Mastermind Manual

Mastermind developement process description

Moez Bouhlel
Firas Chaaben

Table of Contents

1 Project Presentation

This project is open source Mastermind Game implementation on C. It's customizable with mutli-platform support. Currently, support for changing number of holes, colors and guesses is supported. Support for enabling/disabling remise is available. Mutli use support is limited to changing session account name and saving each score with its account name. Top 20 scores are saved. Support for saving and restoring the session is included.

Booth command line interface and graphical user infterface are implemented. Various configuration options are implemnted on booth build time and runtime. Build time options are available on `config.h.in` and `CMakeList.txt` files. Runtime options are descripted on `doc/config.md` file and man:mastermincli(1). The command line interface support internationalization, booth short and long options and internal command auto-completion.

The graphical interface is based on SDL2.

Multi platorm support is implemented as follow:

- CLI: Linux, Windows, Mac OS X
- GUI: Linux, Windows, Mac OS X, iOS, Android, Web

2 C Standards

While writing the code, we used C standard API and syntax to easily support multi compilers and platforms and to simplify the code. Currently, mastermind code base compiles well on the five main compilers (Visual Studio C compiler, GCC, Clang, AppleClang, Intel Compiler) with at least six offical supported platforms.

2.1 Standard API

We made use of C89 and C99 standard API on first class with some use of POSIX standard APIs and fallback to MS Windows specific functions when no standard alternative is provided on Windows.

2.1.1 C89/C99 APIs

- stdio.h: fopen, fclose, printf, fprintf, scanf, fscanf, FILE, ...
- time.h: time
- stdlib.h: malloc, free, rand, srand, ...
  - getenv: get environement variable
  - strtol: convert string to long int
- string.h: strcmp, strlen, ...
- errno.h: errno (last system call error value)
- stdin.h: uint8_t (force variable memory size to be 8 bits)
- assert.h: assert

2.1.2 POSIX APIs

- unistd.h: mkdir (make directory with specified permission)
- sys/stat.h: access (check application access permission to specified path)
- sys/utsname.h:
  - uname: get system uname value
  - struct utsname: system uname representation struct
- string.h: strndup (deplicate string on new malloced memory with n char at most)
- stdlib.h: srandom, random (POSIX rand more secure rand alternative)

2.1.3 WINDOWS APIs

- windows.h: CreateDirectory (create direcotry)

2.2 Syntax and Types

- enum: when defining several related constants.
- struct: when defining several related variables.
- union: when defining several related types and only one to use.
- #define
  - define a standalone constant

  - define a macro (inlined function without call overhead)
  - Include Guards to not include header file more than once
- Conditionals (build time conditions to only include code needed on target platform or selected features)
- Bitwise operators (flags support to pass more than a state on same variable)

3 Third party libraries

To improve software fonctionality and protablity, we used third party libraries on booth mastermind user interface front-ends implementations.

3.1 Command line interface

Command line interface applications are very common on Unix systems. Most of cli applications use a set of common libraries (booth standards and third party libraries) to provide a minumal level of usability. We made use of the most common libraries.

3.1.1 Readline

Readline is part of the bash project. It provides a system independent powerful command line auto-completion system. We used Readline library to provide command completion on mastermindcli internel command line implementation. By using the Readline library, we got more extra features like bash based shortcuts (ctrl + w to remove laste word, ctrl + l to clean screen,...).

3.1.2 Gettext

Gettext is the GNU implemetation of POSIX localization standard and it's available on all Unix systems. It provdes an easy solution to support internationalization. Gettext GNU implementation comes with a set of command line tools to extract strings from source files, generate/update translation files and compile them.

3.1.3 Getopt

Getopt Library is a POSIX standard to provide a common solution to implement command line based arguments. POSIX standard only supports short options, we used GNU implementation to support long options too.

3.2 Graphical user interface

3.2.1 SDL2

There are many third party GUI libraries with different goals and technologies:

3.2.1.1 Widget Toolkits

They are the common choose for desktop applcations. They came with a set of desktop-oriented widgets (menu, buttons, text input,...). Mobile and Tablets based platforms have their own implementations too.

Examples: GTK, QT, Win32, Cocoa

Benefits:

- simple
- many predefined widget

Disadvantages:

- lot of unneeded features

- platform dependent
- does not work well on mobile/tablet platforms

3.2.1.2 Game Engnies

Game engines reduce the need to write a lot of code and remove the need to use platform specific APIs.

Benefits:

- a lot simpler
- predefined graphical object
- a lot of visual effects
- multi-platform support

Disadvantages:

- even more unneeded features
- very slow
- long execution time
- less C, more higher level scripting languages (Lua, python, C#, ...)

3.2.1.3 Context/Window Toolkits

Context toolkits are a set of APIs to provide a drawing surface and drawing functions. Context/window toolkits provide a set of APIs to open windows and handle events too. Many third party libraries provide a 2D and 3D drawing APIs.

OpenGL is the standard API for multi-platform 3D drawing. 3D contexts/window toolkits provide an OpenGL implementation with system integration (e.g: GLFW)

Benefits:

- all the power of 3D

Disadvantages:

- 3D is much more than we need or a simple game
- C + GLSL (OpenGL Shader Language: a scripting language for 3D drawing)

Most common multi-platform 2D drawing APIs are SDL and Cairo.

Cairo is a very powerful 2D library, it's used on many applications (Firefox, Google Chrome) and as a backend for many toolkits (GTK, PDF libraries,...).

Benefits:

- powerful 2D design API plus font support
- clean and stable API
- it has many backend: OpenGL, PNG, PDF, X11, ...
- It may be included as C++17 standard API even it's written on C.

Disadvantages:

- It's just a Context toolkit, it has neither event handling support nor window management support

SDL2 is a 2D context/window toolkit. It's the de facto standard on game developement on Unix systems. And it's used by many popular Game engines.

Benefits:

- Support almost every platform (win32, Linuw, Mac, iOS, Android, Blackberry, Web, BSDs, ...)
- has many extensions to provide different functions (net, font, music, ...)
- support tablets/mobile platform out the box

Disadvantages:

Well nothing for our simple use case.

We choosed SDL2 as Mastermind GUI backend.

3.2.2 Gettext

We used gettext library on GUI version too to support localization.

4 Developement Tools

While developing the code, we used serval tools.

4.1 Git

Git is the most used version control system. It supports distributed development which means that developers do not need a network access to the remote repository to access to its history or commit changes. It allows many developers to make changes (patchs) on the same code easily.

A demo of commands to change current direcotry to a repository and commit current changes.

```
git init
git add --all
git commit -m "First commit"
```

To host our repository online, we used the popular git repository hosting service Github.

4.2 Build System

There are many build systems for C based projects:

4.2.1 Makefiles

Makefile is part of POSIX standard. It's a text file containing a list of commands (rules) to execute. It's based on depencedy tree model where every rule has it's depencedies and it's generated target. Developing on Makefiles is easy but not perfect when targeting many different platforms.

4.2.2 Unix configure script

An alternative solution is to use Unix Configur scripts. It's an executable shell script designed to aid in developing a program to be run on a wide number of different computers. It generates needed Makefiles to build the software on the target platform. It's the standard way to build softwares on Unix systems and it's well documentated. The draw back is that MS Windows system does not support shell scripts natively and third party tools are needed to add Unix shell support. To build a project using Unix configure script:

```
./configure
make
sudo make install
```

4.2.3 CMake

CMake is a cross-platform build system. It is a family of tools designed to build, test and package softwares. It's one of top used build systems including Makefiles and Unix Configure script. CMake is our Unix configure alternative choose on Mastermind project. It supports windows out the box without any third party tool. It also allows generating workspace/project files to many target IDEs including Visual Studio, eclipse and Xcode. The draw back of using CMake is the need to learn its scripting language and it lack of support of Android and iOS as targets. To build a project using CMake:

```
mkdir build && cd build
cmake ..
cmake --build .
```

4.3 Doxygen

Doxygen is the de facto standard tool for generating internal documentation from annotated C++ sources. It also supports many other languages out of the box including C, Java, Python,... It can generate documentation from the code comments on many formats including HTML based documentation, PDF documents, simple Plain Text files, Unix man pages, RTF, XML, Devbook,... Developers just need to well document their code on a standard way, either using javadoc style or Qt style.

4.4 Unix man page

Man page is the standard way to document programs and standards on Unix systems. Man page manuals are split into serval numbered sections, mainly:

- 1 - General commands
- 2 - system calls
- 3 - Library functions
- 4 - special files
- 8 - system administration commands and daemons.

Man page uses its own syntax. Each Man page contains serval sections: NAME, SYNOPSIS, DESCRIPTION,... We used Man page to document the command line version of the project and its configuration file options.

4.5 Texinfo

Texino is a the prefered way to generate projects manuals on GNU systems. Its sytax language is based on Latex sytax. It allows to generate documentation on many formats including HTML pages, PDF, DVI, Info, Plain Text,...

4.6 GNU Standards

Our developement repository struct is based on GNU Standards recommanded struct with some modifications:

- `src/` directory containing the code source.
- `po/` directory containing the tansalations.
- `ide/` directory containing IDEs specific files.
- `doc/` directory containing documentation files (we also included man page file here)
- `res/` directory containing resources needed on runtime (icons, fonts,...).
- `ext/` direcotry containing third party files needed on build process on some platform (compiled libraries for MS Windows, iOS and emscripten, build files for android based on SDL2 android project files).
- `CMake/` directory containing cmake files.

- `configure.h.in` file to easily change build time configuration options.

The generated linux build is also based on Linux filesystem hierarchy specification.

- `bin/` directory containing the excutable binairies.
- `share/applications` containing the desktop file
- `share/man/` containing the man page file.
- `share/icons` containing the project icons on diferent sizes.
- `share/info` containing Info files.
- `share/fonts` containing needed fonts
- `share/locale/` containing the localization files.
- `share/doc/mastermind` directory containing extra documentation files.
- `share/bash-completion` containing the bash completion file.
- `share/zsh/site-functions` containing zsh completion script.

On Linux systems, runtime variable files path are choosen based on XDG standard.

- Configure file direcotry: `$XDG_CONFIG_HOME/mastermind/`
- Score and store files direcotry: `$XDG_DATA_HOME/mastermind/`

When XDG based directories are not found or not accessible, we use the Unix preferred way (`~/.mastermind/` directory to store all files).

4.7 Bash and Zsh completions

Shell completion is a very powerful feature on Unix systems. We wanted to support shell completion for the two main shell implementations on Unix systems. Bash completion provide all needed arguments and configuration options completion. Zsh completion system is more powerful than bash own completion system. We supported Zsh extra features to provide a more clean and better documentated completion.

4.8 Code Formating

Code formating is a very important part to proide a cleaner and easy to read code. Many projects have their own code styling specifications. Most popular code styling standards are: Google, GNU, Visaul Studio and Linux standards. We prefered Linux code standards because it provides a very clean code on its huge code base. We used a LLVM clang-format formating tool to help fixing styles issues.

Index