Mono for Game Developers

Miguel de Icaza
miguel@xamarin.com, @migueldeicaza

Xamarin Inc
Agenda

• Mono in Games
• Using Mono for Games
• Performance
• Garbage Collection
• Co-routines, Asynchronous Programming
MONO IN GAMES
C#
Java
JavaScript
Ruby
Python
Visual Basic
F#
Sims 3

- Mixed Code:
  - C/C++ engine
  - C# scripting/AI
  - C# high-level

- Visual Studio + Mono

- X86, PS3, Xbox360
Bastion – on Google Chrome NaCl

- C# XNA codebase
- Originally on Xbox
- Ported to NativeClient
  - Mono
  - MonoGame (XNA)
- Mac, Windows, Linux
Pure C# - SoulCraft

- DeltaEngine
  - Pure C# engine
  - Open source
  - Android, iOS, Mac, Win
Unity 3D

- **Unity Engine**
  - C/C++ game engine
  - Embedded Mono
- **User code**
  - C# or UnityScript
  - Extends Unity itself

Shadow Gun, built with Unity
SecondLife

- Mono on the server
- Powers LSL scripts
- Nice 200x perf boost
- Code Injection
Infinite Flight

- Subject of the second part of this session
WHY MONO?
Because Life is too Short

• To debug another memory leak

• To track another memory corruption bug

• Because you deserve better
# The Quest for Productivity

## System Languages

**Pros:**
- Low-level
- Good control of hardware
- Typed
- Fast code

**Cons:**
- Easy to corrupt state
- Low productivity
- Crash often
- Complex for newcomers

## Scripting Languages

**Pros:**
- High-level, good productivity
- Easy to write
- Safe, prevent crashes
- Loosely typed

**Cons:**
- Poor control of hardware
- Slow (interpreted)
## John Ousterhout Scripting Quest

### IEEE 1998 Summary Paper

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Language 1</th>
<th>Language 2</th>
<th>Time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database application (Ken Corey)</td>
<td>C++ version: 2 months</td>
<td>Tcl version: 1 day</td>
<td>60</td>
<td>C++ version implemented first; Tcl version had more functionality</td>
</tr>
<tr>
<td>Computer system test and installation (Andy Belsey)</td>
<td>C test application: 272,000 lines, 120 months</td>
<td>47</td>
<td>C version implemented first; Tcl/Perl version replaced both C applications</td>
<td></td>
</tr>
<tr>
<td>Database library (Ken Corey)</td>
<td>C++ version: 2-3 months</td>
<td>Tcl version: 1 week</td>
<td>8-12</td>
<td>C++ version implemented first</td>
</tr>
<tr>
<td>Security scanner (Jim Graham)</td>
<td>C version: 3,000 lines</td>
<td>Tcl version: 300 lines</td>
<td>10</td>
<td>C version implemented first; Tcl version had more functionality</td>
</tr>
<tr>
<td>Display oil well production curves (Dan Schenck)</td>
<td>C version: 3 months</td>
<td>Tcl version: 2 weeks</td>
<td>6</td>
<td>Tcl version implemented first</td>
</tr>
<tr>
<td>Query dispatcher (Paul Healy)</td>
<td>C version: 1,200 lines, 4-8 weeks</td>
<td>Tcl version: 500 lines, 1 week</td>
<td>2.5</td>
<td>C version implemented first, uncommented; Tcl version had comments, more functionality</td>
</tr>
<tr>
<td>Spreadsheet tool</td>
<td>C version: 1,460 lines</td>
<td>Tcl version: 380 lines</td>
<td>4</td>
<td>Tcl version implemented first</td>
</tr>
<tr>
<td>Simulator and GUI (Randy Wang)</td>
<td>Java version: 3,400 lines, 3-4 weeks</td>
<td>Tcl version: 1,600 lines, &lt;1 week</td>
<td>2</td>
<td>Tcl version had 10 to 20 percent more functionality and was implemented first</td>
</tr>
</tbody>
</table>


[http://www.xamarin.com](http://www.xamarin.com)
John was always ahead of his time

• Professional workstations in 1998
  – SPARC, HP-PA

• Not achievable on PCs of the time
2000 – Desktop Development

• Building desktop apps with C and C++
  – Slow progress, error prone, frequent crashes

• Windows 2000 Requirements:
  – 133 Mhz or more
  – 64 megs for desktop, 256 for server

• Windows XP Requirements (one year later)
  – 233Mhz or more
  – 128 megs for desktop

• Development desktops at the time:
  – ~1Ghz speed
  – ~1 GB of memory
C# Introduced in 2000

• C# 1.0 was a Java-like system

• With many design fixes
  – 10 years of experience
  – Change defaults (all virtual, vs opt-in virtual)
  – Introduce structs (help GC, no boxing)
  – Direct access to native libraries (P/Invoke)
  – Delegates (foundation for lambdas)
Game Software Components

- **Display**
  - Rendering
  - Shading
  - Scene
  - Animation
  - Geometry
  - GUI

- **Simulation**
  - Physics
  - Collision
  - Particles
  - Terrain

- **Game Logic**
  - World rules
  - Enemy AI
  - User control
  - Camera
  - Behavior

- **Support**
  - Audio
  - Input
  - Networking
The Problem
Games are real-time programs

• 30 to 60 frames per second (0.016 seconds)
Problem: Scripting Is A Bottleneck
Gaming's Achilles' Heel

Display
- Rendering
- Shading
- Scene
- Animation
- Geometry
- GUI

Simulation
- Physics
- Collision
- Particles
- Terrain

Game Logic
- World rules
- Enemy AI
- User control
- Camera
- Behavior

Support
- Audio
- Input
- Networking

C/C++

C/C++

C/C++

Script

C/C++
Problem: Scripting Is A Bottleneck
Gaming's Achilles' Heel

Display
- Rendering
- Shading
- Scene
- Animation
- Geometry
- GUI

Simulation
- Physics
- Collision
- Particles
- Terrain

Game Logic
- World rules
- Enemy AI
- User control
- Camera
- Behavior

Support
- Audio
- Input
- Networking

C/C++

C/C++

C/C++

C/C++

C#
What C# Offers

• Close to native performance
  – 50%-90% of native performance

• Safe Execution Environment
  – With optional support to shoot yourself in the foot.
C# - An Evolving Language

- Managed Code, strongly typed
- Generics, Iterators, Lambdas
- Language Integrated Query, Functional
- Dynamic extensions
- Asynchronous Programming
USING MONO
Designing Mono Applications

• Provided:
  – C# Language
  – Base Class Libraries

• Not Provided:
  – User Interface, Graphics, Audio
  – These are all platform specific
Code Sharing and Native Experience

- C#: Plus ECMA languages
- Runtime
- OS
- Core Engine, Shared Logic, Business Logic
- .NET
- Windows, Xbox, WinPhone
- Mac
- iOS
- Mono
- Android

Not a comprehensive list
Code Sharing and Native Experience

Native UI APIs
- XAML
- XNA

C#
- Plus ECMA languages

Runtime
- Mono
- MonoMac
- MonoTouch
- MonoDroid

OS
- .NET
- Windows
- Xbox
- WinPhone
- Mac
- iOS
- Android

Core Engine, Shared Logic, Business Logic

Not a comprehensive list
Modes of Use

- Drive the application
- Scripting engine
  - Sandboxxed
  - Full access
Run on Mono

class AstroHunt : Game {

    static void Main ()
    {
        InitNetwork ();
        InitGraphics ();

        new AstroHunt ().Start ();
    }
}

Mono for Game Developers – AltDevConf 2012
http://www.xamarin.com
```
int main ()
{
    InitEngine ();

    domain = mono_jit_init_version ("myapp", "v2.0.50727");
    mono_add_internal_call ("GameObject::Move", game_object_move);
    mono_add_internal_call ("GameObject::Explode", game_object_explode);

    assembly = mono_domain_assembly_open (domain, "scripts.exe");
    StartEngine ();
}

void run_frame_scripts (void *script, void **params)
{
    MonoException *exception;
    mono_runtime_invoke (run_scripts_for_frame, script, params, &exception);
}

void game_object_move (GameObject *obj)
{
    // ...
}
```
Use Mono as a Library

Game Engine

Game Engine Libraries

Mono

Audio

Graphics

Game – Your C# Code
TIPS ON USING MONO
Two Code Generation Backends

Mono’s Native Backend
- Very fast codegen
  - .3 seconds bootstrap
- Not great code output
- JIT’s default engine

LLVM Backend
- Very slow codegen
  - 7 second bootstrap
- Great output quality
- Opt-in:
  - mono --llvm
Just in Time vs Ahead of Time

• Just in Time Compilation
  – Default Mode of Operation
  – Very fast at compiling code
  – Not great quality of code generation

• Ahead of Time Compilation
  – Mandatory on some platforms
    • PS3, XBox360, iOS
  – Can afford expensive compiler optimizations
Arrays Bounds Checking

for (int i = 0; i < 10; i++)
    mesh [i].x += delta;

Mono Runtime translates this to:

for (int i = 0; i < 10; i++){
    if (i < 0 || i > mesh.Length)
        throw new IndexOutOfRangeException();
    mesh [i].x += delta;
Disabling Arrays Bounds Checking

• Very unsafe
  – GC depends on system integrity
  – But admissible if no error ever found on testing

• We give you the tools to shoot your feet
  – mono -O=unsafe

• Ask your QA team
GARBAGE COLLECTION
Mono’s Garbage Collectors

• Boehm GC:
  – Traditional Mono GC
  – Mostly-precise, stack conservative
  – Scans everything on each GC

• Generational Collector (SGen)
  – New (default on Android)
  – Generational (Old generation, nurseries)
  – Copying (plus mark+sweep for large objects)
SGen

Nursery

- New objects
- Small size (4MB)
- Per thread regions
- Very fast collection

Old Generation

- Aged objects
- Slower collection
- Fixed or variable heaps
- Parallel collection
Garbage Collector determines when to run and release memory

- Heuristics are platform-specific
- GC.Collect() is the only deterministic option
Best Practices

• Pre-allocate major objects before Game Loop
  – Managed objects
  – Or unmanaged buffers
  – Try to only use the nursery (stay under 4M)
  – If you must collect, only collect the nursery:
    • GC.Collect (0) – Performs only a nursery collection
    • GC.Collect () – Performs a complete GC on the heap

• On Main loop:
  – Use structs instead of classes
Schedule GC Collection

Physics → Game Logic 1 → Game Logic 2 → Game Logic 3 → Network Checks → Audio Updates

Physics → Game Logic 1 → Game Logic 2 → Game Logic 3 → GC Collect → Network Checks → Audio Updates

GC.Collect (0)
Limit Collection to Nursery
Mono’s GC Thread Control

- Garbage Collection Stops all Mono Threads
- Non-Mono threads are not affected

Alternative:
- Use a Render Scene + Render Thread
- Like Apple’s CoreAnimation or Microsoft WPF
State-based programming

```csharp
public void AlienShip ()
{
    switch (state){
    case State.ActivePatrol:
        if (PlayerIsInRange){
            SetSprite ("attacking");
            state = State.Chase;
        } else if (ReachedEdge)
            state = State.PerformSpin;
    else if (--alert_state == 0){
        state = State.PassivePatrol;
    }
    break;
    case State.Chase:
        if (!PlayerIsInRange)
            state = State.ActivePatrol;
        else {
            direction = GetDirection (player);
            SetDirection (direction);
        }
    break;
    case State.PassivePatrol:
        // ...
    break;
}
```
Problems with Callbacks and State-Machines Systems

- Repetitive
- Cumbersome
- Error Prone
- Poor Error Propagation protocols/practices
- Life is too short
Co-routines

• Popular solution to simplify AI code
• Each Game Object has a script attached
  – Runs Game Logic
  – AI bits

• Many solutions
  – longjmp/setjmp for unmanaged code
  – Stack fiddling (Mono.Tasklets)
  – Interpreted languages with VM support
C# 5.0 and Async Programming

• Mono master has a complete C# 5 Compiler

• Turns repetitive callback-based async programming into linear programming
  – Compiler rewrites the code into a state machine
  – Tasks are scheduled on the main thread
  – Scheduling is customizable

• Originally designed for interactive UIs
Using Await

async void AlienShip ()
{
    while (true){
        while (PlayerIsInRange){
            await SetSprite ("attacking");
            direction = GetDirection (player);
            SetDirection (direction);
        }
        while (--alert_state > 0)
            await PassivePatrol ();

        if [ReachedEdge]
            await PerformSpin ();
    }
}
The Magic

• Await lets you write linear code

• Lets you focus on the problem
  – The compiler is at your service

• Microsoft conventions for responsive UIs:
  – If it takes more than 50ms, make it async
More on await

• await introduces a suspension point
  – Code returns to caller
  – Execution resumes after “await” instruction
  – Very cheap memory-wise

• Works with IO, Networking stacks, slow code
  – System.IO, System.Net, Database access
  – Slow processing: XML, Json data
  – Blends transparently with Threads on multi-cores
async Task<int> KillEnemiesInRange (IShooter source)
{
    List<Enemy> enemies;
    int casualties = 0;

    while ((enemies = GetEnemiesInRange (source)) != null){
        foreach (var enemy in enemies){
            if (!source.Alive)
                return casualties;

            await RotateTowards (enemy.Position);
            if (IsEnemyInRange (enemy)){
                while (enemy.Alive){
                    if (await Shoot (enemy).Power == 0){
                        await enemy.Destroy ();
                        casualties++;
                    } else {
                        await StartAnimation ("reload", delay=3.0);
                        if (enemy.Alive && Distance (source, enemy) > 0)
                            await MoveTowards (enemy.Position);
                    }
                }
            }
        }
    }

    return casualties;
}
Current Trends in Async Programming

Callback based

Where:
- GUI programming
- Scalable web servers
- Responsive mobile and desktop applications

```c
void DownloadTweets ()
{
    FetchUrl (tweetUrl, result => {
        tweetDb.Populate (JsonResult (result), () => {
            View.ReloadData ();
        });
    });
}
```
With some error handling.

```csharp
void DownloadTweets ()
{
    FetchUrl (tweetUrl, result => {
        if (result == null)
            View.InvokeOnMainThread (delegate {
                ShowError ("Could not download tweets");
            });
        tweetDb.Populate (ParseJsonResult (result), (error) => {
            if (error) {
                Tweet.UpdateLastRead (lastValidCode, errorPost => {
                    if (errorPost)
                        View.InvokeOnMainThread (delegate {
                            ShowError ("twitter is down");
                        });
                });
            } else {
                View.BeginInvokeOnMainThread (delegate {
                    View.ReloadData ();
                    lastValidCode = currentCode;
                });
            }
        });
    });
}
```
C# 5.0 Async Support

```csharp
async void DownloadTweets ()
{
    var tweets = await FetchUrl (tweetUrl);
    if (DownloadTweets == null){
        ShowError ("Could not download tweets");
        return;
    }
    if (!await tweetDb.Populate (ParseJsonResult (result))){
        if (!await Tweet.UpdateLastRead (lastValidCode))
            ShowError ("Twitter is down");
    } else {
        View.ReloadData ();
    }
}
```
Q&A

• Mono, ISO Standard, C# Async
  – http://www.mono-project.com

• Xamarin, Mono on iOS, Android:
  – Discount for AltDevConf attendees:
  – http://www.xamarin.com/altdevconf

• Contact:
  – miguel@xamarin.com, @migueldeicaza

• Resources:
  – @MonoGameTeam, @Unity3D

• Live Chat on IRC: irc.gnome.org
  – #mono, #monotouch, #monodroid, #monogame
Iterators

• C# compiler provided assistance
  – Built on top of C# IEnumerable
  – C# compiler rewrites iterators into state machines

• Developers build on top of conventions

• Unity3D uses this approach

• Open Source Iterator game framework:
  – http://mjhutchinson.com/journal/2010/02/01/iteratorbased_microthreading
public IEnumerable AlienShip ()
{
    while (true)
    {
        while (PlayerInRange)
        {
            SetSprite ("attacking");
            yield return 0;
            direction = GetDirection (player);
            SetDirection (direction);
        }
        while (--alert_state > 0)
        {
            // keep looking for player
            yield return 0;
        }
        if (ReachedEdge)
        {
            PerformSpin ();
            yield return 0;
        }
    }
}
Mono.Tasklets

• Pros:
  – No need to rewrite code
  – You can suspend execution/resume without new conventions.

• Cons:
  – Not available on every platform – Stack Fiddling
  – Does not work with Mono’s new Precise GC
  – In particular, won’t work with Microsoft .NET
Not a comprehensive list

GAME ENGINES USING C#
• Commercial Engine
• Very extensive support:
  – Consoles: XBox360, PS3, Wii
  – iOS, Android
  – Mac, Windows
  – Google Native Client
  – Flash target
MonoGame – Open Source XNA

• Open Source XNA implementation
  – Currently 2D-based
  – 3D support coming

• Runs on many platforms:
  – iOS (iPhone, iPad)
  – Android (phones and tablets)
  – Linux
  – Mac
  – Windows
Delta Engine

- Open Source Game Engine
- Written 100% in C#
- Runs on:
  - Android
  - Windows Phone
  - iOS
  - Mac
  - Windows
Axiom

- Open source
- Based on the OGRE C++ Engine
- Windows, Linux
- XNA, DirectX and OpenGL support
F# - Fascinating Language

- [http://sharp-gamedev.blogspot.com/](http://sharp-gamedev.blogspot.com/)
  - Blog tracking the experiences of game development using F#

- F# introduced Async
  - Later adopted by C#
Architecture

• Computer Architecture – A Quantitative Approach

• Unix Systems for Modern Architectures
  – It says “Unix”
  – But applies to low-level systems engineering
  – Caches, MMUs, performance
  – Hardware Architectures design