The MAME story:

FROM C TO MODERN C++

MIODRAG MILANOVIĆ

Who am I ?

Working as software developer from October 2000

Experience in C, C++, C#, Java, …

Software architect in Levi9 – Serbia

From April 2012 coordinator of MAME project

What is MAME ?

Multiple Arcade Machine Emulator

Nicola Salmoria started project 1997

MESS as sister project

Preservation of software

Accuracy over performance

Who we are ?

About 50 active developers

Over 200 contributors

Team contains :

- experienced developers (gaming and not gaming related)
- emulation enthusiasts

Community:

- Developers of different experience
- Software dumpers
- Documentation acquirers
- Testers

Misconception about MAME

Made for free playing games

Is game itself

Way to sell new arcade cabinets

Perfect solution for all emulation

Platform for enhancing games

Why preservation of software is important ?

Companies do not backup all their software

Things get lost

Storage mediums are unreliable

It is always easy to find just “good” and “nice” software

It is not possible to buy some software for bit older platforms

What we do not do ?

Do not emulate recent hardware (if not permitted by author(s))

Do not support recent software unless permitted

Do not try to improve how things look and work

Why do we do it ?

"We do these things not because they are easy, but because they are hard“

-John F. Kennedy, 1962

How modernization started ?

Plain C project till February 2009

Aaron Giles started conversion to C++

2015 going modern C++

Why converting to C++ ?

Was quite hard to understand code even for existing developers.

Learning curve was bad, so we could not attract new developers.

Adding new functionality was hard, since it side effects were not clear.

Lot of global variables making emulation reuse that part of code impossible.

We wish to have “code as documentation” approach.

Code reuse was not clear.

Global symbol pollution was high.

First steps

Compile your C code as C++

Treat warnings as errors

Use multiple compilers on multiple platforms

OOP

Recognize classes and objects in your code

Recognize connections between them

Object oriented is great and natural way of documenting

Do not create over-engineered model of classes

Express your thoughts

First problems

Global variables

Large number of macros

No tools to help this process

Enforcing coding conventions

Manual labor

Team effort, but keep group working on conversion close

Remove all deprecated code once you remove their usage

Doing one small change you will end up redoing large portion of code

Keep track of changes

Clean/reformat your code

Automatization

Try using REGEX when applicable

Clang-tidy for modernization

- **modernize-use-nullptr**
- **modernize-use-override**
- **modernize-use-using**
- **modernize-use-default**
- **modernize-use-bool-literals**
- **modernize-use-auto**
- **modernize-make-unique**

Commercial tools

ReSharper C++ / Visual Studio 2015

- Do code analysis part by part
- Document your code where applicable
- Place TODOs

PVS Studio

- Static Code Analyzer for C, C++ and C#

Small example (1/2)

```
// license:BSD-3-Clause
// copyright-holders:Robbbert

#include "bus/rs232/rs232.h"
#include "cpu/s2650/s2650.h"
#include "machine/terminal.h"
#include "imagedev/snapquik.h"


class pipbug_state : public driver_device
{
public:
    pipbug_state(const machine_config &mconfig, device_type type, const char *tag)
        : driver_device(mconfig, type, tag),
        m_rs232(*this, "rs232"),
        m_maincpu(*this, "maincpu")
    {
    }

    DECLARE_WRITE8_MEMBER(pipbug_ctrl_w);
    required_device<rs232_port_device> m_rs232;
    required_device<cpu_device> m_maincpu;
};

WRITE8_MEMBER( pipbug_state::pipbug_ctrl_w )
{
// 0x80 is written here - not connected in the baby 2650
}

static ADDRESS_MAP_START(pipbug_mem, AS_PROGRAM, 8, pipbug_state)
    ADDRESS_MAP_UNMAP_HIGH
    AM_RANGE( 0x0000, 0x03ff) AM_ROM
    AM_RANGE( 0x0400, 0x7fff) AM_RAM
ADDRESS_MAP_END
```

Small example (2/2)

```
static ADDRESS_MAP_START(pipbug_io, AS_IO, 8, pipbug_state)
//  ADDRESS_MAP_UNMAP_HIGH
    AM_RANGE(S2650_CTRL_PORT, S2650_CTRL_PORT) AM_WRITE(pipbug_ctrl_w)
    AM_RANGE(S2650_SENSE_PORT, S2650_SENSE_PORT) AM_READNOP // this has to return zero or the parameter to write_sense is ignored
ADDRESS_MAP_END

/* Input ports */
static INPUT_PORTS_START( pipbug )
INPUT_PORTS_END

static DEVICE_INPUT_DEFAULTS_START( terminal )
    DEVICE_INPUT_DEFAULTS( "RS232_TXBAUD", 0xff, RS232_BAUD_110 )
    DEVICE_INPUT_DEFAULTS( "RS232_RXBAUD", 0xff, RS232_BAUD_110 )
    DEVICE_INPUT_DEFAULTS( "RS232_STARTBITS", 0xff, RS232_STARTBITS_1 )
    DEVICE_INPUT_DEFAULTS( "RS232_DATABITS", 0xff, RS232_DATABITS_7 )
    DEVICE_INPUT_DEFAULTS( "RS232_PARITY", 0xff, RS232_PARITY_EVEN )
    DEVICE_INPUT_DEFAULTS( "RS232_STOPBITS", 0xff, RS232_STOPBITS_1 )
DEVICE_INPUT_DEFAULTS_END


static MACHINE_CONFIG_START( pipbug, pipbug_state )
    /* basic machine hardware */
    MCFG_CPU_ADD("maincpu",S2650, XTAL_1MHz)
    MCFG_CPU_PROGRAM_MAP(pipbug_mem)
    MCFG_CPU_IO_MAP(pipbug_io)
    MCFG_S2650_FLAG_HANDLER(DEVWRITELINE("rs232", rs232_port_device, write_txd))

    /* video hardware */
    MCFG_RS232_PORT_ADD("rs232", default_rs232_devices, "terminal")
    MCFG_RS232_RXD_HANDLER(INPUTLINE("maincpu", S2650_SENSE_LINE))
    MCFG_DEVICE_CARD_DEVICE_INPUT_DEFAULTS("terminal", terminal)
MACHINE_CONFIG_END


/* ROM definition */
ROM_START( pipbug )
    ROM_REGION( 0x8000, "maincpu", ROMREGION_ERASEFF )
    ROM_LOAD( "pipbug.rom", 0x0000, 0x0400, CRC(f242b93e) SHA1(f82857cc882e6b5fc9f00b20b375988024f413ff))
ROM_END

/* Driver */

/*    YEAR  NAME    PARENT  COMPAT   MACHINE    INPUT    INIT    COMPANY   FULLNAME       FLAGS */
COMP( 1979, pipbug,  0,       0,     pipbug,    pipbug, driver_device,    0,  "Signetics", "PIPBUG", MACHINE_NO_SOUND_HW )
```

Delegates

```
class MyClass {
public:
        MyClass() { }
        virtual ~MyClass() { }
        virtual void docount(int) {  }
};
```

```
typedef delegate<void(int value)> callback_delegate;

MyClass mc;

callback_delegate md = callback_delegate(FUNC(MyClass::docount), &mc);
```

Why we needed delegates?

Providing callback functionality between various objects

Late binding (resolving objects referenced in runtime)

Minimal cost (using method function pointers)

Implemented in period of using C++ 98

Make your code public

Better feedback from users

Commits become better since people are aware more are looking at their work

More people get interested in project -> more pull requests

Do not use private repository sites to distribute your code

GIT over SVN

Why join open source project?

Share your ideas

Experiment

Improve your knowledge

Knowledge gained during work on open source projects help you do your regular work.

Meet more people, learn from them.

3rd party libraries and tools

BGFX – Branimir Karadžić

LUA – PUC Rio

RapidJSON – Milo Yip

GLM – GL Math

BGFX (1/2)

Cross-platform, graphics API agnostic, "Bring Your Own Engine/Framework" style rendering library.

Supported rendering backends:

Direct3D 9

Direct3D 11

Direct3D 12 (WIP)

Metal (WIP)

OpenGL 2.1

OpenGL 3.1+

OpenGL ES 2

OpenGL ES 3.1

WebGL 1.0

WebGL 2.0

BGFX (2/2)

Supported platforms:

Android (14+, ARM, x86, MIPS)

asm.js/Emscripten (1.25.0)

FreeBSD

iOS (iPhone, iPad, AppleTV)

Linux

MIPS Creator CI20

Native Client (PPAPI 37+, ARM, x86, x64, PNaCl)

OSX (10.9+)

RaspberryPi

SteamLink

Windows (XP, Vista, 7, 8, 10)

WinRT (WinPhone 8.0+)

Supported compilers:

Clang 3.3 and above
GCC 4.6 and above
vs2008 and above