Top Level Design for CS Contra

By: Taha Mohamedali (tmohamed)

Introduction

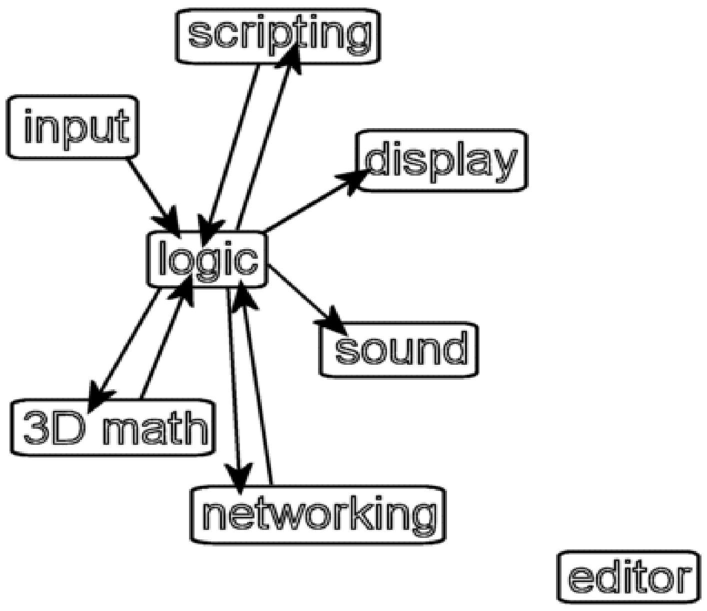
Contra is a popular side scrolling, adventure type game first released by Konami. It involves a character or a set of characters that a player can be and the mission is to destroy all the monsters / robots / bad guys, avoid traps, defeat the "bosses" and finally save the day.

Since the switch to the Linux in the Sun Lab there are fewer games available for CS students to play when it comes to time for a study break. CS Contra is built on a similar concept to the original contra and is particularly suitable for the Sun Lab due to it's ease of use, wide audience and multiplayer networked game scenarios.

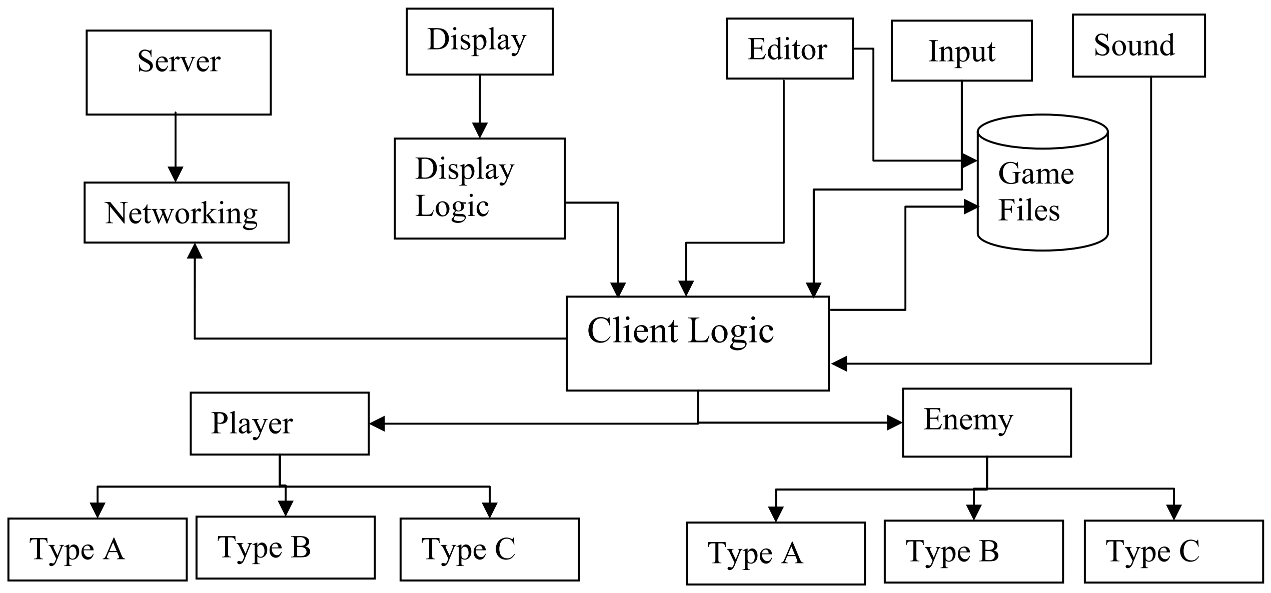
In the specifications documents for CS Contra, there is significant detail about implementation considerations, in this stage however, I will focus primarily on the different components of the program, how they will interact and how the project is going to be divided among the team members.

High Level Component Diagram:

Original System Block diagram as defined in the specifications:



Proposed component Diagram:



Description of components

Server

This component will be initialized if a player wishes to play a networked game. There can be an arbitrary number of servers running at any one time. Once a server is initialized, it will be searchable on the network and will interface between two (or more) players. It will contain methods to allow different client sessions to communicate. It will receive and send information to and from the clients so that they both know each other's statuses.

Networking

The networking component will handle the packaging and transmitting of information across the network; it will send and receive data packets to and from the server and client logic.

Display

This component will be the user interface and will display the game in session. It will be able to display the current environment, any enemies and the player. User input will be received by the Input component and sent to the Display Logic, which will tell the Display how to change. The Display will do all the collision detection and send information to the Display Logic in the event of collisions between any two objects. The Display Logic will then deduce which two objects collided and decide what to do. Certain collisions that do not change the game state will not be relayed to the client logic but will be processed exclusively by the Display Logic.

Display Logic

I have made Display Logic independent from the Display because the Display will be concerned with drawing and the Display Logic will receive instructions from the client logic to alter the display and it will also receive collision information from the Display component. Any mathematical computations that need to be made in order to determine future positions of objects will be done here. For example, if the Client Logic decides that an enemy should fire at the player, it will send this information to the Display Logic, which will determine where the enemy and player are and thereby instruct the Display to draw.

Client Logic

This component will be control the actual game play, flow of control of events, state of the game, player characters and enemy characters. User input will be received by the client logic from the Input component and will then decide what to do with it. If the input requires a change in the display, the client logic will relay the information to the Display Logic, which will then update the display. The Client Logic will track all the game objects that are active and all the interactions that they make. Any changes in game state or player status will be sent to the Client Logic from the Display via the Display Logic.

Editor

The editor will allow users to create new level files. The user will also have the option to use pre-existing level files that we create and save new files. They will not be able to overwrite the pre-existing level files. The Editor will contain a set of game objects that can be added to a level. It will also be able to save and open files from the game file repository.

Input

All user input will be taken in by the Input module and sent to the client logic to parse. This includes both keyboard and mouse input.

Sound

The sound module will be relatively small and make use of a pre-existing sound library. It will contain functions that are called directly by the client Logic for audio output before, during and after game play.

Game Files

All the level files will be stored in the Game Files component in the form of some type of graphical array such as a pixmap or in some textual form.

Enemy

There will be an enemy interface that will be implanted by an arbitrary number of different implementations. Each implementation will consist of an enemy object with certain characteristics, appearance and behaviors. Certain enemy objects may be static and other dynamic. The Client Logic will have access to each enemy's properties and behaviors. There will be a unique enemy type that will be a "boss" for each level.

Player

The user can choose what type of character he/she wants to be. The different players will all implement the same interface and have the same basic functionality. All player characters will be able to move left and right, jump, shoot and all the other requirements as described in the specifications. There will be differences in certain properties of each character, for example one may be able to move faster of inflict more damage.

<u>External Dependencies</u>

There are some external dependencies in the Design, but none of them can be considered risky.
Graphics

- Open GL will be used as the rendering engine
- Some artwork will be integrated from Andy Hull's independent Study Project.

Networking

- Qt or something similar will be used to send data between players

Sound

- One of many sound libraries can be used for this purpose

For all three of these dependencies, the libraries and application frameworks are freely available, robust and well documented.

Task Breakdown & Group Organization

Team Lead / Project Manager

This person will be most familiar with the high-level detail of the design and implementation of the entire project. The Team Lead will have the ultimate say on deciding the design specification, but implementation decisions will be left up to the specific coders. The Project Manager will also be responsible for generating the documentation for the game and ensuring that all the code is archived correctly and efficiently in CVS.
(1 Person)

Coding

- Server & Networking
  - Ensure that the satisfies all the functionality requirements as specified and that the Server can use the networking component to transmit information to a dummy client (1 person)
- Client Logic
  - Devise and implement algorithms and data structures to keep track of all game items and their interactions (2 People)
- Display
  - Be able to draw and update game detail using Frustum Culling.
  - Be able to send and receive information from Display Logic. (1/2 People)
- Display Logic
  - Receive data from Client logic and Display, process the data and send information either to the Client Logic, the Display or both. (1 Person)
- Editor
  - Allow the user to create level files by positioning enemy objects and landscape (1 person)
- Sound & Input
  - Use an external sound library to control audio output during game play as well as while using other portions of the game
  - Detect all forms of user input and send to the Client Logic (Tester)
- Players and Enemies
  - Generate classes for player characters and enemy objects (1 Person)

Testing

Devise a feasible and thorough test plan for individual components and for the game after integration. Assist the Program Manager with documentation while if the testing plan is complete and test drivers have been created. Work on the Sound & Input components of the system. I think that the tester will be able to finish the testing plan and test drivers comparatively quickly and can then proceed to code up the Sound and Input modules that are very independent and can be tested independently
(1 Person)

Schedule

3/3	Review everyone else's Top Level Designs and prepare any relevant comments
3/5	Assign Preliminary roles, discuss issues/concerns begin work on final Top Level Design
3/7	Hand in final Top Level Design for project
3/12	Propose first draft of interface prototypes.
3/17	Group meeting to review dependent interfaces.
3/21	Hand in complete final interfaces. Changes and updates have been reviewed and approved by the program manager documented as necessary.
4/4	Hand in detailed designs & begin coding
4/11	Group meeting to discuss test plan and test procedures. Any changes or alterations that have been made up to now may alter details in the test plan
4/16	Begin component integration and independent component testing by tester.
4/24	Finish all post integration testing integration and code optimization
4/27	Complete all documentation and fix as many bugs as possible without hurting functionality. Freeze functionality implementation
4/28	In-class Demos
4/30	Group meeting to discuss any final organizational and documentation based improvements
5/2	Public Demos
5/9	System Submission. Program must me installed and fully functional. Complete documentation must be available.

Assumptions

I did not need to make any assumptions about the project, I have detailed my functionality as required by the specifications, but have slightly altered the design in parts compared to what had been previously suggested.