

Table of Content

	Page
I. Abstract	3
II. Introduction	4
1. Wireless Technology and Architectural Evaluation	5
1.1 Introduction	5
1.2 What is a Wireless LAN?	5
1.3 Why use Wireless LAN?	5
1.4 Some Examples uses of Wireless LAN	6
1.5 Security in Wireless LAN	6
1.6 How Wireless LAN Works?	7
1.7 Wireless LAN Configurations	8
1.8 Hardware Evaluation of some wireless devices	11
2. Introduction to Direct X	15
2.1 What is Direct X?	15
2.2 Structure of Direct X	16
3. Introduction to WinSock	17
3.1 What you need to run WinSock application?	18
3.2 The Sockets Programming Paradigm under Windows	19
3.3 Overview of Connection-Oriented and Connectionless Application	20
3.4 Description of WinSock function	21
3.5 The Windows Programming Paradigm with WinSock	26
3.6 Blocking and event-Driven Application	27
3.7 Overlapped I/O, Scatter and Gather	27
4. Our Own Libraries	28
4.1 Our Direct Draw class:	28
4.2 Description of Direct Draw function:	29
4.3 Class "Sprite":	32
4.4 Class "Frame":	34
4.5 Class "Region":	35
4.6 An example of using a Spt file to represent an interactive object:	36
4.7 Our Direct Sound Class:	37
5. WinTalk (Program 1)	40
5.1 Introduction	40
5.2 The aim of this writing this program:	40
5.3 The pseudo-code of WinTalk:	41
5.4 Outcome	41

6. Reversi (Program 2)	42
6.1 Introduction	42
6.2 The aim of this writing this program:	42
6.3 The pseudo-code of Reversi:	43
6.4 Outcome	43
7. Plane (Program 3)	44
7.1 Introduction	44
7.2 The aim of this writing this program:	44
7.3 The pseudo-code of Plane:	45
7.4 Algorithm Analysis:	47
7.5 Outcome	49
8. Ball (Program 4)	50
8.1 Introduction	50
8.2 The aim of this writing this program:	50
8.3 The pseudo-code of Ball:	51
8.4 Outcome	51
9. ChatRoom (Program 5)	52
9.1 Introduction	52
9.2 The aim of this writing this program:	52
9.3 The new protocol and the generic server	53
9.4 The pseudo-code of ChatRoom-Client Program	54
9.5 The pseudo-code of ChatRoom-Server Program	54
9.6 Outcome	55
10. Fun with Learning English (FWLE)	56
10.1 Introduction	56
10.2 Interface of Fun with Learning English	57
10.3 Demonstration of a Scenario	58
10.4 The aim of this writing this program:	60
10.5 The pseudo-code of FWLE:	61
10.6 Algorithm	62
10.7 Outcome	65
11. Our Future Plan	66
References	67
Appendix A Statistics of Programs	68
Appendix B Progress Report	69

I. Abstract

Wireless technology provides ubiquitous connection in a concentrated area. When wireless network is established for a campus, teachers and students can enjoy closer interactions and better access to knowledge resources. Not only students will feel more open and comfortable when they ask questions through wireless connections with the teachers, but also teachers can interact with the students more privately and directly, thus creating more personal relationship between teachers and students and improving the quality of education. This close-up interactive environment is hard to be provided with clumsy desk-top computers in a classroom setting, where such a classroom would have to be carefully furnished and wired, thereby lowering the density of participants in the room.

Wireless campus also allows teachers and students to stay in touch with education material. They can conveniently access the library, check their e-mails, chat with others, search information on the Internet, and retrieve or review education material. Combining with thin-client devices, the wireless campus is an effective way for IT ownership in schools with much reduced cost, compared with wired campus. Furthermore, wireless campus encourages portability, connectivity, and communication efficiency. The deployed network is scalable and flexible.

II. Introduction

In our project, we need to implement the software infrastructure for wireless environment. Here is a brief introduction to our FYP.

Firstly, we evaluate the differences between different OS. Finally, We choose Windows as our working platform. It is because we can develop multi media more easily under Windows. Moreover, Windows is more common among general users.

Secondly, we start to build our own libraries for software development in future. We have created the libraries for socket programming, graphical and audio. In the meantime, we have written some testing programs for testing our libraries.

Finally, we try to integrate all of the libraries, engine to build an educational application for wireless network. At this stage, our final product is not finished yet. But with our libraries and engines, we can widely extend our product for different uses. For example, the learning style will change greatly. The classes can carry out outside the classroom. People work in open area can access the resource though the wireless LAN.

Although the wireless environment is still not command in the society, this technology will become very important in the future years. We hope that our project can improve the learning or working environment for the whole society.

1. Wireless Technology and Architectural Evaluation

1.1 Introduction

This objective calls for investigation on the different hardware in wireless and thin-client equipment, and the forecast of what would be available in the near future for the best performance and price consideration.

1.2 What is a Wireless LAN?

A wireless local area network (LAN) is a flexible data communications system implemented as an extension to or as an alternative for, a wired LAN. Using radio frequency (RF) technology, wireless LANs transmit and receive data over the air, minimizing the need for wired connections. Thus, wireless LANs combine data connectivity with user mobility.

1.3 Why use Wireless LAN?

There are several advantage for we to use wireless LAN:

1. **Mobility:** Wireless LAN systems can provide LAN users with access to real-time information anywhere in their organization. This mobility supports service opportunities not possible with wired networks.
2. **Installation Speed and Simplicity:** Installing a wireless LAN system can be fast and easy and can eliminate the need to pull cable through walls and ceilings.
3. **Installation Flexibility:** Wireless technology allows the network to go where wire cannot go.
4. **Reduced Cost-of-Ownership:** Overall installation expenses and life-cycle costs of Wireless LAN can be significantly lower than tradition wired LAN. Long-term cost benefits are greatest in dynamic environments requiring frequent moves and changes.
5. **Scalability:** Wireless LAN systems can be configured in a variety of topologies to meet the needs of specific applications and installations. Configurations are easily changed and range from peer-to-peer networks suitable for a small number of users to full infrastructure networks of thousands of users that enable roaming over a broad area.

1.4 Some Examples uses of Wireless LAN

- Doctors and nurses in hospitals are more productive because hand-held or notebook computers, PDA or HPC with wireless LAN capability deliver patient information instantly.
- Network managers in dynamic environments minimize the overhead caused by moves, extensions to networks, and other changes with wireless LANs.
- Training sites at corporations and students at universities use wireless connectivity to ease access to information, information exchanges, and learning.
- Network managers installing networked computers in older buildings find that wireless LANs are a cost-effective network infrastructure solution.
- Trade show and branch office workers minimize setup requirements by installing pre-configured wireless LANs.
- Warehouse workers use wireless LANs to exchange information with central databases, thereby increasing productivity.
- Network managers implement wireless LANs to provide backup for mission-critical applications running on wired networks.
- Senior executives in meetings make quicker decisions because they have real-time information at their fingertips.

1.5 Security in Wireless LAN

Many people thin that Wireless LAN should be less secure than traditional wired LAN. However, wireless technology has roots in military applications, security has long been a design criterion for wireless devices. Security provisions are typically built into wireless LANs, making them more secure than most wired LANs. It is extremely difficult for unintended receivers (eavesdroppers) to listen in on wireless LAN traffic. Complex encryption techniques make it impossible for all but the most sophisticated to gain unauthorized access to network traffic. In general, individual nodes must be security-enabled before they are allowed to participate in network traffic.

1.6 How Wireless LAN Works?

Wireless LANs use electromagnetic airwaves (radio or infrared) to communicate information from one point to another without relying on any physical connection. The data being transmitted is superimposed on the radio carrier so that it can be accurately extracted at the receiving end. This is generally referred to as modulation of the carrier by the information being transmitted. Once data is superimposed (modulated) onto the radio carrier, the radio signal occupies more than a single frequency, since the frequency or bit rate of the modulating information adds to the carrier.

Multiple radio carriers can exist in the same space at the same time without interfering with each other if the radio waves are transmitted on different radio frequencies. To extract data, a radio receiver tunes in one radio frequency while rejecting all other frequencies.

In a typical wireless LAN configuration, a transmitter/receiver (transceiver) device, called an access point, connects to the wired network from a fixed location using standard cabling. At a minimum, the access point receives, buffers, and transmits data between the wireless LAN and the wired network infrastructure. The access point (or the antenna attached to the access point) is usually mounted high but may be mounted essentially anywhere that is practical as long as the desired radio coverage is obtained.

End users access the wireless LAN through wireless-LAN adapters, which are implemented as PC cards in notebook or palmtop computers, as cards in desktop computers, or integrated within hand-held computers. Wireless LAN adapters provide an interface between the client network operating system (NOS) and the airwaves via an antenna. The nature of the wireless connection is transparent to the NOS.

1.7 Wireless LAN Configurations

1. A simple wireless peer-to-peer network

Wireless LANs can be very simple. At its most basic, two PCs equipped with wireless adapter cards can set up an independent network whenever they are within range of one another. This is called a peer-to-peer network. Here is one of an example.



Fig 1.1 A simple wireless peer-to-peer network

2. Client and Single Access Point

Installing an access point can extend the range of an ad hoc network, effectively doubling the range at which the devices can communicate. Since the access point is connected to the wired network each client would have access to server resources as well as to other clients. Each access point can accommodate many clients; the specific number depends on the number and nature of the transmissions involved. Here is an example.



Fig 1.2 Client and Single Access Point

3. *Multiple access points and roaming*

Access points have a finite range. In a very large facility such as a warehouse, or on a college campus it will probably be necessary to install more than one access point. The goal is to blanket the coverage area with overlapping coverage cells so those clients might range throughout the area without ever losing network contact. The ability of clients to move seamlessly among a cluster of access points is called roaming. Access points hand the client off from one to another in a way that is invisible to the client, ensuring unbroken connectivity.



Fig 1.3 Multiple access points and roaming

4. *Use of an extension point*

To solve particular problems of topology, the network designer might choose to use Extension Points to augment the network of access points. Extension Points look and function like access points.

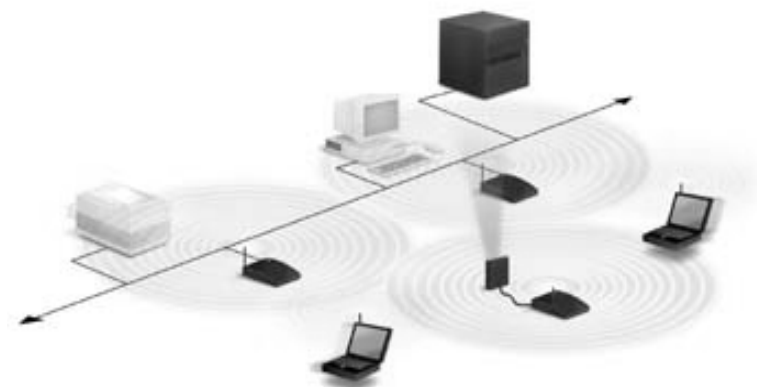


Fig 1.4 Use of an extension point

5. *The use of directional antennas*

One last item of wireless LAN equipment to consider is the directional antenna. Let's suppose you had a wireless LAN in your building A and wanted to extend it to a leased building, B, one mile away. One solution might be to install a directional antenna on each building, each antenna targeting the other. The antenna on A is connected to your wired network via an access point. The antenna on B is similarly connected to an access point in that building, which enables wireless LAN connectivity in that facility. Here is an example.



Fig 1.5 The use of directional antennas

1.8 Hardware Evaluation of some wireless devices

1. Proxim RangeLan2

Our department has this set of Wireless device in last year. Proxim uses license-free 2.4 GHz Frequency Hopping Spread Spectrum (FHSS) technology for robustness and scalability. FHSS is well known for its ability to resist interference and its ability to scale.

Specifications

Radio Data Rate	1.6 Mbps per channel, 800 Kbps fallback rate for extended range
Range	~400 feet (~122 m) in typical office environments ~700 feet (~213 m) in open spaces
Support Roaming	Yes
OS need	Win95/98, Windows NT, Windows CE 2.0, NetWare, DOS, Windows for Workgroups
Frequency Band	2.4 GHz band



Fig 1.6 RangeLAN2 7401/02 PC Card



Fig 1.7 RangeLAN2 Ethernet and Token Ring Access Points

2. Lucnet Technologies WaveLAN

Our department just buys this set of Wireless device in this year. WaveLAN also uses license-free 2.4 GHz Frequency for communication. One interesting feature is that the communication bandwidth is related to the range of coverage.

Specifications

Radio Data Rate	11 Mbit/s, 5.5 Mbit/s, 2 Mbit/s or 1 Mbit/s (depend on range of cover)
Range	Open Office: 525 to 1750 feet Semi Open Office: 165 to 375 feet Closed Office: 80 to 165 feet (depend on communication bandwidth also)
Support Roaming	Yes
OS need	Novel Client 3.x & 4.x, Windows 95/98/2000, Windows NT (NDIS Miniport driver), Apple, Windows/CE
Frequency Band	2.400-2.4835 GHz band



Fig 1.8 WaveLAN IEEE Turbo PC Card



Fig 1.9 WavePOINT-II Access Point

3. Apple iBook Airport

Apple iBook is a notebook run in Mac OS. It provides a Wireless device called AirPort, and it lets you use your iBook anywhere else you make yourself comfortable. Its range is up to 150 feet from an AirPort hardware access point.

Specifications

(As Apple haven't release detail specifications about this product, we can't get too much information about it)

Radio Data Rate	Up to 11 Mbit/s
Range	Open Office: 150 feet
Support Roaming	unknown
OS need	Mac OS 8 IEE 802.11 DSSS compliant
Frequency Band	unknown



Fig 2.0 Apple iBook and the Airport station

4. BlueTooth

Actually, BlueTooth is not a Wireless device. It is a standard for future Wireless communication. This the new Bluetooth technology was formed by representatives from Ericsson Mobile Communications, Nokia Mobile Phones, and the IBM, Intel, and Toshiba corporations.

Bluetooth answers the need for short-range wireless connectivity within three areas:

1. *Data and Voice access points*

Bluetooth facilitates real-time voice and data transmissions.

The technology makes it possible to connect any portable and stationary communication device as easily as switching on the lights.

You can, for instance, surf the Internet and send e-mails on your portable PC or notebook regardless of whether you are wirelessly connected through a mobile phone or through a wire-bound connection (PSTN, ISDN, LAN, xDSL).

2. *Cable replacement*

Bluetooth eliminates the need for numerous, often proprietary, cable attachments for connection of practically any kind of communication device.

Connections are instant and they are maintained even when devices are not within line of sight. The range of each radio is approximately 10 meters, but it can be extended to around 100 meters with an optional amplifier.

3. *Ad hoc networking*

A device equipped with a Bluetooth radio establishes instant connection to another Bluetooth radio as soon as it comes into range.

Since Bluetooth supports both point-to-point and point-to-multipoint connections, several sub-nets can be established and linked together ad hoc. The Bluetooth topology is best described as a multiple sub-net structure.

2. Introduction to Direct X

2.1 What is Direct X?

DirectX is a set of development libraries for high performance games under **Windows95**. **DirectX** consists of five major parts:

DirectDraw, DirectSound, DirectPlay, DirectInput, and Direct3D.

(Our project has used **DirectDraw** and **Direct Sound** only.)

DirectDraw

It is the most important. It allows **direct access to the bits on the video card**. It also has the ability to **store surfaces directly on the video card**, this makes for some amazingly fast blits.

DirectSound

It does **low latency mixing of sound**, as well as some **basic sound manipulations** such as volume, pan, and frequency.

DirectPlay

It allows **multiplayer games** to connect via modem, null modem, lans, or other networks. The interface is the same for all the different connect methods.

DirectInput

it is actually part of Windows95. It allows one to easily take advantage of all the latest **joysticks** with ease.

Direct3D

It is a part of DirectX 2. It consists of two major modes, **Retained Mode**, a high-level API in which the application retains the graphics data, and **Immediate Mode**, a low-level API in which the application explicitly streams the data out to an execute buffer.

2.2 Structure of Direct X

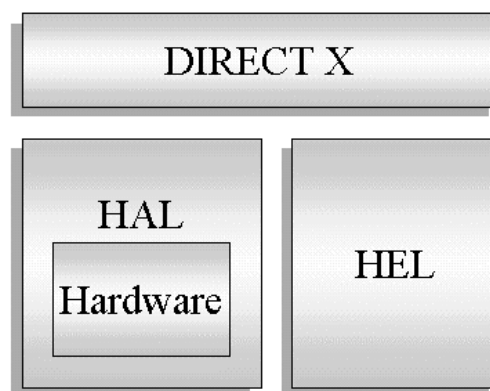


Fig 2.1. Structure of Direct X

DirectX is designed with hardware acceleration in mind. It tries to **provide the lowest possible level access to hardware**, while still **remaining a generic interface**.

HAL provide standard interface for accessing the same kind of hardware. When there is a request for hardware, HAL will search if the hardware exists, if it fails, HEL will be called.

When HEL is called, it will try to simulate the hardware by software. If it fails, the request will be ignored.

Advantages of this structure

1. It **provides the lowest possible level access to hardware**.
2. It **provides the abstraction and a standard access interface of hardware**.
Developers no need to concern about the implementation details of that hardware.
3. After installing the newer version of direct x and the computer will support the new hardware automatically.

The main reasons for using Direct X in our project

1. We want to write a **fancy and interactive application**, which need to deal with large amount of graphics, animations and sounds.
2. **The generic APIs** provided by visual C for sound and graphics **are too weak**.
For example, there will be twinkling when updating the screen using the generic API. Also, the generic API does not support mixing of sound buffer. So we cannot play more than one wav file at the same time.

3. Introduction to WinSock

WinSock is the network application-programming interface (API) for Microsoft Windows Operating System.

It is a translator of sorts. For programmers, it provides generic network services; WinSock translates those generic network services into protocol-specific requests and performs the necessary task. Thus, WinSock shields the programmers from the details of low-level network protocol.

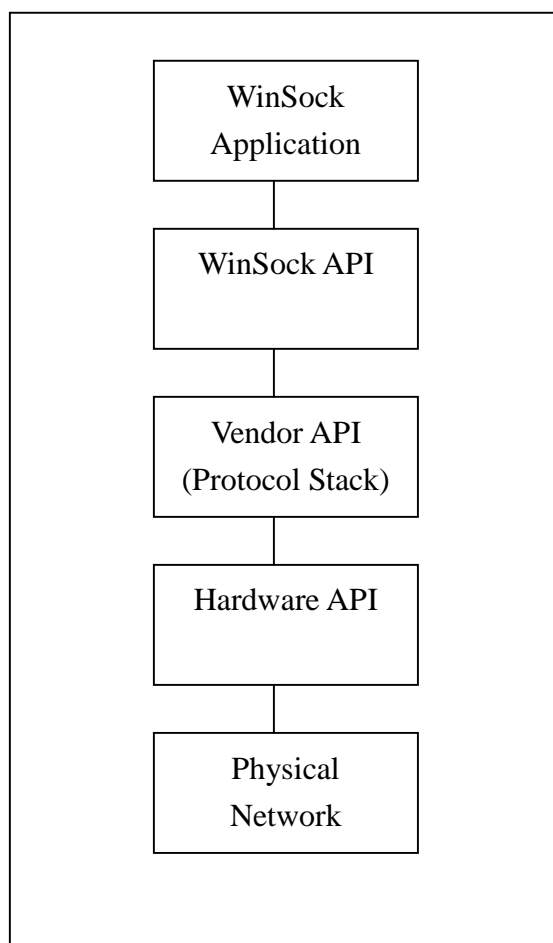


Fig 3.1 Overviews of how WinSock works

3.1 What you need to run WinSock application?

- Windows 95 or Windows NT 4.0 Operating System. (Windows 95 ship with WinSock 1.1, Windows NT 4.0 ship with WinSock 2.0)
- Several vendors make WinSock implementations available for previous versions of Windows (e.g. Windows 3.1 or Windows NT 3.51)
- Most of C and C++ compilers which can run under Windows Operating System.

Three distributions of WinSock currently exist. Fig x.x list the files associated with each distribution.

Dynamic Link Library (DLL)	Application	Development Files	Platform
WINSOCK.DLL	16-bit WinSock 1.1	WINSOCK.H WINSOCK.LIB	16-bit or 32-bit Windows
WSOCK32.DLL	32-bit WinSock 1.1	WINSOCK.H WSOCK32.LIB	32-bit Windows
WS2_32.DLL	32-bit WinSock 2.0	WINSOCK2.H WS2_32.LIB	32-bit Windows

Table 3.1 Files and platforms for the Three WinSock Distributions

3.2 The Sockets Programming Paradigm under Windows

Most of the WinSock development is follow the Berkeley sockets model. With some exceptions, WinSock includes the most Berkeley sockets API. Most WinSock function names and parameters is identical with the Berkeley sockets library. WinSock offers additional functions that are used to cope with 16-bit Windows system.

WinSock use a client/server approach to communicate. One application is theoretically always available (server side) and another request services as needed (client side).

The server “creates” a socket, name it so that it can be identified and found by a client, and then “listens” for services requests. A client application creates a socket, finds a server socket by name or address, and then “plugs in” to initiate a conversation. Once a conversation is start, data can be sent in either direction.

At application level, both server and client need to know what messages and data to expect from the other. They must use the same protocol.

Two fundamental types of client/server application pair exists in WinSock also: *connection-oriented* and *connectionless application*.

3.3 Overview of Connection-Oriented and Connectionless Application

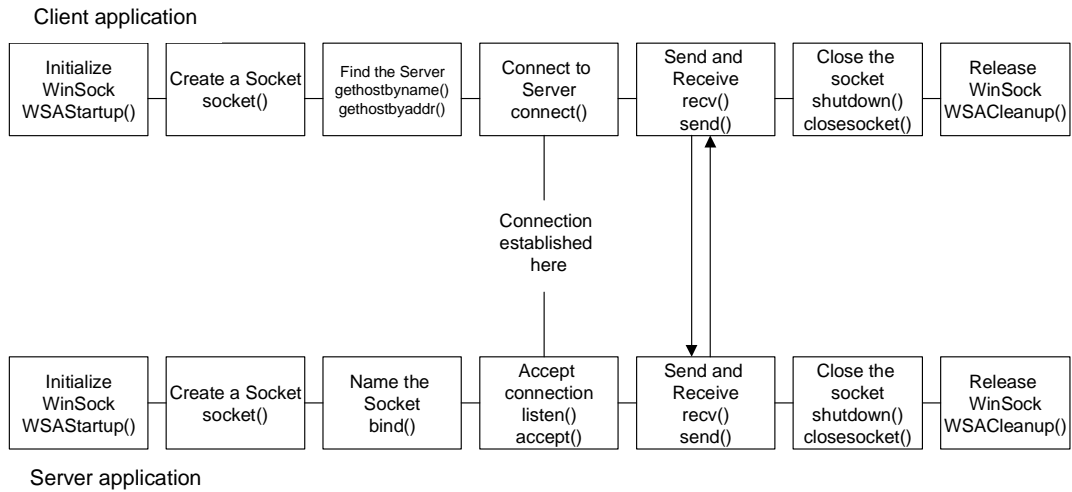


Fig 3.2 Overview of Connection-Oriented Application

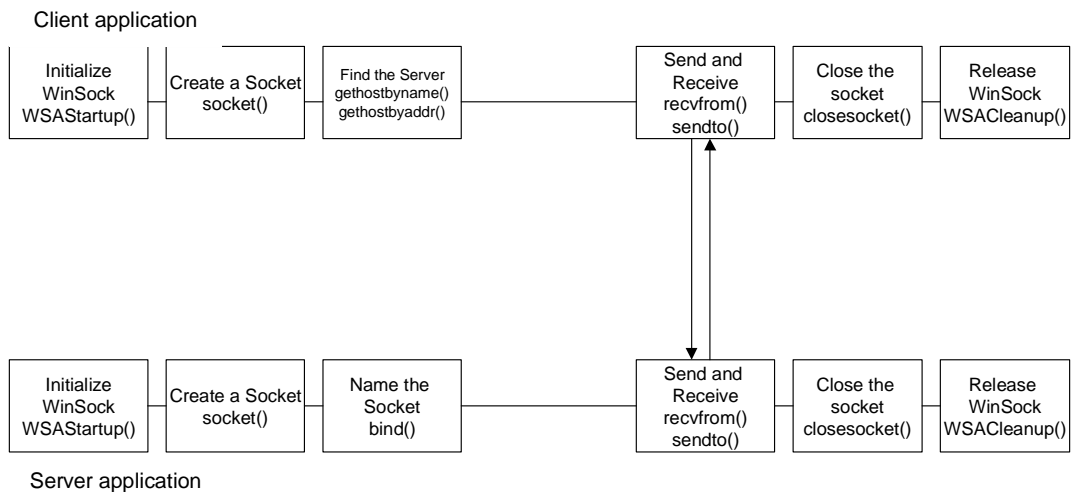


Fig 3.3 Overview of Connectionless Application

3.4 Description of WinSock function

WSAStartup() – Every WinSock application must make a successful call to this function before making other WinSock calls. This function gives the WinSock DLL to allocate resources, register the calling process, and possibly refuse service if no system resources are available. Also, it uses to check with the WinSock DLL to see if the version and features of WinSock is correct or not. Here is the function prototype of WSAStartup() in **WINSOCK.H**

```
int PASCAL FAR WSAStartup (   WORD wVersionRequired,
                             LPWSADATA lpWSADATA   );
```

socket() – This function is like the “socket()” function in Berkeley socket model. Before any connection start, we have to create a socket. This socket is the end-point between two end of the connection. When you successfully create a socket, it will return a socket descriptor. User can pass this socket descriptor to other WinSock function to designate the socket they want to use. Here is the function prototype of **socket()**

```
SOCKET PASCAL FAR socket (   int address_family,
                             int socket_type,
                             int protocol           );
```

bind() – This function is like the “bind()” function in Berkeley socket model. This function allows the **server** to name a socket fully. Here is the function prototype of **bind()**

```
int bind (   SOCKET socket_descriptor,
            const struct sockaddr FAR * name,
            int namelen           );
```

3.4 Description of WinSock function (Cont'd)

gethostbyname() – This function is like the “gethostbyname()” function in Berkeley socket model. It is used by the **client** application. It accepts a character string containing a host name. It returns a pointer to a **hostent** structure if it finds the host, or NULL if it does not. Here is the function prototype of **gethostbyname()**

```
struct hostent FAR * PASCAL FAR gethostbyname ( const char FAR * host_address );
```

gethostbyaddr() – This function is like the “gethostbyaddr()” function in Berkeley socket model. It is used by the **client** application. It accepts a host address in **IPv4** format. It returns a pointer to a **hostent** structure if it finds the host, or NULL if it does not. Here is the function prototype of **gethostbyaddr()**

```
struct hostent FAR * PASCAL FAR gethostbyaddr ( const char FAR * host_address,
                                                int len,
                                                int address_type );
```

connect() – This function is like the “connect()” function in Berkeley socket model. It is used by **connection-oriented client** application. This function is used to connect a client socket to a server socket. Here is the function prototype of **connect()**

```
int PASCAL FAR connect ( SOCKET socket_descriptor,
                        const struct sockaddr FAR * name,
                        int namelen );
```

listen() – This function is like the “listen()” function in Berkeley socket model. It is used by **connection-oriented server** application. After binding a socket to its address, the server then call this function to wait for client requests. Here is the function prototype of **listen()**

```
int PASCAL FAR listen ( SOCKET socket_descriptor,
                       int backlog );
```

3.4 Description of WinSock function (Cont'd)

accept() – This function is like the “accept()” function in Berkeley socket model. It is used by **connection-oriented server** application. After setting the socket to “listening”, the server application should call this function to accept client requests. Here is the function prototype of **accept()**

```
SOCKET PASCAL FAR accept (    SOCKET socket_descriptor,
                             struct sockaddr FAR * addr,
                             int FAR * addrrlen                );
```

send() – This function is like the “send()” function in Berkeley socket model. After the connection is established, the **connection-oriented** application can use this function to send data to the other side. The destination of the data is understood to be the address and port specified in the call to connect(). Here is the function prototype of **send()**

```
int PASCAL FAR send (    SOCKET socket_descriptor,
                        const char FAR * data_buffer,
                        int len,
                        int flags                );
```

recv() – This function is like the “recv()” function in Berkeley socket model. After the connection is established, the **connection-oriented** application can use this function to receive data from the other side. The source of the data is understood to be the address and port specified in the call to connect(). Here is the function prototype of **recv()**

```
int PASCAL FAR recv (    SOCKET socket_descriptor,
                        const char FAR * data_buffer,
                        int len,
                        int flags                );
```

3.4 Description of WinSock function (Cont'd)

sendto() – This function is like the “sendto()” function in Berkeley socket model. After the connection is established, the **connectionless** application can use this function to send data to the other side. The destination of the data is the socket specified in the “to” field. Here is the function prototype of **sendto()**

```
int PASCAL FAR sendto ( SOCKET socket_descriptor,
                       const char FAR * data_buffer,
                       int len,
                       int flags,
                       const struct sockaddr FAR * to,
                       int to_len
                       );
```

recvfrom() – This function is like the “recvfrom()” function in Berkeley socket model. After the connection is established, the **connectionless** application can use this function to send data to the other side. The source of the data is the socket specified in the “from” field. Here is the function prototype of **recvfrom()**

```
int PASCAL FAR recvfrom ( SOCKET socket_descriptor,
                          const char FAR * data_buffer,
                          int len,
                          int flags,
                          struct sockaddr FAR * from,
                          int from_len
                          );
```

shutdown() – This function is used for **connection-oriented** application. It is used to notify the peer application before closing the socket. Here is the function prototype of **shutdown()**

```
int PASCAL FAR shutdown( SOCKET socket_descriptor,
                          int by_how
                          );
```


3.4 Description of WinSock function (Cont'd)

closesocket() – This function is like the “close()” function in Berkeley socket model. It is used to close a communication. For **connectionless** connection, it can be call directly. For **connection-oriented** connection, a shutdown() function should be call before this function. Here is the function prototype of **closesocket()**

```
int PASCAL FAR closesocket ( SOCKET socket_descriptor );
```

WSACleanup() – This is used to release all resource related to WinSock back to the Operating System. It should be call at the end of every WinSock application. Here is the function prototype of **WSACleanup()**

```
void WSACleanup ( void );
```

3.5 The Windows Programming Paradigm with WinSock

Feature of WinSock

Today, the newest version of WinSock is 2.2. Here is some feature of the newest WinSock:

1. Multi-protocol support

The newest allow an application to use the familiar socket interface to achieve simultaneous access to any number of installed transport protocols. Not like the previous version, WinSock is no longer to use on TCP/IP only.

2. Asynchronous I/O and event objects

With Win32 programming environments, WinSock can extend to communicate asynchronously. Asynchronous I/O enables an application to continue with other processing while waiting for the I/O operation to complete.

3. Quality of Service

The newest WinSock established conventions for applications to negotiate required service levels of communication service such as bandwidth and latency for some quality demanding application such as multimedia communication.

As we can see, WinSock still has some advantages over traditional Berkeley socket model.

3.6 Blocking and event-Driven Application

Blocking means that when a function is called, it stops all other processing in the application and does not return until that function is completed. This is all right in console application. However, in Windows GUI environment, it is a great problem. Windows GUI are based on **event driven**. They receive and must quickly respond to all the events (e.g. keystroke, mouse movement ... etc.).

Luckily, the WinSock specification adds many extensions to the original Berkeley socket API. One of the most important extensions for Win32 programming is the support of **asynchronous mode**. **Asynchronous notification** is the best way to doing Win32 GUI applications. Unlike traditional blocking operations, asynchronous operations return immediately, no whether it fail or success. After the operation is finished, the WinSock DLL will send a message to that calling application, indicate the function is finished.

This helps us a lot in designing our program.

3.7 Overlapped I/O, Scatter and Gather

Not just asynchronous notification, the newest WinSock also provide one extension that helps the programmers a lot in designing application: **Overlapped I/O**.

Asynchronous input and output (overlapped I/O) function return immediately, even when an I/O request is still pending. With overlapped I/O, an application can continue with other process while waiting for a send or receive operation to complete. These overlapped I/O functions are:

1. WSARecv()
2. WSASendTo()
3. WSARecv()
4. WSARecvFrom()

Also, the newest WinSock extends its Overlapped I/O with the concept of **scatter** and **gather**. Scattered/Gathered I/O is much like vectored I/O in UNIX. All four functions mention above can take an array of buffers as input parameters and can be used for scatter/gather I/O. This technique is very useful when data being transmitted is structured into two or more logical pieces.

4. Our Own Libraries

In order to write a fancy and interactive application for the wireless network, we have built our own graphical libraries and audio libraries. The following is the introduction of them.

4.1 Our Direct Draw class:

Since the APIs provided by Direct Draw are so confused and complicated, we have constructed our **Class of Direct Draw** to **encapsulate the details of the function calls**. With this Class, we can create and access the objects of Direct Draw easily.

The prototype of our Direct Draw class: (written in pseudo-code)

```

Class DD {

Public:

    DDStartup();

    CreateDesktopWindow( Window_handler, Direct_X_object );

    DDFullConfigure( Direct_X_Object );

    DDWinConfigure( Direct_X_Object );

    DDLoadPalette( Bmp_file, Palette_object );

    DDLoadBitmap( Bmp_file, Surface_object);

    DDSetColorKey( Surface_object, Key_color );

    DDMakeOffscreenSurface();

    DDCreateFlipper( Flipper_Object);

    DDCreateFakeFlipper( Flipper_Object );

    DDFlipping();

    DDFillSurface( coordinates, color );

    DDTextOut ( Surface_object );

    CleanUp();

Private:

    Direct_X_Object DX;

    Surface primary_surface, secondary_surface;

    Window_handler ghwnd;

    Character Mode;

};

```


4.2 Description of Direct Draw function: (Cont'd)

DDSetColorKey(Surface object, Key color):

This function is used to set a key color (transparency color) for a surface.

Here is an example:

Background surface:



Foreground surface:



Color Key:



Put the foreground surface
without color key:



Put the foreground surface
with color key:



DDMakeOffscreenSurface():

This function is used to make offscreen surface (secondary surface).

Typically, we will construct at least 2 surfaces when using DirectDraw. The primary surface and the secondary surface. Primary surface can be considered as the memory content in the video card and is always be shown on the screen. In the flipping process, the content of secondary surface will be copy to the primary surface.

If we don't use flipping, the cleanup process must be done directly to the primary surface. Because the background surface will cover the foreground surface when updating, the showing time of background picture will be longer than that of foreground picture. Therefore, there will be twinkling effect in the human eyes. If we use flipping, the cleanup and updating process will be done behind the primary screen. We will flip the secondary surface only when all of the foreground surfaces are ready. So it can eliminate the twinkling problem.

4.2 Description of Direct Draw function: (Cont'd)

DDCreateFlipper(Flipper Object):

This function is used to create a flipper instance

DDCreateFakeFlipper(Flipper Object):

This function is used to create a fake flipper.

In window mode, flipper object is not supported, so we need to use this function to create a fake flipper. It simulates the flipping process by using the memory copy function. Its performance is as good as a real flipper.

DDFlipping():

This function is used to do the Flipping. This function will know whether we are using the fake flipper or the real flipper.

DDFillSurface(coordinates, color):

This function is used to fill a rectangle on the surface. We can use this function to cleanup the screen if we do not have a background picture.

DDTextOut (Surface_object):

This function is used to print text on a surface.

CleanUp():

This function is used to cleanup all Direct Draw Object.

4.3 Class “Sprite”:

Class *Sprite* is built to manipulate the *.*spt* files. It has included the **data structure for a spt file**, the **method for opening spt file** and the **method for handling spt’s data**.

Introduction to spt files:

In order to deal with the *picture* files (*.*bmp*) easily and efficiently. We have defined a file type called *spt* file. *Spt* file can be used to represent a man, a dog, a robot, or other interactive object in our application. Each *sprite* file (*.*spt*) has its corresponding *picture* files (*.*bmp*). The file name of the *picture* files (*.*bmp*) and the information of the *sprite* will be stored in its *sprite* file (*.*spt*).

Here is an example:

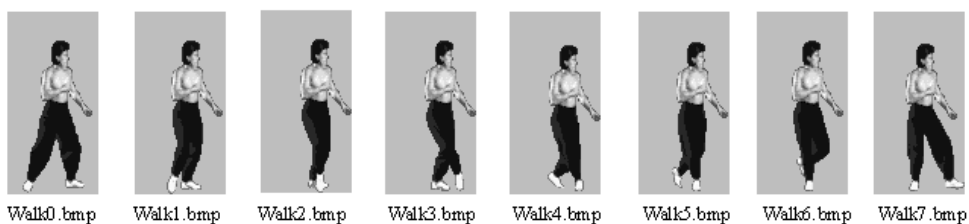


Fig.4.1 8 action pictures for a walking man

We have got the pictures of a walking man and want to play as an animation. So we can define a spt file for that walking man.

Here is the content of the spt file:

```
Sprite_name: "Walking Man"
No_of_frame: 8
Size_of_Picture: 150 x 100
Picture_filetype: bmp
Picture_filename: walk0, walk1, walk2, walk3, walk4, walk5, walk6, walk7
Transparency_color: 27
Frame_rate: 25
Reserved_space_for_future_use: .....
```

By using this kind of spt file (Our programs have a different format of spt file, this example is used for demo only), the programmer can write a generic animation player to play some simple animation. Actually spt file can be used to represent some interactive object rather than animation. There will be an example later in this chapter.

4.3 Class “Sprite”: (Cont’d)

The prototype of Class *Sprite*: (written in pseudo-code)

```

class Sprite
{

public:

    LoadSprite ( String sprite_file );
    SaveSprite ( String sprite_file );

    Setbmpfilename( string );
    SetId ( int );
    InsertFrame ( int, Frame );
    DeleteFrame ( int );
    SetReserved_string ( int, string );
    SetReserved_integer ( int, int );

    String getbmpfilename();
    Integer getId ();
    String GetReserved_string ( int );
    Integer GetReserved_integer ( int );

protected:

    Frame frm[];
    /* Frame will be introduced in the next session */
    /* Frame is not private because its member Retion need to be read by some friend
    functions for region calculations */

private:

    String Bmp_filename;
    Integer id;
    Integer no_of_frame;
    String Reserved_string[20];
    Integer Reserved_integer[100];

};

```

4.4 Class “Frame”:

Class Frame is built to store the information for each action. (*Frame* instance is defined as the member under *Sprite*.)

A *Sprite* may include several actions. In page ?, the example of *Sprite*, the *spt* file is representing a walking man with 8 frames.

Frame splitting:

Because each frame has its action picture, if there are many frames, it is so troublesome to deal with large number of picture files. Therefore, we have written a function for frame splitting. It let us to draw all the frame pictures in one picture file. And the method for opening a *spt* file can recognize all the frames' position automatically.

Algorithm:

(Assumption: each frame is drawn within a square)

```

For y = 0 to height of picture
  For x = 0 to width of picture
    If Pixel(x, y) != background_color and Pixel(x, y) is not used
      begin
        /*Pixel (x, y ) is the upper left corner of this frame*/
        Find the upper right corner of this frame
        Find the bottom left corner of this frame
        Save the coordinates of this frame in a linked list
      end
    Next x
  Next y

```

The prototype of our *Class Frame*: (written in pseudo-code)

```

class Frame
{

public:
  Frame get_frame ();
  Set_frame ( Frame );

private:
  String name_of_frame;
  Integer no_of_region;
  Region rg[];
  /* Region will be introduced in the next session */
  int go; /* for animating purpose */
  int wait; /* for animating purpose */
  String Reserved_string[20];
  Integer Reserved_integer[100];
};

```

4.5 Class “Region”:

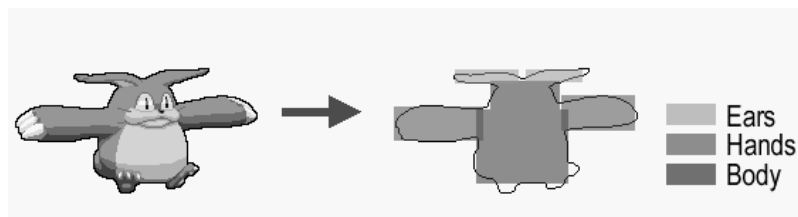


Fig 4.2. Define regions for different parts of body

Region is an attribute of *Frame*. It stores the coordinates of a rectangle in the action picture. (*Region* instance is defined as the member under *Frame*.)

The 2 main purposes of region definition:

1. Because most of the pictures have an irregular shape, it is expensive to detect the collision between them. On the other hand, if we define some rectangular regions to represent the picture body, the **collision detection will become simple**.
2. Moreover, **regions can represent** buttons, different parts of a body or other **interactive objects** on the picture.

The prototype of our *Class Region*: (written in pseudo-code)

```
class Region
{
protected:
    Integer x, y, w, h;
    /* They are not private because they need to be read by some friend functions
    for region calculations */
};
```

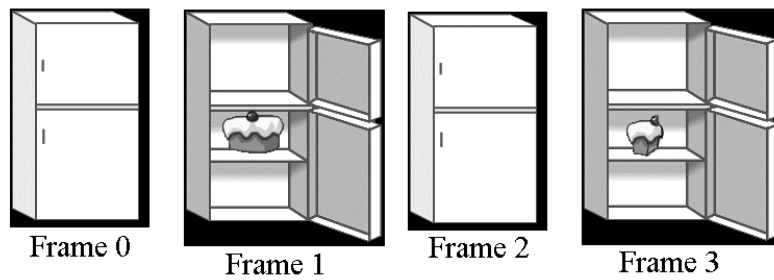
Attachment of other Frame Attributes

Actually, frames' and regions' positions are just some integer data attached in the sprite file. We have also reserved some string and integer in a *Spt* file for future use. These reserved data can have many meanings in future. For example, string can represent a message, or a *.wav file name for the sound effect. An integer can represent the frame rate, or other animating information.

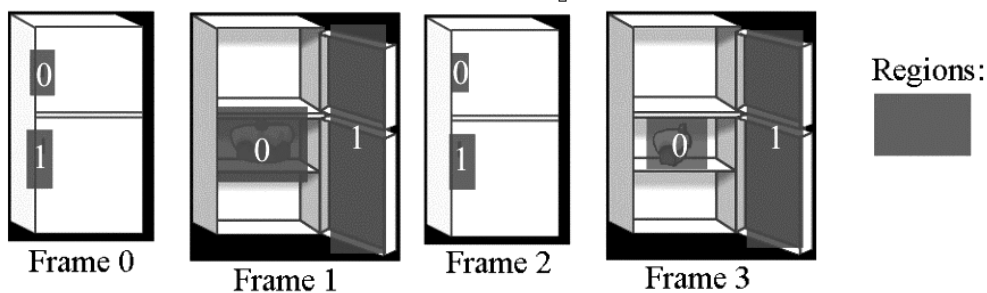
4.6 An example of using a Spt file to represent an interactive object:

We want to create a refrigerator using a spt file. The user can open and close the refrigerator by clicking on the side of the doors. There is a cake inside the refrigerator. The user can eat the cake when click on it. But if the cake has been eat once, it cannot be eat again.

The pictures for the refrigerator:



The regions defined for the frames:



We construct the spt file using the technique of constructing a finite state machine. Here is the DFA for the refrigerator:

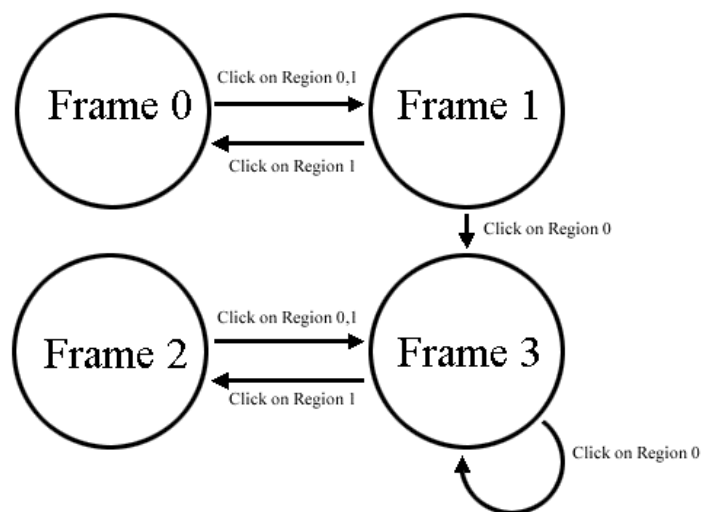


Fig. 4.3 DFA for the refrigerator

Here is the content of the spt file:

```
Sprite_name: "Refrigerator"
No_of_frame: 4
Transparency_color: 0

Frame0:
  No_of_Regions: 2
  Coordinates_of_regions: 20 30 20 30 20 80 20 40
  When_clicking_region0_goto_frame: 1   message: "Open the refrigerator"
  When_clicking_region1_goto_frame: 1   message: "Open the refrigerator"

Frame1:
  No_of_Regions: 2
  Coordinates_of_regions: 24 60 80 70 110 30 40 160
  When_clicking_region0_goto_frame: 3   message: "Eat the cake"
  When_clicking_region1_goto_frame: 0   message: "Close the refrigerator"

Frame2:
  No_of_Regions: 2
  Coordinates_of_regions: 20 30 20 30 20 80 20 40
  When_clicking_region0_goto_frame: 3   message: "Open the refrigerator"
  When_clicking_region1_goto_frame: 3   message: "Open the refrigerator"

Frame3:
  No_of_Regions: 2
  Coordinates_of_regions: 24 60 80 70 110 30 40 160
  When_clicking_region0_goto_frame: 3   message: "I have just eat!"
  When_clicking_region1_goto_frame: 2   message: "Close the refrigerator"
```

4.7 Our Direct Sound Class:

We have constructed our **Class of Direct Sound** to encapsulate the details of the **function calls**. With this Class, we can create and access the objects of Direct Sound easily.

The features of our audio library:

1. **Support** Wav file with **different sample rate**.
2. Different **sounds can be overlap** at the same time.

The prototype of our *Direct Sound Class*: (written in pseudo-code)

```
class SoundBuffer
{
public:
    SetupBufferFromWave( String );
                        /* load a wave file into sound buffer */
    Play ( integer );
                        /* play the wave, there is 2 mode of sound playing
                        the normal mode and loop mode.
                        Loop mode are usually use for background music */
    Stop ();
                        /* stop the playing of the wave */
    Static InitializeDirectSound();
                        /* Initialize Direct Sound, This function is static because
                        it function is common to all instance */
    Static ShutdownDirectSound();
                        /* Shut down Direct Sound, This function is static because
                        it function is common to all instance */

private:
    LPDIRECTSOUNDBUFFER soundbuffer;
                        /* the actual sound buffer */
    Static DirectSound_Object DS;
                        /* Direct Sound Object */
};
```

Mixer problem in Direct Sound:

Direct Sound supports sound mixer which let different sounds to overlap at the same time. Without mixer, when we want to play a sound buffer, we need to wait until there is no sound playing.

We find that there is a restriction on the Mixer provided by Direct Sound. The same sound buffer cannot be overlapped itself. For example, when we click on a man, there will be a “hello” sound in our program. But if we click on the man many times so frequently, the man will not respond to say “hello” every time. It is because there is a “hello” sound still playing.

Mixer problem in Direct Sound: (Cont'd)

We have considered 3 possible solutions for the Mixer problem:

1. Create duplicate buffers for the same sound file

Advantage:

- a. It is easy to implement.

Disadvantages:

- a. It wastes a lot of memory space.
- b. Don't know how many buffers is enough. Longer sound may need more buffers.

2. Create a constant number of buffers, load the sound file into the buffer only when it needs to play.

Advantage:

- a. It is difficult to implement.

Disadvantages:

- a. It saves the memory space.
- b. It leads to substantial delay of the sound, which is not acceptable.

3. When play a sound buffer, check whether it is being played. If yes, stop the sound and play again from the beginning.

Advantage:

- a. It is easy to implement.

Disadvantages:

- a. If the sound have a lower volume at beginning and higher volume at the end, user may realize that there is a cropping of sound.
- b. If the sound is a human conversation, user may realize that there is a cropping of sound.

Conclusion:

We have use method 3 for the solving the problem.

It is because usually, only sound effect will be overlapped itself. And Sound effect is always very short and has a higher volume at the beginning.

5. WinTalk (Program 1)

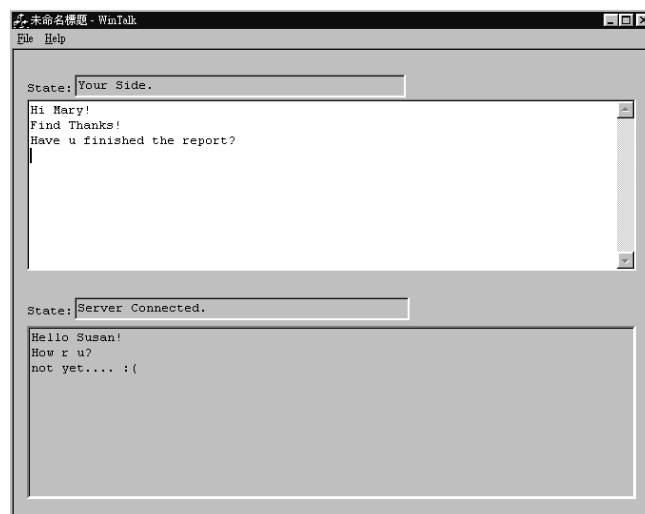


Fig 5.1 Snapshot

5.1 Introduction

WinTalk is a two-way communication program under Window OS. It works like the “talk” program under UNIX. Its function is very simple. It allows two users at two different PC can talk to each other. One assumption is that one of the users needs to know the other's PC host address. It is built based on Microsoft Foundation Class (MFC) using Visual C++ compiler.

5.2 The aim of this writing this program:

1. Try to use **Visual C ++** and **Microsoft Foundation Class (MFC)**.
This is the first time we try to use Visual C++ to compile a program written in MFC.
2. Try to use **Winsock**.
The structure of WinTalk is simple and it has so few of codes. So we can focus on how to use WinSock on real programming.
3. Try to use **Asynchronous mode of communication with event-driven programming paradigm**
This program is so simple that this can allow use to experience the asynchronous mode of communication with event-driven programming paradigm more clearly.

5.3 The pseudo-code of WinTalk:

```
Main()
{
    Initialize WinSock
    Ask user the other user IP address
    Connect to the other side
    Wait until Connected

    Loop
    {
        dispatch the event;

        for event "someone hit a key on keyboard"
            get the key stroke from keyboard
            Send the key stroke to the other user

        for event "someone send a key by network"
            receive the key stroke from network
            display the key stroke on user screen

    } until (someone quit)

    Shut Down WinSock
}
```

5.4 Outcome

1. We have to use **Visual C++ to compile program written in MFC.**
2. We have successfully **implement communication program using WinSock.**
3. We can understand more about **asynchronous mode of communication with event-driven programming paradigm.**

6. Reversi (Program 2)

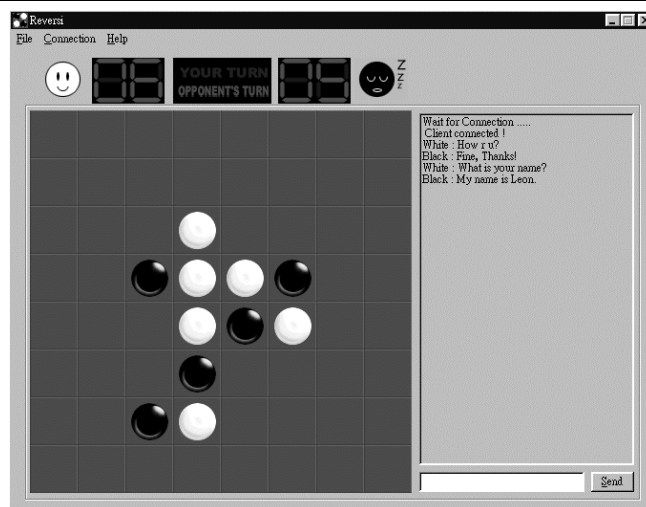


Fig 6.1 Snapshot

6.1 Introduction

Reversi is a simple chess program under Window OS. It works like the “Othello” chess we play. Its rules are very simple. A player picks black and the other picks white. Black always plays first. When it is your turn to play, you can place a disc of your color onto one of the empty squares on the board, provided that your move **MUST** be able to flip at least one of your opponent's discs. At the same time, you must flip **ALL** your opponent's discs between the disc you just put and the remaining one on the board. And that flipping goes in eight directions, horizontally, vertically and diagonally. It builds base on Microsoft Foundation Class (MFC) using Visual C++ compiler.

6.2 The aim of this writing this program:

1. Try to get more experience with **Visual C ++** and **MFC**
2. Try to **use MFC to display bitmap (bmp) files**
MFC has internally support how to display a bitmap file. We need to try it out because most of application we develop need to display bmp files

6.3 The pseudo-code of *Reversi*:

```
Main()
{
    Initialize WinSock
    Ask user the other user IP address
    Connect to the other side
    Wait until Connected

    Loop
    {
        dispatch the event;

        for event "someone place a chess of keyboard"
            check with the rule that the move is legal or not;
            if( not legal move )
                ignore the move;
            else
                get position of the move
                Send the position to the other user

        for event "someone send a position of a chess by network"
            receive the position of the chess from network;
            update the user screen using MFC;

    } until (someone quit or someone win)

    Shut Down WinSock
}
```

6.4 Outcome

1. We get more experience **in Visual C++ to compile program written in MFC.**
2. We have successfully **display bitmap file using MFC.**
3. However, one problem we found is that the performance of using MFC to display bitmap is extremely poor. There is a substantial delay with displaying bmp. Moreover, the user will see a twinkling effect when our program updating the screen. So, We need to try another approach to display bitmap file using Visual C++.

7. Plane (Program 3)



Fig 7.1 Snapshot

7.1 Introduction

Plane is a flight shooting game which supports 2 players mode.

Player can use the keyboard to do the following operations on his plane.

1. Turn Left
2. Turn Right
3. Move Forward
4. Move Backward
5. Turn on the Shield
6. Fire

Rules of the game:

1. When a plane is being shoot, its life will decrease.
2. The red bars at the top of the screen are showing the life power.
3. When the life power becomes empty, the plane will explode and dead.
4. The player will win if all his enemies dead.
5. Some Letter (food) will be appears on the screen randomly. A plane can be equipped by “eating” those Letters (food).

7.2 The aim of this writing this program:

1. Try to use our *Graphical and Audio Libraries*.
2. Try to use *Winsock, Direct Draw and Direct Sound* together in the same program
3. Try to implement the **synchronous connection (1 to 1)**.

7.3 The pseudo-code of *Plane*:

```

Main()
{
    Initialize WinSock
    Initialize Direct Draw
    Initialize Direct Sound
    Ask Player the opponent's IP address
    Connect to the other side
    Wait until Connected

    If I am player one
        {   Generate the random seed using timer
            Send the random seed to opponent     }

    Loop
        {
            Get the key stroke from keyboard
            Send the key stroke to the opponent
            get the key stroke from the opponent

            for i = 1 to no_of_players

                if player i is pressing Up
                    {   accelerate the plane i   }

                if player i is pressing Down
                    {   decelerate the plane i   }

                if player i is pressing Left
                    {   rotate plane i anti-clockwise }

                if player i is pressing Right
                    {   rotate plane i clockwise }

                if player i is pressing Fire
                    {   create a Fire object     }

                if player i is pressing Shield and shield_no > 0
                    {   create a Shield, Shield_no-- }

            next i

            Generate the Food on the screen randomly
            Calculate the new position and new speed of Planes using Physics Laws
            Calculate the new position and new speed of Fires using Physics Laws
            Calculate the new position and new speed of Foods using Physics Laws

            for i, j = (1,1) to (no_of_players, no_of_players)

                if plane i collide with plane j and i!=j
                    {   caculate the new position and speed of plane i, j
                        decrease the life power of plane i,j
                        play the collision sound effect     }

            next i, j
        }
}

```

7.3 The pseudo-code of *Plane*: (Cont'd)

```

for i, j = (1,1) to (no_of_players, no_of_fire)

    if plane i collide with fire j
        {
            caculate the new position and speed of plane i
            decrease the life power of plane I
            destruct fire j
            play the crash sound effect    }

next i, j

for i, j = (1,1) to (no_of_players, no_of_food)

    if plane i collide with food j
        {
            change plane i's attribute /*equip the plane */
            destruct food j
            play the eating sound effect    }

next i, j

Clean up the off screen buffer with space background
Draw the planes on the off screen buffer
Draw the fires on the off screen buffer
Draw the fires on the off screen buffer
Flipping /* Update the screen by the offscreen buffer */

} until (someone quit)

Shut Down Direct Sound
Shut Down Direct Draw
Shut Down WinSock
}

```

7.4 Algorithm Analysis:

1. Compress the data before sending

Our program has tried to compress the information as small as possible before sending to the other side. It can reduce the time for data transfer which is the bottleneck for frame rate. We use One byte (keystroke character) to represent the key stroke of a player.

Here is an example of a keystroke character:

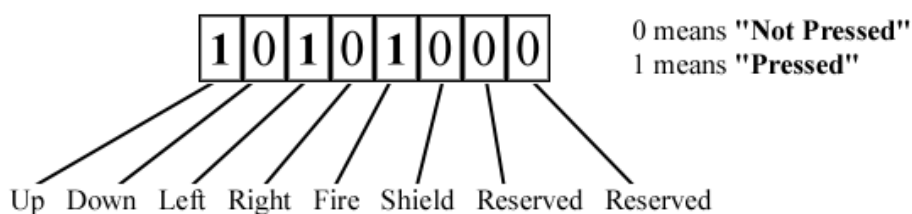


Fig7.2 An example of keystroke character

In this example, the player is pressing the “Up” button, “Right” button and “Fire” button.

2. Sharing of random seed

When 2 computers have the same value of random seed, they will generate the same sequence of random numbers. So we can just send the keystroke of the player to the other side.

After receiving the keystroke, our program can treat the other side player as a local player. It simplifies the structure of our program.

If 2 computers do not have the same random seed, our program will need to send the coordinates, states of all objects to the other side of computer. All the calculations of interaction also need to do centrally using one computer. It will make the structure of our program complex and increase the burden of data transfer.

7.4 Algorithm Analysis: (Cont'd)

3. Asynchronous connection

In Plane, we have implement the 1 to 1 synchronous connection.

The potential problems of using synchronous connection:

1. We realized that the quality requirement for synchronous connection is very high. If the bandwidth is not high enough. The frame rate of both side will become very low. It just like watching a movie in slow motion. In future, we need to implement the multi-client connection and transfer of video data. It is not possible for us to use synchronous mode to connect all the clients.
2. Also, if some client is at the other side of the earth, even the connection speed is as fast as light, there will be still several 10 ms of delay in data transfer.
3. Moreover, in a synchronous mode. All the client will suffer if the connection of one client has problem.

Therefore, we must use asynchronous mode connection in our future program.

We have considered some possible features for asynchronous connection:

a. Estimation the movement of the other side before receive the data.

For example, we are writing a program for virtual chat room. In the program, there is a 3-D room and we can walk through this room. We can see other user's message and movement.

If we update other user's position only when receiving their data. Their movement will be not smooth and strange. If our program can interpolate their intermediate position using their previous speed and position, the problem can be solved.

b. Store the message in a message queue if the other side is not ready

For example, we are writing a write board program. The program supports multi-clients. Clients can draw something on the write board and all clients share the same write board.

When the client is drawing, the coordinates of the pixel will be sent to the server. But if the server is busy at that moment. The client program will need to store the coordinates of the pixel in a queue until the server is ready to receive the data.

The potential problem of using message queue:

Obviously, this program may have a concurrency problem if client can draw different colors. And the solution for concurrency problem may need to deadlock. Anyway, we cannot discuss it deeply because we have not implemented message queue yet.

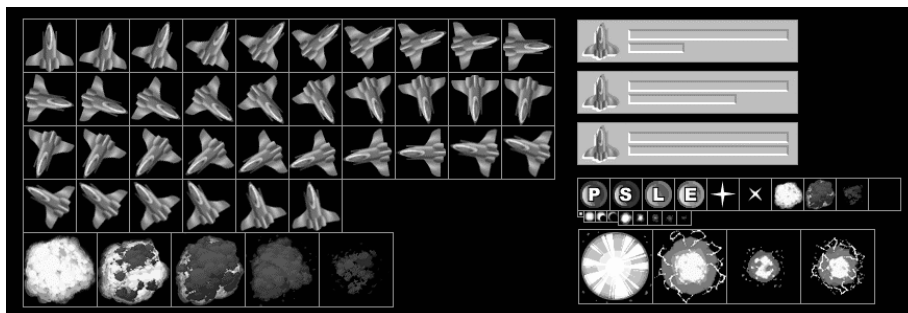


Fig 7.3 The frames of a plane

7.5 Outcome

1. We have discovered some problems in the structure of our *Graphical* and *Audio* libraries and have improved them.
2. We have successfully used *Graphical* and *Audio* libraries in the same program.
3. We have experienced the performance of *Direct Draw* and *Direct Sound*. For example, the maximum frame rate of the game can be higher than 70 Hz using a Pentium II computer. Actually, their performance is far enough for our application.
4. At the beginning, we failed to use *WinSock* and *Direct X* libraries together. It is because there are some bugs in the initialization of the *WinSock* and *Direct X* object. Since there are too many lines of code in *Plane* and the structure of program is so complex, we faced problems in finding the bugs. Therefore, we decide to write the 4th testing program – *Ball*.

Ball's structure is far simpler than *Plane*. Therefore, we can focus on finding the reason why we failed to combining the libraries.

Finally, we succeed to use *WinSock* and *Direct X* libraries together in *Ball* and *Plane*.

5. *Spt file* is actually a text file. There are about 40 frames in a plane's *spt file*. It is so exhausting to input all the frame's information by using a text editor. So we have built a *Frame Engine* to manipulate the data of frames.

We will have a introduction on the *Frame Engine* at Chapter ?.

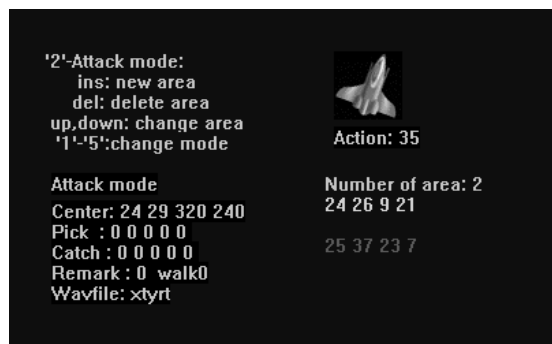


Fig.7.4 Snapshot of the frame engine for *Plane*

8. Ball (Program 4)

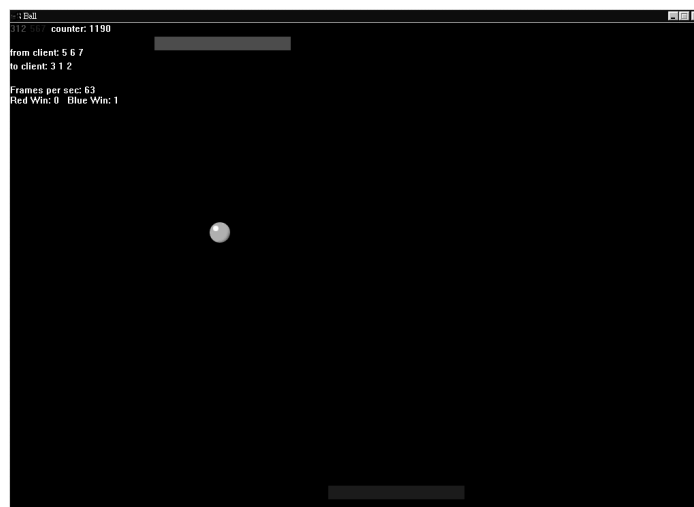


Fig 8.1 Snapshot

8.1 Introduction

Ball is a 2 players' game. It is very simple. Each player can control a bar. The bar can only move left or right. There is a ball in the middle of the screen. It will rebound when hitting the bar. If a player fails to rebound the ball, he will lose.

8.2 The aim of this writing this program:

1. Try to use *Winsock*, *Direct Draw* and *Direct Sound* together in the same program.
The structure of Ball is simple and it has so few of codes. So we can focus on how to combine those libraries together.
2. Try to implement the **synchronous connection (1 to 1)**.
3. **Evaluate the synchronous connection in the wire and wireless network** Because the graphics used in *Ball* is simple and small, the bottleneck of the frame rate is the latency of the network. So we can evaluate the latency of the network connection by counting the frame rate.

8.3 The pseudo-code of *Ball*:

```

Main()
{
  Initialize WinSock
  Initialize Direct Draw
  Ask Player the opponent's IP address
  Connect to the other side
  Wait until Connected

  Loop
  {
    get the key stroke from keyboard
    Send the key stroke to the opponent
    get the key stroke from the opponent
    Calculate the new position of the Ball

    for i = 1 to 2
      if player i is pressing Left and bar[i].x > 0
        { bar[i].x -= bar_speed }
      if player i is pressing Right and bar[i].x < screen_width
        { bar[i].x += bar_speed }
      if bar[i] hit the ball
        { calculate the new speed of the ball }
    next i

    if ball.y > bottom_threshold
      {
        player[2].lose ++
        initialize and restart the game
      }

    if ball.y < top_threshold
      {
        player[1].lose ++
        initialize and restart the game
      }

    Clean up the off screen buffer
    Draw the bars on the off screen buffer
    Draw the ball on the off screen buffer
    Flipping /* Update the screen by the offscreen buffer */

  } until (someone quit)

  Shut Down Direct Draw
  Shut Down WinSock
}

```

8.4 Outcome

1. We have **successfully used *WinSock*, *Direct Draw* and *Direct Sound* libraries in the same program.**
2. We have successfully **implement the synchronous connection.**
3. We have **evaluated the the synchronous connection in wire and wireless network.** (See page ? for the result of evaluation)

9. ChatRoom (Program 5)

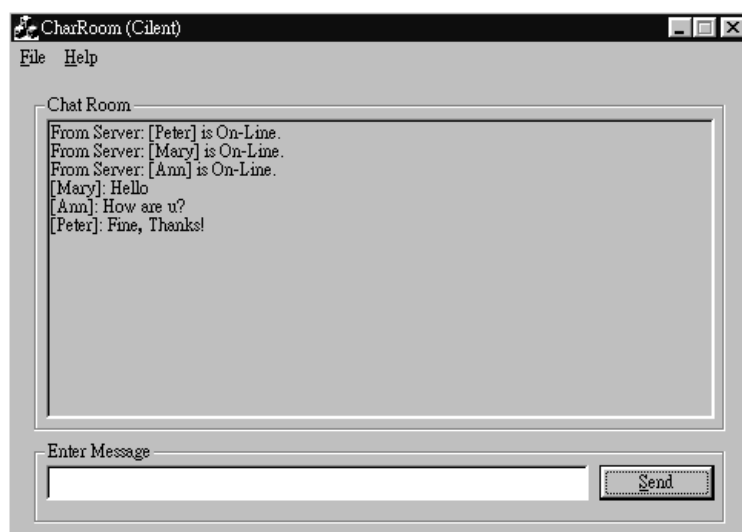


Fig 9.1 Snapshot

9.1 Introduction

“ChatRoom” is the first multi-client program we try to write. The whole system has two components, the server program and the client program. The server program is a **generic server**. It will broadcast a message it receives to all clients, which has registered in this server. The clients will show up the message received from server in front of user screen.

The system is mainly use of create a Chat Room over a Wireless LAN.

9.2 The aim of this writing this program:

1. Try to **use write a multi-client program**

All the testing programs we wrote before are only one to one programs. That means only two users can be involved. Of course, this can't be accepted. Thus, we want to try to write program in “multi-client” mode.

2. Try to write a **generic server**

The server programs we write before is all specific server programs. That means when the goal of the program is change, we need to re-write the server again. We try to write a generic server to handle this job more easily.

3. Try to **design a protocol** for a application

For the generic server to work probably, we need to design a new protocol to communicate between to client and the server. Also, if a standard protocol is defined, several extensions can be achieved. Those extensions may be encryption can be applied, our people can write program to communicate with the server.

9.3 The new protocol and the generic server

In our system, when client want to use the services provide by the server, it need to register to the server first. A message **“Client want to close” (send as raw bit stream of data over the network)** is send from the client to the server. When the server get this message, it will check the client cab be register on this server or not. If the client has the authority to register, a message **“You can connect.” follow by 4 bytes of connect_ID** is send back to client. The connect_ID is and identity of client in the server.

For normal message (string), the client need to append its connect_ID that the back of the message. This is used for the server to know which client is sending. If no connect_ID is appended, server will ignore this message. Otherwise, it will broadcast. Similarly, if the client want to disconnect, it also needs to send a message **“Client want to close” with 4-byte connect_ID append at the back of the message**. This message is needed because the server need to know which client is going to dis-connect.

Advantage of this approach

1. The server can control which client can be register on this server. Thus, the control flows is on the server side.
2. If protocol is standardizing later, the program can be extension very easily by using the same protocol.

Disadvantage of this approach

1. The protocol is too simple! It will easily hack by the hackers. We will try to improve it by applying some encryption on it.

9.4 The pseudo-code of *ChatRoom-Client Program*

```

Main()
{
  Initialize WinSock
  Ask user for the server IP address
  Connect to the server
  Register to the server
  Wait until server reply

  Loop if ( register succefully)
  {
    dispatch event;

    for event ( user input a string )
      change the string to message the server understand;
      send the message to server;

    for event ( receive something from server )
      update user screen;

  } until (client quit or server quit)
  Shut Down WinSock
}

```

9.5 The pseudo-code of *ChatRoom-Server Program*

```

Main()
{
  Initialize WinSock
  Set a socket to accept client connect

  Loop
  {
    dipatch message;

    if message is ( client want to connect )
      try to check the client can connect or not
      if can connect
        tell the client register successfully;
      else
        tell the client register fail

    if message is ( client want to disconnect )
      remove client information from the client list store in server;
      tell the client it can disconnect;

    if message is ( string from cleint )
      send to string to all client which had regsiter on this server;

  } until (server quit)

  tell all the client to disconnect;
  Shut Down WinSock
}

```

9.6 Outcome

1. We have **successfully used multi-client program**.
2. We have **successfully design a protocol** for our application. Although it is just a very simple protocol, it is enough for us to use right use. Later, we will try to improve it in server fields (i.e. uses encryption on the message, minimize the bandwidth it used, etc.)
3. We have **successfully written a generic server**. We can **re-use it** later in our integrated system – **“Fun with Learning English”**.

10. Fun with Learning English (FWLE)

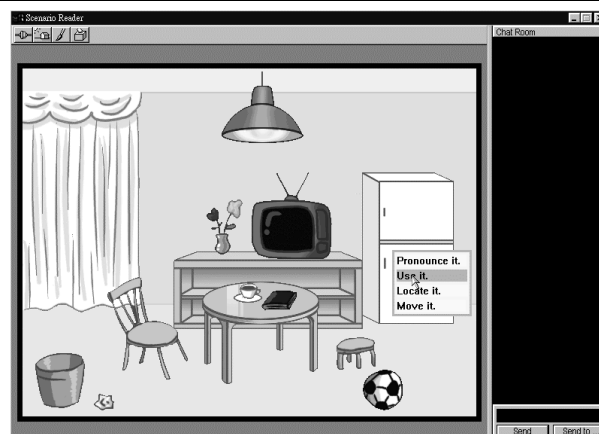


Fig 10.1 Snapshot

10.1 Introduction

Fun with Learning English (FWLE) is a CAL for wireless environment. Its target is primary school student and its aim is to provide a funny and wireless learning environment.

There is 2 modes running *FWLE*, the online mode and offline mode.

In online mode, we assume teacher and student use *FWLE* in the classroom. At teacher will use the server program and student will use the client program.

Some features in online mode:

1. Chat Room

Teachers and students can talk and exchange information through this chat room.

2. Voting (not finished)

The teacher can post a voting topic to the students. After student's reply, the server will calculate the statistics of the result and send to all people. The same technique can be applied to MC question.

3. Write board (not finished)

Teachers and students can draw on a write board. This "white board" may be writable or read-only by others. Teachers can explain some idea by drawing the picture through the write board. The student can save the write board at any moment.

4. Scenario Reader

Student can use *Scenario Reader* to open the *scenario files* which are prepared by teachers. A scenario is a teaching material which consists of many interactive objects. Student can learn by interacting with the objects. Student can also interact with teacher/students through the object of Scenario. We will have an example scenario later in this chapter.

In offline mode, only scenario reader is support.

10.2 Interface of Fun with Learning English

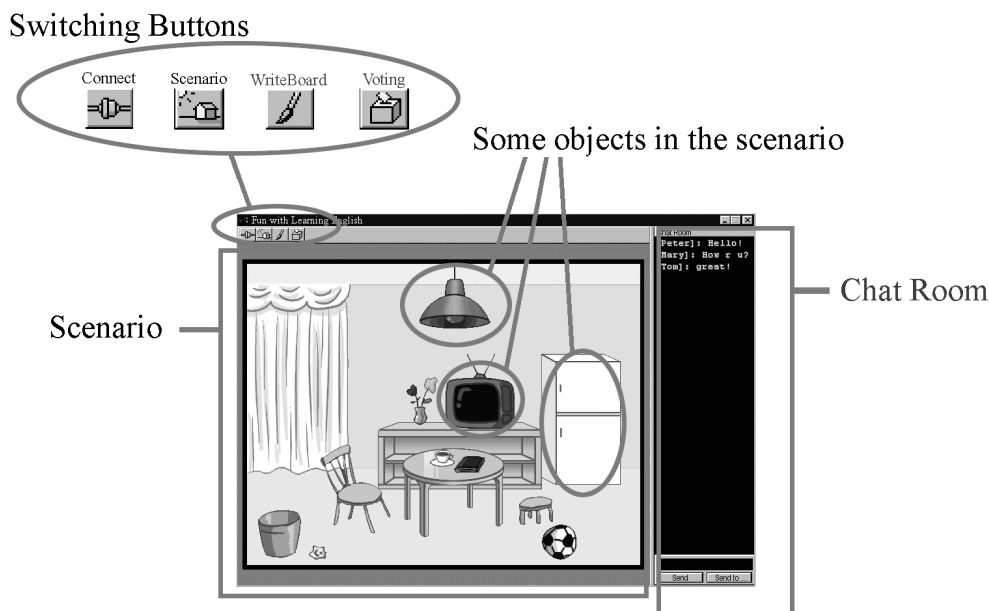




Fig. 10.2 Interface of Fun with Learning English


Switching Buttons:

Connect : 

This button is reserved for future use. It is for network connection purpose.

Scenario: 

This button is reserved for future use. It is switching to scenario mode from the write board mode.

Writeboard: 

This button is reserved for future use. It is switching to write board mode from the scenario mode.

Voting: 

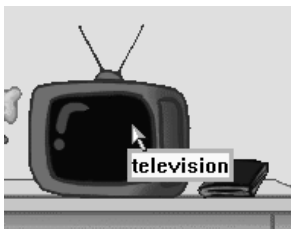
This button is reserved for future use. It is for posting/replying a voting request.

10.3 Demonstration of a Scenario

We have create one scenario , called “Home”, for FWLE.

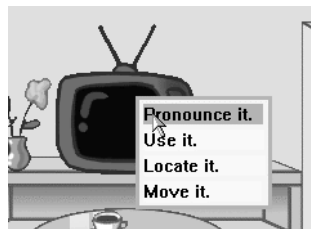
Student can learn vocabulary, pronunciation, verb and preposition in this scenario:

1. Vocabulary



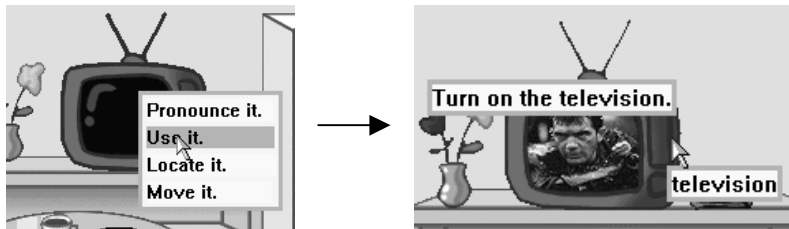
When the cursor is pointing on an object, its name will be shown.

2. Pronunciation

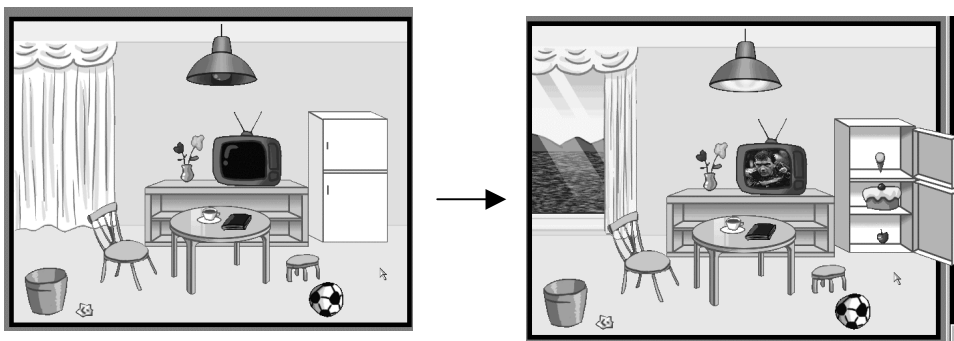


When u click on the object, a command menu will be pulled out. You can hear the pronunciation of the object by clicking “Pronounce it”.

3. Verb



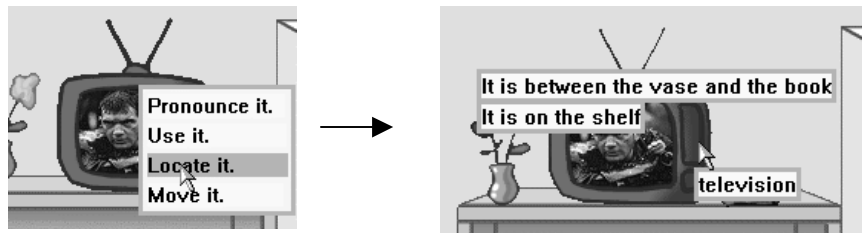
If u click on “Use It”, you can use the object. A message will shown stating what action u have done on that object.



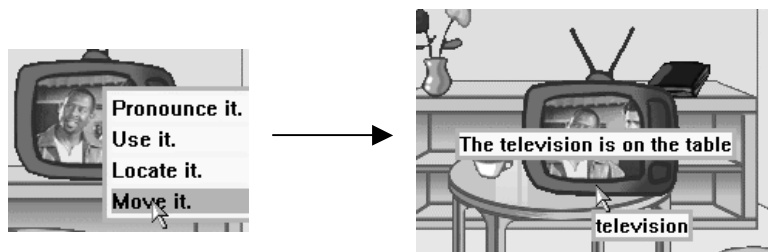
You can also use other objects in the scenario.

10.3 Demonstration of a Scenario (Cont'd)

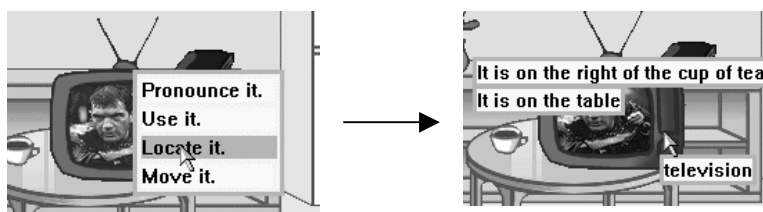
4. Preposition



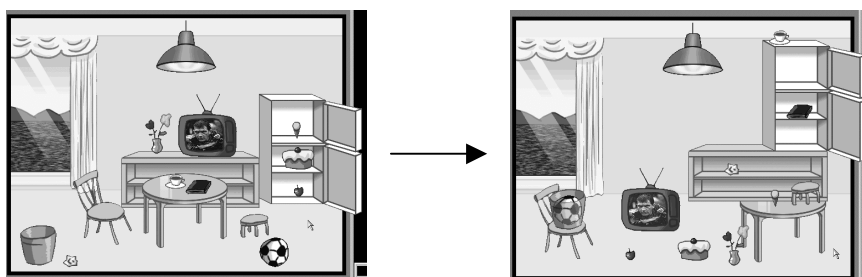
If u click on “Locate It”, a message will be shown to state the location of the object. Student can learn how to use preposition to describe the location of the object by reading this message.



If u click on “Move It”, you can drag the object to other place. After placing the object, a message will be shown to state where the object has been put on/in.



The message stating the location is generated at real time. After placing the object in the new position, you can ask for its location again.



You can also move other objects in the scenario.

10.4 The aim of this writing this program:

1. We want to integrate all of the techniques we learnt in this semester to write an educational application for wireless environment. Here is the list of the integrated components:

- A. Direct Draw**
- B. Direct Sound**
- C. WinSock**
- D. Use spt files to represent objects**
- E. Use of Frame Engine**
- F. ChatRoom**
- G. Use of Generic Server**
- H. Asynchronous connection**

2. We try to design a software which is suitable for future world.
We have set some goals for FWLE

- A. Fancy and User friendly interface**
- B. Suitable for a wireless environment**
- C. Use of Multi-media**
- D. Allow teacher to design some Interactive Teaching material**
- E. Provide a good Interaction between teacher and students**
- F. Maximize the potential use of our software**

10.5 The pseudo-code of FWLE:

```

Main()
{
    Initialize WinSock
    Initialize Direct Draw
    Ask Player the opponent's IP address
    Connect to the other side
    Wait until Connected
    Initialize_scenario

    Loop
    {
        Clean up the off screen buffer

        get the state of mouse

        If click on scenario button
            Switch to scenario mode

        If click on write board button
            Switch to write board mode

        /* If click on voting button */
        /*     call configure_voting()     */

        If it is scenario mode
            call Scenario_Reader()

        If it is write board mode
            /* call write_board () */

        /* Voting, write board is not finished */
        Flipping /* Update the screen by the offscreen buffer */

    } until (user quit)

    Shut Down Direct Draw
    Shut Down WinSock
}

Procedure Scenario_Reader()
{
    get the state of mouse

    check if the cursor is pointing an object

    if menu is showing and not moving an object
    {
        if clicking on "Pronounce" it
            play the wav file of the pointing_object
        if clicking on "use" it and the object is usable
            change the state of the pointing_object and print out the use message
        if clicking on "locate" it
            print out position_of ( pointing_object )
        if clicking on "move" it
            set the pointing_object as a moving object
    }
}

```

10.5 The pseudo-code of FWLE: (Cont'd)

```
if moving an object
{
    check if the object can be place in the current position
    if cannot place
        printtext "Cannot place here"

    if can place and use has click the mouse
    {
        update the position of moving object
        clear the moving flag
    }
}

if menu is not shown and not moving an object
    show the menu if click on an object

Draw the object on the screen
}
```

10.6 Algorithm

1. Animating the object:

In FWLE, there will be animation on some object.

We have attached 2 attributes called "wait_time" and "wait_goto" in the spt file for animating the object. Each frame has its wait and goto value. "Wait_goto" specifies which frame will go in the animation . Wait_time specifies the time to wait before jump to next frame.

2. Make the object "usable":

In FWLE, some objects will change its state when being used.

We have attached 2 attributes called "use_goto" and "use_message" in the spt file for "use" function. Each frame has its "use_goto" and "use_message". "Use_goto" specified which frame will go when being used. "Use_message" describes the "use" action.

10.6 Algorithm (Cont'd)

Here is an example:

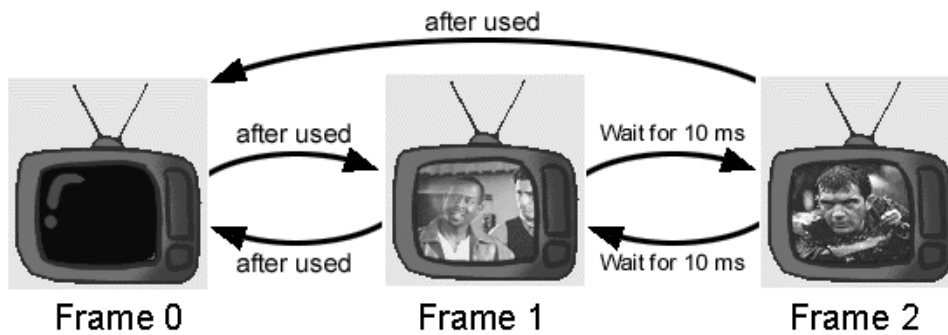


Fig 10.3 DFA of Television in FWLE

Content in the spt file:

	Frame 0	Frame 1	Frame 2
Wait_goto	0	2	1
Wait_time	0 ms	10 ms	10 ms
Use_goto	0	1	1
Use_message	"Turn on the TV"	"Turn off the TV"	"Turn off the TV"

3. "Position Tree":

In FWLE, objects can be put onto or into other object.

We have defined a data structure called position tree to implement "on" and "in" relations between objects.

Here is an example:

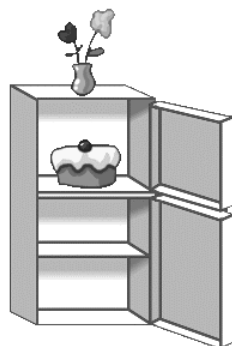


Fig 10.4 Some objects in the scenario

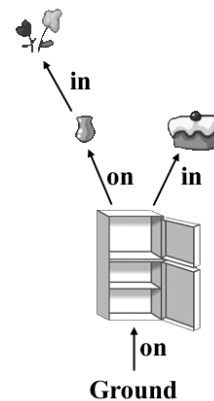


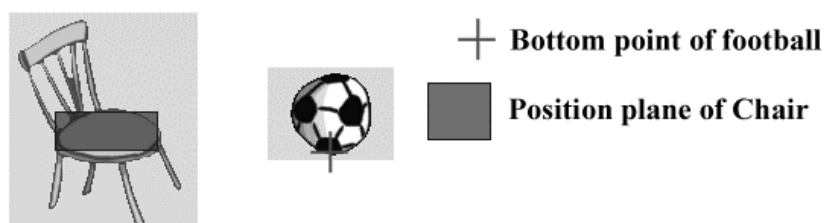
Fig 10.5 The position tree for fig 10.4

10.6 Algorithm (Cont'd)

4. "Position Plane" and "bottom point"

In FWLE, when objects is being moved, we use position plane and bottom point to detect whether the object can be placed at the current position.

Here is an example:



The football can be put on the chair only if its bottom point hits the position plane of chair. Actually, a position plane is a region with on/in flag and a capacity. The on/in flag stating whether the plane is a "on" or "in" relation. An object is not allowed to put on a plane with capacity larger than its volume.

5 Generate the position message in real time

In FWLE, we can ask for the position of objects. FWLE can detect the "on", "in", "on the left of", "on the right of", "between", "in front of", "behind" relation. "on" and "in" relation is detected by searching the position tree. The other relations are detected by comparing the slope of the position difference with the objects on/in the same position plane.

Here is a pseudo-code of procedure for detecting "on the left of":

```

On_the_left_of ( int detecting )
{
    for i = all objects on/in the same plane with object[detecting]
        if ( object[i].x > object[detecting] and
            (x_difference(i, detecting) / y_difference(i, detecting)) > 3 )
            object[detecting] is on the left of object[i]
    next i
    If no object find, return null
    Else return the object with closest distance.
}
    
```

The algorithm for "on the right of", "between", "in front of", "behind" is similar.

10.7 Outcome

We successfully integrate all the following components into FWLE.

- A. Direct Draw
- B. Direct Sound
- C. WinSock
- D. Use spt files to represent objects
- E. ChatRoom
- F. Use of Generic Server
- G. Asynchronous connection

We have written a object editor (OE) for creating FWLE's objects. Our aim for writing the OE is to provide a convenient way for teacher to create the scenario. We can editor all of the attribute of an object using OE.

Actually, OE is modified from the frame engine of plane. At this stage, OE is not so user friendly. We are going to improve it in future.

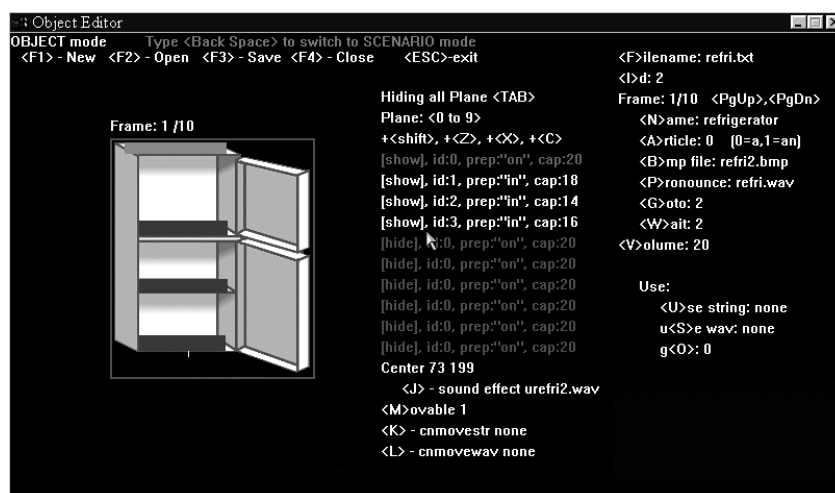


Fig 10.6 Snapshot of object editor

Actually, FWLE is not finished yet. The voting and writing board are just at the early stage. Moreover, we have founded some possible extensions to FWLE. It will be discussed in the next chapter.

11. Our Future Plan

Of course, our FYP project is still not finished. Up to this state, we have some ideas about what is our next move. Although we may not reach all the following goals in next semester, here is some extension we hope to try:

1. Try to integrate Video Streaming Function into FWLE.

If our FWLE can have Video Streaming Function, we can try to use our FWLE to create a new style of learning – students can have lessons outside the classroom. Imagine a geography lesson which needs to study rock that can be carried out at the beach. Also, for medical students, they can see their teacher doing an operation more clearly if there is a camera zooming the operation and all of them carry a thin client device outside the room.

2. Try to implement our FWLE to work with a database server.

Up to now, our FWLE have just a few object. So, we can still manipulate it by programming. However, if the number of objects in our FWLE system increases, to say, the whole physics experiment book of secondary school, then our coding can't cannot handle all objects well. Thus, we hope to implement our FWLE system with a database server.

3. Try to extent our project to other thin client devices such as PDA and HPC.

For now, the client program we develop is mainly executing on a notebook running Win98. However, if a thinner client such as Palm Pilot or HPC running WinCE, is used, the user can carry the machine more conveniently. Thus, we also hope that we can extent our system to those thin clients to see whether those environments are more suitable for the new learning style.


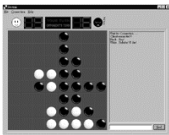

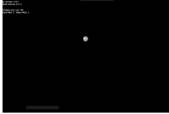
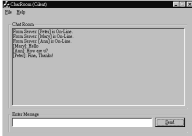

References

1. **MSDN Online:** <http://msdn.microsoft.com/default.asp>
2. **WaveLAN:** <http://www.wavelan.com/products/>
3. **Microsoft DirectX:** <http://www.microsoft.com/directx/>
4. Ralph Davis. **Win32 Network programming: Windows 95 and Windows NT network programming using MFC.** Addison-Wesley, c1996.
5. Bob Quinn, Dave Shute. **Windows sockets network programming.** Addison Wesley Pub. Co., c1996.
6. Stevens. **TCP/IP Illustrated Volume 1: The Protocols.** Addison Wesley. Feb 1998.
7. Pat Bonner. **Network programming with Windows Sockets.** Upper Saddle River, NJ : Prentice Hall PTR, c1996.
8. Martin Heller. **Advanced Win32 programming.** New York: Wiley, c1993.
9. R. Yavatkar, D. Hoffman, Y. Bernet, and F. Baker. SBM(Subnet Bandwidth Manager): **A proposal for Admission Control over IEEE 802-style networks. Internet Draft**

Appendix A. Statistics of programs

Component	Number of files	Number of pictures drawn	Number of lines in the source code
DirectDraw Library	6	0	1435
Class Sprite	2	0	296
Class Frame	2	0	247
DirectSound Library	2	0	348
WinSock Library	2	0	576
WinTalk	8	0	1372
Reversi	8	7	1864
Plane	24	50	2426
Ball	24	1	840
Chat Room	8	0	3125
Generic Server	1	0	756
FWLE	26	42	8650
Total	113	100	24261

Appendix B. Progress Report

Date	Description
June, 99.	Evaluating among different OS (WinNT, Win95/98, WinCE and Linux)
June, 99.	Evaluating among different programming language (Visual C++, Java)
start at June, 99	Studying Direct X
start at June, 99	Studying Winsock
12 nd - 14 th July, 99	Trying to setup a intranet
15 th July, 99	Trying the Wireless Devices
14 th July, 99	 The first testing program - WinTalk, released
25 th July, 99	The second testing program - "Apple Chess", released
10 th Aug, 99	 The third testing program - "Reversi" (Actually it is a newer version of Apples Chess), released
11 th - 30 th Aug, 99	Build our Direct Draw library (graphical library)
1 st - 14 th Sep, 99	 Using our Direct Draw library to write the forth testing program - "Plane"
15 th Sep, 99	Start to build our Direct Sound library (audio library)
22 nd Sep, 99	Add the audio library to "Plane".
6 th Oct, 99	 Write a new game "Ball", to test combining Winsock and Direct Draw together.
9 th Oct, 99	Add the Winsock to "Plane"
11 th - 16 th Oct, 99	Studying Multi Client for WinSock
18 th - 20 th Oct, 99	Design the structure of Chat Room - WinChat and a generic server for chat room.
21 th - 26 th Oct, 99	 Write a Chat Room - WinChat using WinSock
24 th Oct - 15 th Nov, 99	Construct the scenario editor and preparing the spt file for FWLE
12 th - 20 th Nov, 99	 Write the scenario reader for FWLE