic algorithms)
  - Great for user defined objects and AI.
  - Code can be loaded at anytime.

A Solution

- For our interpreted language, why use Perl?
  - Fast interpreted language.
  - Mature
  - Great libraries and community support
  - Perl can be embedded
  - Perl can use C to call non-Perl libraries.
  - Adoption - many users know Perl and there is much documentation on learning Perl.

Game Patterns

- What are the components of Games?
  - Video system - sprites / 3d animation
  - Audio system - event sounds
  - Music system - background music
  - Input system - keyboard, mouse, joystick
  - Communication system - Networking, protocols etc.
  - Logic system - The game rules, user defined objects etc.
  - You can break out a lot of this into threads or an event based system.

Game Patterns

- What are the important parts of a game that people often forget?
  - Text/Fonts
  - Menu and GUI components
  - Content
  - Script-ability
  - Pausing
  - Loading and Saving State
  - Back-end tools to aid in content creation
  - ...

Game Patterns

- What does the main method of a game look like?
  - initialize
  - menu
  - run game loop
  - clean up and exit

Game Patterns

- What does the game loop look like?
  - Check for input and process - this includes the AI (it's best if your AI acts like a player rather than a separate subsystem).
  - Update game objects
  - Draw your screen
  - Play your sounds
  - Play your music

Game Patterns

- How do we use Perl when making games?
  - Perl calls C
  - C calls Perl
  - Perl acts as a client or a server
  - Tool implementation

Game Patterns

- Perl Calls C
  - Game written in Perl
  - Following the 90/10 optimization rule only small parts of the program really benefit from conversion to C.
  - We use the Perl to C interface “XS” to bind C code to Perl.
  - External libraries can be wrapped in C

Game Patterns

- C Calls Perl
  - Game Predominantly written in C (Or other interfacing language)
  - Perl runs the AI or the objects inside of the game.
  - Best for games that demand High Performance
  - User Perl to extend already existing games
  - This option probably gives the greatest performance
  - You can use Perl name spaces or Perl snippets

Game Patterns

- Perl as a client
  - Using a RPC or distributed object system (CORBA), Perl acts as a client.
  - Useful for AI clients.
  - Good for low bandwidth tasks (authentication)
  - Makes extension even easier and not restricted to Perl alone.

Game Patterns

- Perl as a server
  - Use Perl to run the game logic and the networking
  - Provides services to other servers (meta-server)
  - If your client have to be extremely optimized much of the time a Perl server will work just fine as the network I/O is the biggest bottleneck.
  - Many games are being designed as a client / server architecture thus these patterns are becoming more relevant.

Graphics

- How can we make GUIs and Graphics in Perl?
  - Gtk - GUI
  - Tk - GUI
  - Qt - GUI
  - FLTK - GUI
  - Wx - GUI
  - SDL - Graphics, 3D etc.

Graphics

- SDL is probably the best bet for Games
  - See graphics.pl
  - SDL::Surface and SDL::App are very easy to deal with.
  - Image Loading, surfaces, alpha channels, color models, full-screen are all handled by SDL.

Sound and Music

- How can we play music and sound in Perl?
  - SDL::Mixer - plays music and wav files, uses MikMod to play mods and xm files. MikMod provides much of the tracker functionality.
  - Midi::Music - plays midi music (if necessary)

Sound and Music

- SDL is probably the best bet for Games
  - See sound.pl
  - SDL::Mixer , SDL::Sound and SDL::Music are very easy to deal with.
  - Wave file loading, music file loading, multi-channel mixing, sound amplitude are all handled.
  - Non-blocking sound playing. Sound and music is played in the background.

Input

- SDL is probably the best bet for Games. SDL handles:
  - Keyboard (and special keys)
  - Joystick
  - Mouse
  - Easily handled through SDL::Event

Logic

- There are quite a few Perl modules for game logic.
  - Great for designing AI's for games or as an example how to create a sharing game state object.
  - Card Games, such as poker
  - Game state holders
  - Go
  - Chess
  - Games::*

Games In Perl

- Toad (Frogger)
  - http://www.foo.be/docs/tpj/issues/vol5_3/tpj0503-0014.html
  - 2048 Bytes (Original Frogger on the Atari 2600 was 4k)
  - Won prize in Obfuscated Perl Contest
  - use Tk;

Games In Perl

- Open Mortal
  - http://apocalypse.rulez.org/ upi/Mortal/
  - Animation, Sprites, Music, Sound
  - uses SDL, C, and Perl.
  - Perl is embedded
  - Perl is used to define the characters in the game. Character data is both data and code.
  - Good example of how to make the user defined objects actually user definable and dynamically loadable.

Games In Perl

- Perl FPS
  - http://bloodgate.com/perl/sdl/game.html
  - uses SDL and Perl
  - 3D FPS
  - In development
  - SDL::App::FPS - a framework for developing a FPS

Games In Perl

- Frozen Bubble
  - http://www.frozen-bubble.org/
  - Animation, Sprites, Music, Sound
  - Quite small (2000 Perl LOC, 500 C LOC) but the biggest Perl success story.
  - use SDL;
  - Great Example of the use of SDL – probably the best Perl reference.
  - Excellent example of what superior content can do for a game.

Get Help!

- Good Places to get help:
  - http://search.cpan.org/ - You can probably find what you're looking for
  - http://www.libsdl.org/ - SDL homepage
  - Perldocs For XS: perlembed, perlxstut, perlxs, perlcall, perlguts, xsubpp
  - Perldocs for: SDL, SDL::Mixer, SDL::App, SDL::Surface,...
  - http://www.frozen-bubble.org/ Frozen Bubble source code
  - There are very few sites dedicated to Perl and game programming. Look for other resources and try to apply them to Perl.
  - http://www.thomastongue.com/Code/SDL_Perl_MacOSX.html - MacOSX SDL Perl

Rant

- Problems with Game Development
  - Content is more important than code
  - Game-play is more important than performance or graphics
  - It is very hard to finish anything that is “Cutting Edge”
  - Commercial games are produced by a staff of full time specialized employees. It’s hard to compete at the same level.
  - Preoccupation with performance and optimization is unhealthy and counter-productive to making a game which people will actually play.

Conclusions

- – 2D games are still fun
  - – It is very hard to finish anything that is “Cutting Edge”
  - – The most important part of making a game is finishing
  - – Perl is appropriate for extending existing games
  - – Fast
  - – Well Supported
  - – Easy to code in
  - – Has reasonable level of adoption
  - – Don’t re-invent the wheel.

Code-Listing: sound.pl

```
use SDL::Mixer;
use SDL::Event;
use SDL::App;
use strict;
use Data::Dumper;

my $sdl_flags = SDL_ANYFORMAT | SDL_HWSURFACE | SDL_DOUBLEBUF |
     SDL_HWACCEL | SDL_ASYNCBLIT;

my $app = new SDL::App(-flags => $sdl_flags | 0, -title => 'SDL-Example'
     , -width => 640, -height => 480);
my $bg = new SDL::Surface(-name => "bg.jpg");
my $arect = new SDL::Rect(-width => $app->width, -height => $app->height
     );
$bg->blit($arect,$app,$arect);
$app->flip();#
my @img = ();
my @imgr = ();
my @maxx= ();
my @maxy = ();
for (1..3) {
```

```
        my $image = new SDL::Surface(-name => "$_.png");
        my $rect =  new SDL::Rect(-width => $image->width, -height =>
             $image->height);
        push @maxx,640 - $image->width;
        push @maxy,480 - $image->height;
        push @img,$image;
        push @imgr, $rect;
        print $maxx[$#maxx],"␣",$maxy[$#maxy],$/;
}

my $event = SDL::Event->new;
my $mixer = eval { new SDL::Mixer(-frequency => 44100, -channels => 2, -
     size =>
4096); };

my @keys = ('a'..'z','0'..'9','A'..'Z','!','@','#','$','%','^','&','(',
     ')');
my @sounds = ();
my $map = {};
foreach (@ARGV) {
        my $key = shift @keys;
        my $sound =  new SDL::Sound($_);
        push @sounds, $sound;
```

```
        $map->{$key} = $sound;
}
my $index = 0;
while ($event->wait()) {
       my $type = $event->type();      # get event type
        if ($type == SDL_KEYDOWN) {
                my $sym = $event->key_sym();
                my $key = chr($sym);
                print $sym," [$key]",$/;
                exit if $sym == 27;
                if (exists $map->{$key}) {
                        my $sound = $map->{$key};
                        $mixer->play_channel(-1, $sound, 0);

                        my $in = $index%3;
                        my $x = int(rand($maxx[$in]));
                        my $y = int(rand($maxx[$in]));
                        my $image = $img[$in];
                        my $drect = new SDL::Rect(-width => $image->width
                             ,
-height => $image->height, -x => $x, '-y' => $y);
                        $image->blit($arect,$app,$drect);
                        $app->flip();#
```

```
                    $index++;
                }
        }
        # ... handle event
        exit if $type == SDL_QUIT;
}
```

Code-Listing: graphics.pl

```
use SDL::App;
use SDL::Event;
use SDL::Surface;
use strict;
my $max = 3;
my $fullscreen = 0;
my $color = new SDL::Color ( -r => 0, -g => 0, -b =>0 );
my $sdl_flags = SDL_ANYFORMAT | SDL_HWSURFACE | SDL_DOUBLEBUF |
     SDL_HWACCEL | SDL_ASYNCBLIT;
my $app = new SDL::App(-flags => $sdl_flags | ($fullscreen ?
     SDL_FULLSCREEN : 0), -title => 'SDL-Example', -width => 640, -height
      => 480);
my @img = ();
my @imgr = ();
my $bg = new SDL::Surface(-name => "bg.jpg");
for (1..$max) {
        my $image = new SDL::Surface(-name => "$_.png");
        my $rect =  new SDL::Rect(-width => $image->width, -height =>
             $image->height);
        push @img,$image;
        push @imgr, $rect;
```

```
}
my @sprites = ();
for (1..10) {
        for my $i (1..$max) {
                my $image = $img[$i-1];
                my $imagerect = $imgr[$i-1];
                push @sprites,{
                        img=>$image,
                        imgr=>$imagerect,
                        x=>int(rand(640)),
                        y=>int(rand(480)),
                        maxx=>640 - $imagerect->width,
                        maxy=>480 - $imagerect->height,
                        velx=>(1-2*int(rand(1))),
                        vely=>(1-2*int(rand(1))),
                }
        }
}
my $arect = new SDL::Rect(-width => $app->width, -height => $app->height
    );
#my $irect = new SDL::Rect(-width => $img->width, -height => $img->height
    );
my $event = new SDL::Event;
```

```
for (1..2000) {
        $bg->blit($arect,$app,$arect);
        #$app->fill($arect,$color);
        #$color->r(($color->r + 1)%255);
        foreach my $sprite (@sprites) {
                my ($img,$x,$y,$velx,$vely,$maxX,$maxY,$irect) =
                @$sprite{qw(img x y velx vely maxx maxy imgr)};
                $x = $x+$velx;
                if ($x < 0) { $x = 0; $velx = -$velx; }
                if ($x > $maxX) { $x = $maxX; $velx = -$velx; }
                $y = $y+$vely;
                if ($y < 0) { $y = 0; $vely = -$vely; }
                if ($y > $maxY) { $y = $maxY; $vely = -$vely; }
                my $drect = new SDL::Rect(-width => $img->width, -height
                      => $img->height, -x => $x, '-y' => $y);
                $img->blit($irect, $app, $drect);
                @$sprite{qw(img x y velx vely maxx maxy imgr)} = ($img,$x
                     ,$y,$velx,$vely,$maxX,$maxY,$irect);
        }
        $app->flip();#
        $app->delay(1);
        if ($event->poll()) {
                my $type = $event->type();
```

```
                exit if $type == SDL_QUIT;
        }
}
```