Ultimate Texas Hold'em

Final Report

By:

Adedayo Ajbage

Brad Tissi

Elliot Gross

Ryan Alyea

This is a screen shot from Ultimate Texas Hold'em

Author:
Elliot Gross
(Emg2m9@umr.edu)

Table of Contents

Introduction:

Texas Hold 'em as of late has become a very popular game, now that is featured not only in casinos, but played by many people all over the world even in their own homes. It is a game that has swept the nation and the world mainly through it's showing on television through various tournaments and programs such as The World Series of Poker (http://www.worldseriesofpoker.com/), World Poker Tour (http://www.worldpokertour.com/), Celebrity Poker Showdown (http://www.bravotv.com/Celebrity_Poker_Showdown/), just to name a few. It has easily grown to be the most popular poker variant of this time period as we know, and will continue to grow in popularity as long as Poker remains alluring.

Purpose:

The purpose of Ultimate Poker Showdown is to have a fun and entertaining game of one of today's most popular poker variants that you can not only play with yourself, but also up to five of your friends on not only a computer (through the use of VisualBoy Advance), but through an actual Gameboy Advance as well. Furthermore, this program is created to help the players get better and fine tune their skills so that they play well in a live game of Texas Hold'em as well.

The Solution:

The beginning of the solution must begin with a certain number of players, based upon a selection from a menu, each of which have a certain amount of money, who all will play the game together. Next, we need to be able to define all of the rules of Texas Hold'em as well as essentially define what a deck of cards is, as well as a method for shuffling the deck as well as dealing random cards. Next, the program will deal out two cards to each of the players, be able to take bets from each player, and then at the end of the hand deem who the winner of the hand is and give each player the correct amount of money that is due. The computer artificial intelligence must also simulate the actions of a human player and be limited to how a human player should act. For example, the computer players must not know what the human player or

the other computer players are holding. Additionally, a system for AI for the computer must be developed in order to make the game as realistic as possible.

Recommended hardware:

As mentioned in the End User Guide, the current system requirements/ recommended hardware are either: One Nintendo GameBoy Advance (regular/SP/Micro), or one Nintendo GBA Player for Gamecube, or one Nintendo DS (regular/Lite) if you wish to play Ultimate Texas Hold'em on a system other than the computer. However, if you wish to play Ultimate Texas Hold'em on a Windows based machine it is recommended that you have atleast a 1 GHz+ CPU and 128 MB of RAM, as well as a Keyboard. For specific requirements please see http://vba.ngemu.com/faq.shtml for further information on VisualBoy Advance and their requirements.

Installing and Running:

On Windows and Linux:

As mentioned in the End User guide (see Zip file), to install on your system (Linux, Mac OS X, or Windows) Copy the appropriate executable of visualboyadvance.exe based on your operating system found here: http://vba.ngemu.com/downloads.shtml ,and GBATest.gba on to some folder on your hard drive that is appropriate. Then once that is completed Open visualboyadvance.exe and go under file->open GBATest.gba and the file will be launched.

On Gameboy Advance:

Please refer to your flash cart's manual on how to put the game on it. Most require you to move to it via special software. If the cartridge uses separate media such as Secure Digital or CompactFlash cards, you may need to patch the ROM before running it. To run, just turn the device on (and select

GBA cartridge if need be.) If your flash cart has a multiROM booter, you need to select the ROM file.

Compiling:

One important thing you need to know is that that you need an ARM compiler. This project used the devkitArm package from devkitPro (http://devkitpro.org), a freely available package that includes a copy of GCC 4 which compiles ARM binaries and utilities to convert them into .gba ROM files. If you are running a Windows installation, you must install Cygwin (http://cygwin.com). There's also an installation binary available for Windows as well. If you are on another operating system, please read the instructions on setting your command-line variables. Please install the libGBA that comes with it.

This project was compiled using devkitPro's provided Makefile, which provides an easy way to quickly generate the .gba ROM file from source by running the commands required to do so.

Note that the source does make hardcoded reference to libGBA files to a very specific location. When compiling your own version, please update those references.

Design Constraints:

To begin, there must be only one copy of each card. For example, you do not wish to deal out two King of spades to two different players. Another constraint would be the computer artificial intelligence. For example, which hands do the computer players actually play, what is the percentage of the time they will bluff, how much will they bet on a given hand, etc.

Now that we are using the model of the Gameboy Advance as the platform the graphics have improved greatly and many more options were present. As a design consideration we must also design what the score for the artificial intelligence is (hand strength or what the hand is) as well as decide what defines a win based on the odds.

The sound for the program is also fairly elementary because of the complexity of such a task given the amount of time. However, the most important design consideration/constraint has been time because despite having quite a while, with our knowledge of C++ programming this far it is important to note that we can only accomplish so much.

Source Code:

The program was written specifically for the GameBoy system. There is only 256 kilobytes of memory available, so there are a lot of functions by constant reference to save memory.

Although not written in the code, GFX, Sound, and TexasRules are meant to be singleton. There should only be one of them in a program. Multiple GFX instances can cause havoc.

It's also important to note that all hardware memory (such as DMA memory moving or the location of a sprite) can be modified by anyone at anytime. Although the classes only modify what they say they are (ie. GFX only modifies the Graphics) it's not hard to have a stray pointer or a bad FOR statement messing things up.

Speaking of modifying the hardware, you may notice direct hardware edits, such as "*(u16*)0x04000000 = …" which direct edits the memory at that address. (That particular address is the main video arguments such as what graphics mode the GameBoy is running in.)

Conclusion:

This project was born due to the need of something challenging, and it clearly was. However, it has been conquered, and a working program runs on the hardware it was designed for. It's really awesome to see the program running on a limited system and still run at full speed.

The original Texas Hold'em from the previous design project, although painstakingly classified, still exists here. The basics were redesigned for an OOP-based system.

Obviously, the hardest part was the video core. It took a couple of weeks of dredging through obscure documentation to understand exactly what happens inside of a GameBoy. The biggest surprise was how much the internal video chip took care of everything. All the designer had to do was to move the correct image (from a correct location) to the correct location in the video memory.

Additionally, the random number generator was hard to figure out due to the GameBoy not keeping track of any form of time. That part of code was designed by other GameBoy programmers. The obvious problem with that code is that the random seeder starts with the same few numbers.

Every good has its bad. The obvious ugliness demands an art director for project bigger than this due to its complexity. However, the worst design decision was the use of sprites/objects for text instead of writing them directly onto the screen. It causes the obvious textual errors when playing the game for a long time.

Overall, the project is a huge success because the system limitations were harsh.

Design Notes:

The Sound system, although simplistic in code, was based on a large debate on how sound should run. Eventually we decided to use an external library, thus the logo at the beginning.

There are no warnings when compiling the code, so if you receive one (if you dare to try to compile in the first place), it's due to missing files/libraries.

Due to the weird class system, all operator<< operations are found in the global.h file rather than their own class.

Resources:

gameboy advance developers forum
http://gbadev.org

devkitPro
http://devkitpro.org

Google Image Search
http://images.google.com

UML and Algorithms:

Please see the Design Documents Folder.