ปีการศึกษา 2538

การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่าย

โดย

นาย ธรรมนูญ สัทธานนท์

นาย วุฒิเจตน์ อร่ามดิลกรัตน์

อาจารย์ที่ปรึกษา

สมศักดิ์ วลัยรัชต์

ปริญญานิพนธ์ปีการศึกษา 2538

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่าย

ผู้จัดทำ

1.นาย ธรรมนูญ สัทธานนท์

2.นาย วุฒิเจตน์ อร่ามดิลกรัตน์

..... อาจารย์ที่ปรึกษา

(สมศักดิ์ วลัยรัชต์)

การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่าย

ธรรมนูญ สัทธานนท์
วุฒิเจตน์ อร่ามดิลกรัตน์
สมศักดิ์ วลัยรัชต์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2538

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการพัฒนาโปรแกรมประยุกต์บนระบบเครือข่าย ระบบเครือข่ายที่ทำการศึกษาคือระบบเครือข่ายอินเตอร์เน็ต (Internet) ซึ่งมีผู้ใช้บริการอย่างแพร่หลาย โดยมีโปรโตคอลที่ใช้ในการสื่อสารคือโปรโตคอล TCP/IP ระบบปฏิบัติการที่เลือกในการพัฒนาโปรแกรมประยุกต์นี้คือระบบปฏิบัติการวินโดวส์ 3.11 พร้อมทั้ง WinSock ซึ่งเป็น API (Application Program Interface) ที่ใช้กับการสื่อสารของโปรโตคอล TCP/IP ของระบบปฏิบัติการวินโดวส์

โครงงานนี้ได้ทดลองพัฒนาโปรแกรมเกมหมากรุกไทยสำหรับผู้เล่น 2 คน โดยเล่นผ่านระบบเครือข่ายอินเตอร์เน็ตมาเป็นตัวอย่างของการพัฒนาโปรแกรมประยุกต์ เครื่องมือแปลภาษาที่ใช้ได้แก่ Microsoft Visual C++ 1.5 โดยมุ่งที่จะออกแบบโปรแกรมประยุกต์ให้เป็นลักษณะของการโปรแกรมเชิงวัตถุ เพื่อเป็นหลักในการพัฒนา และศึกษาในสร้างโปรแกรมประยุกต์บนระบบเครือข่ายสำหรับผู้สนใจต่อไป

Network Application Development

Thammanoon Sattanon

Wutijat Aramdilokrat

Somsak Walairacht Advisor

1995

ABSTRACT

This thesis presents the development of an applicatoin on network systems. We'll focalize on the Internet system which is interesting. And it has owned protocol named TCP/IP. The selected operating system is Windows 3.11 comes with WinSock which is API (Application Program Interface)used with the TCP/IP communication protocol of Windows operating system.

This project develloped Thai Chess game for two players to play in the Internet. That is the example of applications development. We used Microsoft Visual C++ 1.5 in order to design by using Object-Oriented Programing desing. This will be the good example for interested people.

สารบัญ

บทที่ 1
บทนำ

ในปัจจุบันระบบเครือข่ายอินเตอร์เน็ต (Internet) เริ่มมีบทบาทสำคัญต่อวิถีชีวิตของผู้คนโดยทั่วไปมากขึ้น จะเห็นได้จากการที่โปรแกรมบนระบบเครือข่ายอินเตอร์เน็ตเริ่มมีแพร่หลายมากขึ้น สำหรับประเทศไทยนั้นระบบอินเตอร์เน็ตนี้เริ่มขยายตัวออกไปสู่ประชาชนทั่วไปมากขึ้น นอกเหนือจากที่ใช้กันในหมู่อาจารย์ และนักศึกษาในมหาวิทยาลัย ผู้จัดทำเล็งเห็นว่าโปรแกรมประยุกต์บนระบบเครือข่ายอินเตอร์เน็ตของไทยในปัจจุบันยังมีน้อยจึงได้คิดที่จะสร้างโปรแกรมประยุกต์ขึ้นมาซักโปรแกรมหนึ่ง โดยจะต้องแสดงเอกลักษณ์ความเป็นไทยไว้ด้วย จึงได้คิดที่จะทำโปรแกรมหมากรุกไทยที่สามารถเล่นผ่านระบบเครือข่ายอินเตอร์เน็ตขึ้นมา และด้วยความสามารถของอินเตอร์เน็ตนี้เองทำให้ผู้เล่นสามารถเล่นจากที่ใดก็ได้ที่เชื่อมต่อเข้ากับระบบเครือข่ายอินเตอร์เน็ต

บทที่ 2
ความรู้เบื้องต้นเกี่ยวกับระบบเครือข่าย และการโปรแกรม

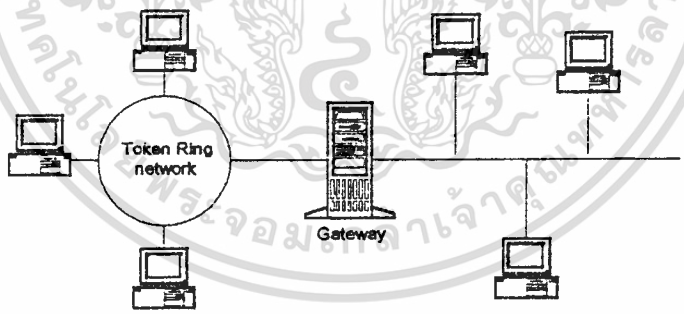
2.1ระบบเครือข่าย และการโปรแกรมบนระบบเครือข่าย

การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายจำเป็นต้องมีความรู้พื้นฐานทางด้านระบบเครือข่ายพอสมควรในหัวข้อนี้จะอธิบายกับคำศัพท์พื้นฐาน และหลักการของระบบเครือข่ายที่จำเป็นสำหรับการทำความเข้าใจในการพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายดังจะกล่าวต่อไป

2.1.1 ความหมายของระบบเครือข่ายคอมพิวเตอร์ และอินเตอร์เน็ตเวิร์คกิ้ง (Internet-working)

ระบบเครือข่ายคอมพิวเตอร์ (Computer Network) คือระบบการเชื่อมต่อระหว่างระบบปลายทาง (End-System) ซึ่งระบบปลายทางเป็นระบบที่เป็นอิสระจากกัน (Autonomous) ระบบปลายทางสามารถเป็นได้ตั้งแต่ไมโครคอมพิวเตอร์ (Microcomputer) ไปจนกระทั่งซุปเปอร์คอมพิวเตอร์ (Supercomputer) ขนาดใหญ่เพื่อจุดมุ่งหมายในการแลกเปลี่ยนข้อมูลและการแบ่งปันทรัพยากรของระบบเช่น ไฟล์ (File) ข้อมูล, เครื่องพิมพ์ (Printer) , โมเด็ม (Modem) ตลอดจนการให้บริการฐานข้อมูลร่วม (Sharing database)

อินเตอร์เน็ทเวิร์คกิ้ง หรืออินเตอร์เน็ท (Internet) คือการเชื่อมต่อของระบบเครือข่าย 2 เครือข่ายขึ้นไป ดังนั้นคอมพิวเตอร์บนระบบเครือข่ายหนึ่งก็สามารถติดต่อกับคอมพิวเตอร์บนระบบเครือข่ายอื่นๆ ได้ เช่น เน็ตเวิร์คโทเคนริงค์ (Tokenring network) เชื่อมกับเน็ตเวิร์คอีเทอร์เน็ท (Ethernet network) โดยมีเกตเวย์ (Gateway) เป็นตัวเชื่อมดังภาพ



ภาพที่ 2.1 ระบบเครือข่ายคอมพิวเตอร์เบื้องต้น

2.1.2 รูปแบบของโปรแกรมบนระบบเครือข่าย (Network Programming Models)

จากหัวข้อ 2.1.1 ได้ให้ความหมายของระบบเครือข่าย แสดงถึงวิธีการที่ระบบคอมพิวเตอร์ใดๆ จะทำการเชื่อมต่อกับระบบเครือข่าย แต่ว่าคอมพิวเตอร์ที่เชื่อมต่ออยู่กับระบบเครือข่ายนั้นมีวิธีการอย่างไรในการแลกเปลี่ยนข้อมูล และแบ่งปันทรัพยากรเหล่านั้นได้ ทำให้ต้องมีโปรแกรมประยุกต์ซึ่งสามารถที่จะจัดการในสิ่งที่กล่าวมาแล้วอย่างเหมาะสม ซึ่งในหัวข้อนี้จะกล่าวถึงรูปแบบของโปรแกรมบนระบบเครือข่าย 2 รูปแบบคือไคลเอนต์เซิร์ฟเวอร์ (Client/Server Computing) และการประมวลผลแบบกระจาย (Distributed Computing)

2.1.2.1 การประมวลผลแบบไคลเอนต์เซิร์ฟเวอร์ (Client/Server Computing)

การประมวลผลแบบไคลเอนต์เซิร์ฟเวอร์นี้การประมวลของโปรแกรมประยุกต์จะแบ่งออกเป็น 2 ส่วน คือ ส่วนฟรอนต์-เอ็น (front-end) ที่ทำงานบนไคลเอนต์ ส่วนนี้จะทำหน้าที่แสดงผลที่ได้จากการประมวลผล และรับข้อมูลจากผู้ใช้ อีกส่วนหนึ่งคือแบ็คเอนด์ (back-end) ทำงานบนเซิร์ฟเวอร์มีหน้าที่ในการเก็บรวบรวม และจัดการข้อมูลจากฟรอนต์เอนด์ในรูปแบบการประมวลผลไคลเอนต์เซิร์ฟเวอร์นี้ เครื่องเซิร์ฟเวอร์มักจะเป็นเครื่องที่มีความสามารถสูงกว่าเครื่องไคลเอนต์ โดยปกติเครื่องเซิร์ฟเวอร์มักจะเป็นเครื่องเมนเฟรม หรือมินิคอมพิวเตอร์ และเครื่องไคลเอนต์มักจะเป็นเครื่องคอมพิวเตอร์ส่วนบุคคล ซึ่งการติดต่อกันระหว่างส่วนฟรอนต์เอนด์ และแบ็คเอนด์ ทำโดยผ่านระบบเครือข่ายซึ่งในทางปฏิบัติแล้ว ในส่วนแบ็คเอนด์ที่อยู่บนเซิร์ฟเวอร์ จะเป็นฝ่ายให้บริการแก่งานของไคลเอนต์หลายงานในเวลาเดียวกัน

2.1.2.2 การประมวลผลแบบกระจาย (Distributed Computing)

การประมวลผลของโปรแกรมประยุกต์มีรูปแบบการประมวล 2 แบบ คือ พรีคอลเล็คชั่น (precollection) และการประมวลผลแบบขนาน (parallel processing) ดังนี้

1. พรีคอลเล็คชั่นเป็นลักษณะการทำงานที่ข้อมูลที่ต้องการในการจัดเก็บ และส่งต่อไปทั่วระบบเครือข่ายอยู่ตลอดเวลาอย่างสม่ำเสมอ การทำงานในลักษณะนี้จะเหมาะสมกับงานบางงาน เช่น ต้องการเก็บสถานะของคอมพิวเตอร์ที่อยู่ในระบบเครือข่ายหนึ่ง ๆ ทุกเครื่อง
2. การประมวลผลแบบขนาน การประมวลผลในลักษณะนี้งานใด ๆ จะถูกประมวลผลด้วยคอมพิวเตอร์หลาย ๆ เครื่อง โดยเครื่องคอมพิวเตอร์เหล่านั้นสามารถจะติดต่อกันโดยระบบเครือข่าย เช่นการทำการพัฒนาโปรแกรมประยุกต์ขนาดใหญ่ โดยทีมพัฒนาที่มีผู้พัฒนาหลายคน สามารถลดเวลาของการแปล และการรวมโมดูล (module) ต่าง ๆ เข้าเป็นโปรแกรมเดียวกันโดยการแบ่งงานการแปลโมดูลเหล่านั้นแก่คอมพิวเตอร์ในระบบเครือข่ายทำการแปลในเวลาเดียวกัน

2.1.3 โมเดล OSI

OSI ปีนคำย่อที่มาจากคำว่า Open Systems Interconnection โดยที่เป็นมาตรฐานที่ถูกเสนอขึ้นโดย International Standards Organization ซึ่งเป็นองค์กรที่จัดตั้งขึ้นมาเพื่อดูแล และส่งเสริมตลอดจนกำหนดมาตรฐานของการติดต่อสื่อสารของระบบเครือข่ายคอมพิวเตอร์ โดยโมเดล OSI นี้มีลักษณะเป็นสถาปัตยกรรม แบบระบบเปิด (Open System) เพราะมุ่งที่จะให้ระบบคอมพิวเตอร์ในหลาย ๆ รูปแบบที่แตกต่างกันสามารถเชื่อมต่อกันได้ OSI โมเดลได้แบ่งโปรโตคอล (protocol) ในการสื่อสารออกเป็น 7 เลเยอร์ (layer) ซึ่งโปรโตคอล คือชุดของกฎ หรือข้อตกลงในการติดต่อ ข้อสังเกตโมเดล OSI เป็นเพียงข้อเสนอแนะ มิใช่ข้อกำหนด และควรรู้ว่ายังไม่มีระบบการเชื่อมต่อใดที่สร้างเหมือนกับโมเดล OSI จริง ๆ

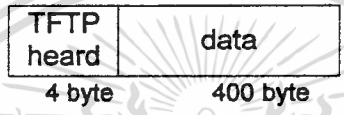
1.Application Layer
2.Presentation Layer
3.Session Layer
4.Transport Layer
5.Network Layer
6.Datalink Layer
7.Physical Layer

ภาพที่ 2.2 OSI โมเดลทั้ง 7 เลเยอร์

ในหนึ่งชั้นของเลเยอร์ไม่ได้การกำหนดว่าจะต้องมีเพียงหนึ่งโปรโตคอลเท่านั้น ที่อยู่ในระดับเลเยอร์เดียวกัน และในทางตรงข้าม ชุดของโปรโตคอลใด ๆ อาจจะมีมากกว่าหนึ่งเลเยอร์ประกอบกันเป็นข้อกำหนดของระบบเครือข่ายเรียกว่าชุดโปรโตคอล (Protocol Suite) เช่น ชุดโปรโตคอล TCP/IP (Transmission Control Protocol/Internet Protocol) เป็นต้น ประโยชน์ในการแบ่งเป็นเลเยอร์ คือกำหนดการติดต่อระหว่างเลเยอร์ ทำได้โดยไม่ต้องคำนึงถึงการเปลี่ยนในเลเยอร์ใด ๆ ที่ติดกัน

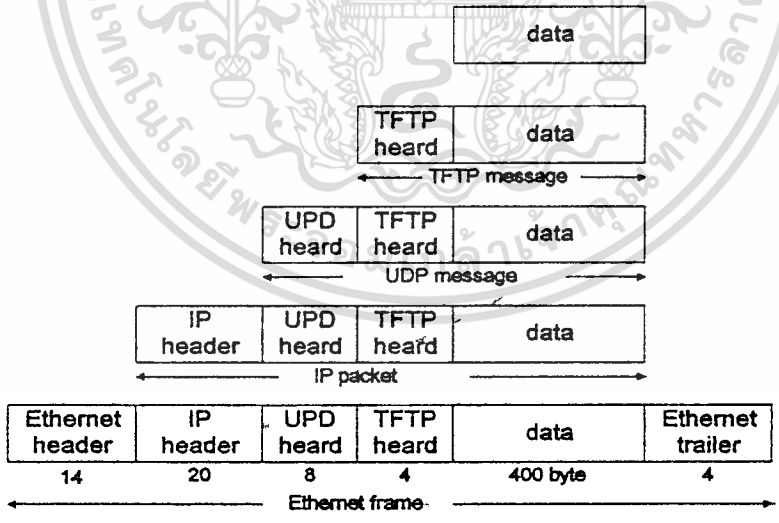
2.1.4 เอ็นแคปซูเลชั่น (Encapsulation)

พิจารณาโปรแกรมประยุกต์ TFIP (File Transfer Protocol) ซึ่งใช้ในใช้ในโปรโตคอล UDP (User Datagram Protocol) ระหว่างสองระบบซึ่งเชื่อมต่อด้วยอีเตอร์เน็ตถ้าโปรแกรมไคล์เอนท์ TFIP มีข้อมูล 400 ไบท์ต้องการส่งไปที่โปรแกรมแชิร์ฟเวอร์ โปรแกรมไคล์เอนต์TFIP จะเพิ่มข่าวสารควบคุม 4 ไบท์เป็นส่วนหัวของข้อมูลก่อนที่จะผ่านข้อมูลไปสู่เลเยอร์ UDP การเพิ่มของข่าวสารควบคุมไปที่ข้อมูลเรียกว่า เอ็นแคปซูเลชั่น ดังแสดงในภาพ 2.3



ภาพที่2.3 การเอ็นแคปซูเลชั่น

เลเยอร์ UDP จะไม่มีการตีความส่วนหัว TFIP 4 ไบท์ งานของเลเยอร์ UDP คือส่งข้อมูล 404 ไบท์ไปสู่เลเยอร์ UDP ของโปรแกรมอีกด้านหนึ่ง จากนั้นเลเยอร์ UDP จะทำการเพิ่มส่วนหัว 8 ไบท์แล้วส่งข้อมูล 432 ไบท์ ไปยังเลเยอร์ดาต้าลิงค์ ที่เลเยอร์มีจะมีการเพิ่มส่วนหัวอีก 14 ไบท์ และส่วนหางอีก 4 ไบท์ ดังภาพ 2.4



ภาพที่ 2.4 ลำดับการเอ็นแคปซูเลชั่น

2.1.5 ลักษณะของการติดต่อ

แบ่งออกเป็น 2 ชนิดคือ

2.1.5.1 Connection-oriented

คือการติดต่อที่ต้องมีการเชื่อมโปรเซสที่จะทำการติดต่อก่อนที่จะมีการส่งหรือรับข้อมูล ซึ่งสามารถใช้คำว่าวงจรเสมือน (Virtual Circuit) เพราะว่าจะทำงานเสมือนมีวงจรต่ออยู่ระหว่างโปรเซส ถึงแม้ว่าข้อมูลนี้อาจจะผ่าน Packet-Switching Network บริการชนิดนี้ส่วนมากจะใช้ในกรณีที่มีข่าวสารต้องการมากกว่าหนึ่งข่าวสาร ดังนั้นสามารถแบ่งขั้นการทำงานออกเป็น

- ขั้นการสร้างการติดต่อ (connection establishment)
- ขั้นการส่งผ่านข้อมูล (data transfer)
- ขั้นยกเลิกการติดต่อ (connection termination)

2.1.5.2 Connectionless หรือดาต้าแกรม (Datagram)

คือจะไม่มีขั้นการสร้างการติดต่อ และขั้นการยกเลิกการติดต่อ แต่จะมีขั้นการส่งผ่านข้อมูลอย่างเดียว โดยข้อมูลเรียกว่าดาต้าแกรมจะถูกส่งจากระบบหนึ่งไปสู่ระบบหนึ่งอย่างเป็นอิสระโดยไม่ขึ้นอยู่กับดาต้าแกรมอื่น

2.1.6 แอดเดรส (Address)

การที่ระบบในระบบเครือข่ายสามารถติดต่อกันได้จำเป็นต้องมีแอดเดรสไว้คล้ายกับหมายเลขประจำตัวซึ่งลำดับของแอดเดรสสามารถพิจารณาได้คือ

- แต่ละเครือข่ายจะต้องมีแอดเดรสสำหรับเครือข่าย
- คอมพิวเตอร์โฮสต์แต่ละเครื่องในเครือข่ายจะต้องมีแอดเดรส
- แต่ละโพรเซสในโฮสต์จะต้องมีหมายเลขประจำตัว

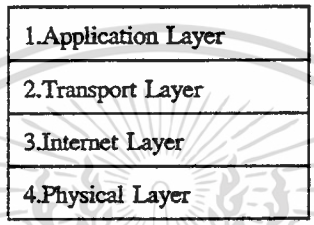
โดยทั่วไปแอดเดรสของโฮสต์จะประกอบด้วยหมายเลขเครือข่าย (Network ID) และ หมายเลขของโฮสต์ (Host ID) ส่วนแอดเดรสของโปรเซสของผู้ใช้จะอยู่ในรูปจำนวนเต็มซึ่งกำหนดโดยโปรโตคอล เช่นโปรโตคอล TCP/IP จะใช้เลขจำนวนเต็มขนาด 32 บิต ในการกำหนดหมายเลขเครือข่าย และหมายเลขของโฮสต์ และทั้ง TCP และ UDP ใช้เลขจำนวนเต็มขนาด 16 บิต เป็นหมายเลขพอร์ตหรือหมายเลขของโปรเซส

ชุดของโปรโตคอลส่วนใหญ่จะมีการกำหนดชุดของแอดเดรสสำหรับการบริการที่เป็นที่รู้กันโดยทั่วกัน เช่นคอมพิวเตอร์ที่โปรโตคอล TCP/IP ส่วนใหญ่จะมี FTP (File Transfer Protocol) ซึ่งไคลเอนต์สามารถติดต่อได้โดยใช้หมายเลขพอร์ตคือ 21

2.2 ความรู้เบื้องต้นเกี่ยวกับโปรโตคอล TCP/IP

2.2.1 เปรียบเทียบระหว่างโปรโตคอล TCP/IP และ OSI โมเดล

การออกแบบโปรโตคอล TCP/IP นั้นไม่ได้เป็นไปตามรูปแบบของ OSI โมเดล เนื่องด้วยถูกออกแบบโดยองค์กรขนาดใหญ่ซึ่งใช้เวลานานในการออกแบบตลอดจนการรับรองมาตรฐานต่างกับโปรโตคอล TCP/IP ที่ถูกแบบด้วยความต้องการอันเร่งด่วนของรัฐบาลสหรัฐ จึงทำให้การพัฒนาโปรโตคอล TCP/IP มีเงื่อนไขของในด้านความต้องการที่ต่างจาก OSI โมเดล ซึ่งหากเรามองโดยรวมแล้วจะเห็นว่าโปรโตคอล TCP/IP มีการแบ่งเป็นเลเยอร์ที่น้อยกว่า OSI โมเดลคือมี 4 ชั้นเท่านั้นดังภาพ 2.5 โดยแบ่งเป็น



ภาพที่2.5 เลเยอร์ของโปรโตคอล TCP/IP

2.2.1.1 แอพพลิเคชั่นเลเยอร์ (Application Layer)

ในเลเยอร์นี้ประกอบโปรแกรมประยุกต์ที่ใช้เครือข่าย เช่น โปรแกรมส่งถ่ายข้อมูล (file-transfer programe) และอาจกล่าวได้ว่าเลเยอร์นี้โปรโตคอล TCP/IP ก็คือ เลเยอร์ในชั้นแอพพลิเคชั่นเลเยอร์ รวมกับชั้นพรีเซนเตชั่นเลเยอร์ (Presentation layer) ใน OSI โมเดลนั้นเอง และในเลเยอร์ชั้นนี้ของโปรโตคอล TCP/IP จะกลืนอยู่ในตัวโปรแกรมประยุกต์

2.2.1.2 ทรานสปอร์ตเลเยอร์ (Transport Layer)

ในชั้นนี้เป็นชั้นที่ให้การส่งข้อมูลจากจุดปลายถึงจุดปลาย หากเปรียบเทียบกับ OSI โมเดล ก็สามารถเทียบได้กับชั้นเซสชั่นเลเยอร์ (Session layer) รวมกับทรานสปอร์ตเลเยอร์นั่นเอง โดยโปรโตคอล TCP/IP มี ซอกเก็ต (socket) เป็นจุดปลาย (end-point) ในการสื่อสาร ซึ่งซอคเก็ตนี้ประกอบไปด้วยหมายเลขของคอมพิวเตอร์ และหมายเลขพอร์ต (port) ของเครื่องที่ต้องการส่งข้อมูลไปถึง ในชั้นนี้มีการรับรองการถึงที่หมาย และลำดับของข้อมูลที่ส่งโดยไม่ซ้ำ และความผิดพลาดข้อมูล

2.2.1.3 อินเตอร์เน็ตเลเยอร์ (Internet Layer)

เลเยอร์นี้มีการกำหนดดาต้าแกรม และทำการหาเส้นทางการส่ง หน้าที่ของเลเยอร์นี้เทียบเท่ากับเน็ตเวอร์กเลเยอร์ (Network layer) และดาต้าลิงค์เลเยอร์ (Datalink layer) ของ OSI โมเดล

2.2.1.4 ฟิสิคอลเลเยอร์ (Physical Layer)

โปรโตคอล TCP/IP ไม่ได้กำหนดรูปแบบของการเชื่อมต่อในระดับนี้ไว้ใหม่ แต่ได้ใช้มาตรฐานที่มีอยู่เดิมที่กำหนดไว้ก่อน เช่น RS232, อีเทอร์เน็ต (Ethernet) เป็นต้น

2.2.2 รูปแบบการกำหนดแอดเดรสของโปรโตคอล TCP/IP

ในหัวข้อนี้จะได้อธิบายรูปแบบการกำหนดแอดเดรสของโปรโตคอล TCP/IP ซึ่งลักษณะแอดเดรสของโปรโตคอลนี้ค่าของแอดเดรสของเครื่องคอมพิวเตอร์ในระบบเครือข่ายจะไม่ซ้ำกันเลย โดยเลขนี้เรียกว่า IP แอดเดรส (IP Address) เป็นเลข 32 บิต ซึ่งแบ่งเป็นคลาส (Class) ตามหลักในการพิจารณาที่จะได้กล่าวต่อไปนี้

2.2.2.1 การแบ่งเน็ตเวอร์กคลาส (Network Classes)

เนื่องจากหมายเลขแอดเดรสของคอมพิวเตอร์เครื่องใด ๆ นั้นจะต้องสามารถบอกถึงความแตกต่างระหว่างตัวเครื่องเองตลอดจนเครือข่ายที่คอมพิวเตอร์นั้นเชื่อมต่ออยู่ด้วย หมายเลข IP แอดเดรส จึงแยกออกเป็น 2 ส่วน ได้แก่ ส่วนที่แสดงหมายเลขของคอมพิวเตอร์โฮสต์ และส่วนที่เป็นหมายเลขของเครือข่าย

การแบ่งคลาสของแอดเดรสทำได้โดยพิจารณาจำนวนบิตของ 2 ส่วนประกอบข้างต้น ซึ่งมีการแบ่งออกเป็น 5 คลาส แต่มีการใช้เพียง 3 คลาสแรก คือ คลาส A ,คลาสB และคลาส C ส่วนคลาส D และ E ถูกสงวนไว้สำหรับจุดประสงค์พิเศษ

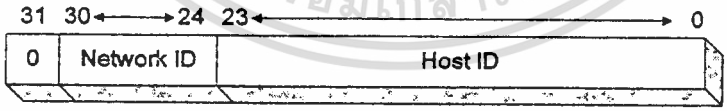
IP address format

ภาพที่2.6 หมายเลข IP แอดเดรส

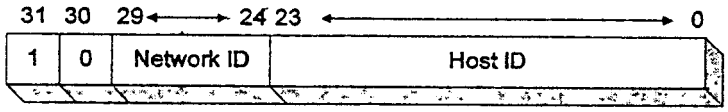
Netwrok Class	Networks	Hosts per Network
A	126	16,777,214
B	16,382	65,534
C	2,097,150	254

ภาพที่ 2.7 แสดงคลาส, จำนวนเครือข่าย และจำนวนโฮสต์ของแต่ละคลาส

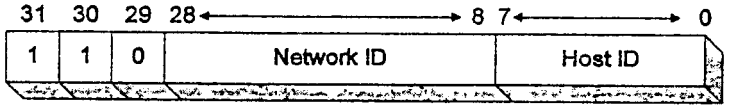
โดยปกติแล้วผู้พัฒนาโปรแกรม ไม่ต้องสนใจความแตกต่างระหว่างคลาสของ IP แอดเดรส



Class A IP address format



Class B IP address format



Class C IP address format

ภาพที่ 2.8 แสดงรูปแบบของ IP แอดเดรสในคลาสต่าง ๆ

2.2.2.2 การแทนด้วยเลขฐานสิบ และจุด (Dotted Decimal Notaion)

เนื่องจากการแทนหมายเลข IP แอดเดรสเป็นเลขฐาน 2 ซึ่งข้างอ่านไม่สะดวกจึงมีการแทนเลขฐานสองเหล่านั้นในรูปเลขฐานสิบ และจุด โดยเลขฐานสิบแต่ละตัวจะแทนเลขฐานสองจำนวน 8 บิต โดยระหว่างเลขฐานสิบแต่ละตัวจะแทรกด้วยจุด ดังนั้นจะต้องใช้เลขฐานสิบ 4 ตัว ในการแทนเลข 32 บิต ที่เป็น IP แอดเดรส ดังตัวอย่างตามภาพ 2.10 จะสังเกตว่าหมายเลข IP แอดเดรสนี้จัดอยู่ในคลาส B โดยหมายเลขของเครือข่ายคือ 166.78 และหมายเลขประจำเครื่องคือ 4.139

Dotted Decimal Notation
166.78..4.139
Binary Representation
10100110 01001110 00000100 10001011

ภาพที่ 2.9 ตัวอย่างหมายเลข IP แอดเดรสทั้งสองแบบ

2.2.2.3 การทำซับเน็ตติ้ง (Subnetting)

การทำซับเน็ตติ้งเป็นการเปลี่ยนแปลงการใช้หมายเลขของเครื่องโฮสต์ และหมายเลขของเครือข่ายในระดับท้องถิ่น โดยในทางตรรกคือการเลื่อนเส้นแบ่งที่แยกหมายเลขเครื่อง และหมายเลขของเน็ตเวอร์กที่อยู่ในหมายเลข IP แอดเดรส โดยที่ปริมาณของหมายเลขเครื่องโฮสต์ และหมายเลขเครือข่ายจะแปรผกผันกัน เช่น หากมีปริมาณของเน็ตเวอร์กมาก ก็จะทำให้เครื่องใด ๆ ที่จะต่อกับระบบเครือข่ายหนึ่ง ๆ จะน้อยลงเป็นต้น ในทางปฏิบัติการทำซับเน็ตติ้งทำโดยการนำซับเน็ตมาร์ค (Subnet mark) คือตัวเลขจำนวน 32 บิต มาทำการกระทำตรรกและ (AND) กับหมายเลข IP แอดเดรส ดังตัวอย่างโดยกำหนดหมายเลข IP แอดเดรส คือ 166.78.4.139 และซับเน็ตมาร์ค คือ 255.255.255.0 ทำการกระทำตรรกและ ดังภาพ

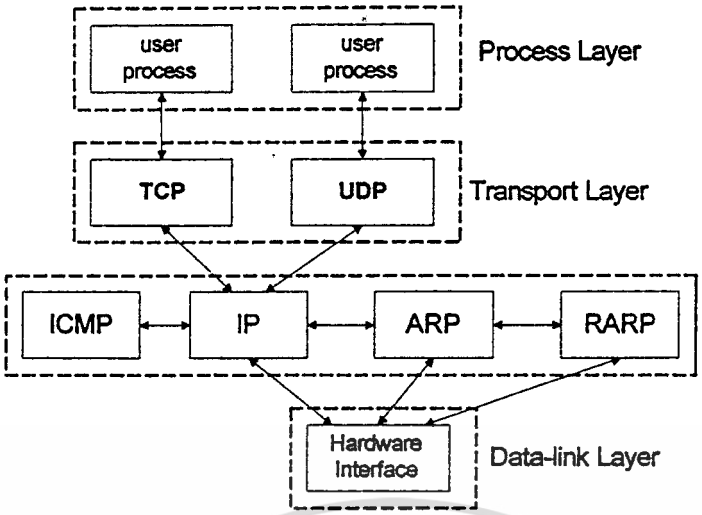
166.78.4.139
AND
255.255.255.0
166.78.4.0

ภาพที่ 2.10 การทำซับเน็ตติ้ง

จะเห็นว่าหากพิจารณาโดยไม่มีการทำซับเน็ตติ้งแล้วจะได้หมายเลขของเน็ตเวอร์กคือ 166.78 และหมายเลขประจำเครื่องคือ 4.139 แต่ผลที่ได้จากการกระทำตรรกและ ในรูปข้างต้น ผลลัพธ์ที่ได้คือ 166.78.4.0 และที่เหลือคือ 139 หากพิจารณาผลลัพธ์ นี้คือหมายเลขเน็ตเวอร์กคือ 166.78.4.0 และหมายเลขเครื่องคือ 139 หรืออาจพูดได้ว่าคอมพิวเตอร์เครื่องนี้มีหมายเลขเครื่องเท่ากับ 139 และอยู่บนเครือข่ายย่อยหมายเลข 166.78.4

2.2.3 ชุดโปรโตคอล TCP/IP (TCP/IP Protocol Suite)

ชุดโปรโตคอล TCP/IP นอกจากมีโปรโตคอล TCP และ IP แล้ว ยังมีโปรโตคอลอย่างอื่นอีก ดังภาพที่ 2.12 แสดงความสัมพันธ์ของชุดโปรโตคอลโดยแบ่งตามเลเยอร์



ภาพที่ 2.11 ความสัมพันธ์ระหว่างชุดโปรโตคอล

2.2.3.1 โปรโตคอล IP (Internet Protocol)

โปรโตคอล IP เป็นโปรโตคอลแบบคอนเนคชั่นเลส (Connectionless protocol) ซึ่งได้กล่าวถึงลักษณะของโปรโตคอลชนิดนี้ไปแล้วในหัวข้อ 2.1.5 โดยที่โปรโตคอล IP ไม่รับประกันว่าข้อมูลที่ส่งจะไปถึงปลายทางซึ่งแพคเก็ต (Packets) ของข้อมูลอาจไปถึงในลักษณะที่ผิดลำดับ, ซ้ำกัน หรือไม่ไปถึงเลย โดยความน่าเชื่อถือของการส่งจะถูกควบคุมในโปรโตคอลในเลเยอร์ต่าง ๆ ไป การหาเส้นทางของข้อมูลจะทำในระดับของโปรโตคอล IP นี้ โดยพิจารณาแต่ละแพคเก็ตแยกออกจากกัน และยังมีหน้าที่ในการจัดเรียงข้อมูลใหม่ที่ปลายทางอีกด้วย

2.2.3.2 โปรโตคอล ARP (Address Resolution Protocol)

โปรโตคอลนี้ทำหน้าที่จับคู่ระหว่างหมายเลข IP แอดเดรสเข้ากับหมายเลขแอดเดรสทางฮาร์ดแวร์ โดยโปรโตคอลนี้ทำการส่งข้อความไปทั่วเครือข่ายท้องถิ่น ซึ่งข้อความนี้เป็นลักษณะข้อความที่ตรวจสอบว่ามีคอมพิวเตอร์ที่มีหมายเลข IP แอดเดรสตรงกับที่ต้องการหาหรือไม่ หากคอมพิวเตอร์ที่มีหมายเลข IP แอดเดรสตรงกันนั้นได้รับข้อความนี้ก็จะตอบกลับ และเป็นที่น่าสังเกตว่าโปรโตคอลนี้ทำงานได้กับระบบเครือข่ายท้องถิ่นเท่านั้น เพราะว่าโครงสร้างหมายเลขทางฮาร์ดแวร์ของเครื่องคอมพิวเตอร์จะขึ้นอยู่กับชนิดของระบบเครือข่ายด้วย

2.2.3.3 โปรโตคอล ICMP (Internet Control Message Protocol)

เป็นโปรโตคอลที่จัดการเกี่ยวกับข่าวสารความผิดพลาดและการควบคุมเกตเวย์ และเครื่องคอมพิวเตอร์ในระบบเครือข่าย

2.2.3.4 โปรโตคอล RARP (Reverse Address Resolution Protocol)

เป็นโปรโตคอลที่ทำหน้าที่จับคู่ระหว่างหมายเลขของฮาร์ดแวร์กับหมายเลข IP แอดเดรส หรือทำงานกลับกันกับโปรโตคอล ARP

2.2.3.5 โปรโตคอล UDP (User Datagram Protocol)

โปรโตคอลนี้เป็นโปรโตคอลที่อยู่ในระดับทรานสปอร์ตเลเยอร์ และมีความสำคัญเพราะว่าเป็นโปรโตคอลที่ผู้พัฒนาโปรแกรมสามารถใช้ได้โดยตรง โปรโตคอลนี้เป็นโปรโตคอลแบบคอนเนคชั่นเลสมีความน่าเชื่อถือต่ำ ไม่รับรองว่าข้อมูลที่ส่งไปจะไปถึงปลายทางหรือไม่ และอาจซ้ำซ้อน หรือผิดลำดับได้ แต่ข้อดีของโปรโตคอลนี้ คือ ค่าความสิ้นเปลือง (overhead) ที่ต่ำ

2.2.3.6 โปรโตคอล TCP (Transmission Control Protocol)

โปรโตคอลนี้อยู่ในระดับทรานสปอร์ตเลเยอร์ เหมือนกับโปรโตคอล UDP แต่มีลักษณะที่ตรงข้ามกันคือ เป็นโปรโตคอลแบบคอนเนคชั่นออเรียนเต็ด โดยจะมีความน่าเชื่อถือในการรับส่งข้อมูล และลำดับของข้อมูลจะมีลำดับเหมือนกับต้นทาง และเนื้อข้อมูลไม่ผิดพลาด จึงทำให้เกิดความสิ้นเปลืองในการเชื่อมต่อของการส่งข้อมูลมากกว่าโปรโตคอล UDP

2.2.4 หมายเลขพอร์ต

เนื่องจากในเวลาใด ๆ สามารถมีโพรเซสของผู้ใช้สามารถใช้ UDP หรือ TCP ได้พร้อมกันดังนั้นจึงต้องมีวิธีแยกแยะว่าข้อมูลเป็นของผู้ใช้คนใด ซึ่งวิธีที่ TCP และ UDP ใช้คือการใช้หมายเลขพอร์ต (port number)

เมื่อโพรเซสไคลเอนต์ (client process) ต้องการที่จะติดต่อกับเซิร์ฟเวอร์ ไคลเอนต์จะต้องเจาะจงเซิร์ฟเวอร์ที่ต้องการติดต่อแต่ลำพังรู้แอดเดรสอินเตอร์เน็ต 32 บิตเพียงอย่างเดียวนั้นไม่เพียงพอ เพราะว่าสามารถติดต่อกับโฮสต์ได้เพียงอย่างเดียวแต่ไม่สามารถเจาะจงโพรเซสที่จะทำการติดต่อได้ ดังนั้นเพื่อแก้ปัญหานี้ทั้ง TCP และ UDP ได้มีการกำหนดหมายเลขพอร์ตมาตรฐาน (well-known ports) ซึ่งเป็นที่รู้จักกัน เช่น ทุก ๆ ระบบ TCP/IP ที่มีเซิร์ฟเวอร์ FTP (File Transfer Protocol) จะมีหมายเลขพอร์ตเป็น 21 เป็นต้น

เมื่อ TCP หรือ UDP กำหนดหมายเลขพอร์ตที่ไม่ซ้ำกันให้โพรเซสของผู้ใช้ เราเรียกหมายเลขพอร์ตนี้ว่าหมายเลขพอร์ตชั่วคราว (ephemeral port numbers) เมื่อไคลเอนต์เลิกใช้หมายเลขพอร์ตนี้แล้ว สามารถกำหนดหมายเลขพอร์ตนี้ให้ไคลเอนต์อื่นได้ โพรเซสที่ได้รับหมายเลขพอร์ตชั่วคราวนี้จะไม่สนใจว่ามีค่าเท่าไร แต่เป็นหน้าที่ของอีกโพรเซสหนึ่งที่ต่อกันที่ต้องสนใจ เพราะต้องส่งข้อมูลกลับมาที่พอร์ตนี้ ใน TCP และ UDP นั้นหมายเลขพอร์ตตั้งแต่ 1-1023 เป็นพอร์ตที่สงวนไว้สำหรับหมายเลขพอร์ตมาตรฐาน

2.3 WinSock (Window Sockets Application Programming Interface)

ในหัวข้อนี้จะได้กล่าวถึง WinSock ที่เป็นไลบรารี (library) ของฟังก์ชันที่ทำหน้าที่เกี่ยวกับการสื่อสาร โดยติดต่อกับซอคเก็ต WinSock ที่มีส่วนขยายมาจาก Berkley socket โดยได้เพิ่มส่วนที่เป็นแมสเสจไดรเวน (Message-driven) เพื่อให้ทำงานได้ดีในสภาวะแวดล้อมของวินโดวส์ และสามารถจัดการระบบเครือข่ายแบบ TCP/IP ได้ ก่อนที่จะกล่าวถึง WinSock จะได้กล่าวถึง Berkley socket เพื่อเป็นพื้นฐานความเข้าใจ WinSock ในอันดับต่อไป

2.3.1 Berkeley Sockets

ซอคเก็ต (Socket) เป็นแอพพลิเคชั่นโปรแกรมอินเตอร์เฟส (Application Program Interface) ในที่นี้จะขอเรียกโดยย่อว่า API โดยที่ API นี้จะเป็นส่วนที่ติดต่อระหว่างทรานสปอร์ตเลเยอร์กับโปรแกรมประยุกต์ ทำให้ผู้พัฒนาโปรแกรมสามารถที่จะเรียกใช้บริการต่าง ๆ ของทรานสปอร์ตเลเยอร์ได้โดยการใช้ซอคเก็ตนั้นเอง Berkeley socket เป็นซอคเก็ตที่ใช้กับทรานสปอร์ตเลเยอร์ของโปรโตคอล TCP/IP โดยที่ Berkeley socket มีความเป็นมาพร้อม ๆ กับอินเตอร์เน็ต

2.3.2 เปรียบเทียบ Berkeley socket และ WinSock

ซอคเก็ตทั้ง 2 มีความแตกต่างกัน โดยแสดงเป็นข้อๆ ได้ดังนี้

1. WinSock รุ่น 1.1 สนับสนุนการทำงานกับกลุ่มโดเมน (domain) TCP/IP เท่านั้น ส่วนซอคเก็ตของยูนิกซ์ (UNIX) นั้นสามารถสนับสนุนการทำงานกับกลุ่มโดเมนของโดเมนยูนิกซ์และกลุ่มโดเมน Xerox XNS ด้วย
2. ค่าที่ส่งกลับจากฟังก์ชัน (return value) ใน Berkely Socket กับ WinSock จะต่างกัน เช่น ถ้าเรียกให้ฟังก์ชัน socket () จะส่งค่ากลับเป็น -1 หากการทำงานผิดพลาดในการทำงานบนระบบปฏิบัติการยูนิกซ์แต่ถ้าหากเป็น WinSock ค่าที่ส่งกลับในกรณีนี้คือ INVALID_SOCKET
3. ชื่อของฟังก์ชันที่ทำหน้าที่เดียวกันอาจแตกต่างกัน เช่น ฟังก์ชัน close () เป็นฟังก์ชันในการปิดการติดต่อของซอคเก็ตในระบบปฏิบัติการยูนิกซ์แต่ถ้าหากเป็นชื่อใน WinSock จะใช้ชื่อ closesocket ()
4. ในระบบปฏิบัติการยูนิกซ์การจัดการกับซอคเก็ตจะเหมือนกับแฟ้มข้อมูล (file) ที่อยู่บนดิสก์ แต่ใน WinSock การจัดการกับซอคเก็ตจะต่างกันออกไป
5. WinSock มีฟังก์ชันที่เพิ่มเติมจาก Berkeley Socket เพื่อจะสนับสนุนการทำงานแบบแมสเสจไดรว์เวน ของสถาปัตยกรรมของระบบปฏิบัติการวินโดวส์ซึ่งจะได้กล่าวในหัวข้อต่อไป

2.3.3 ส่วนเพิ่มเติมของ WinSock จาก Berkeley Sockets

ส่วนที่เพิ่มขึ้นมาหลายส่วนของ WinSock เนื่องมาจากเหตุที่ระบบปฏิบัติการวินโดวส์นั้นมีสถาปัตยกรรมที่เรียกว่าแมสเสจไดรว์เวน และเพื่อจะสนับสนุนการทำงานแบบนอนพรีเอมพ์ทีฟ (nonpreemptive) ของระบบปฏิบัติการวินโดวส์อีกด้วย ในส่วนต่อไปจะได้กล่าวถึงสถาปัตยกรรมแมสเสจไดรว์เวนของระบบปฏิบัติการวินโดวส์ก่อน เพื่อเป็นพื้นฐานความเข้าใจในการอธิบายต่อไป

2.3.3.1 สถาปัตยกรรมแมสเสจไดรว์เวนของวินโดวส์

ในที่นี้จะกล่าวโดยสรุปเกี่ยวกับสถาปัตยกรรมแมสเสจไดรว์เวน ดังนั้นทุก ๆ โปรแกรมที่ทำงานภายใต้ระบบปฏิบัติการวินโดวส์นี้จะต้องมีส่วนประกอบหลักสำคัญ 2 ส่วน ได้แก่ ลูป (loop) ที่ทำหน้าที่รับแมสเสจ และโปรแกรมย่อยอื่น ๆ ที่เป็นส่วนปฏิบัติงานจริงของโปรแกรม โดยในส่วนที่เป็นลูปคอยรับแมสเสจนั้นจะรับแมสเสจจากแมสเสจคิว (Message Queue) เมื่อโปรแกรมได้รับแมสเสจแล้วจะทำการปฏิบัติงานตามโปรแกรมย่อยที่สัมพันธ์กับแมสเสจที่ได้รับนั้น ซึ่งการทำงานแบบนี้จึงเป็นที่มาของแมสเสจไดรว์เวน หรืออีเวนต์ไดรว์เวน (event-driven) เพราะไม่มีส่วนของโปรแกรมใด ๆ ทำงานได้จนกว่าจะมีเหตุการณ์ หรือแมสเสจที่เกี่ยวข้องเกิดขึ้น โดยทั่วไปตัวอย่างของแมสเสจคือ การกดคีย์บอร์ด หรือการเลื่อนเมาส์ และมาจากภายในตัวระบบปฏิบัติการเอง ส่วนโปรแกรมย่อยที่เรียกใช้ในการปฏิบัติการจริงของโปรแกรมที่จะปฏิบัติตามแมสเสจที่ได้รับนั้นจะอยู่ในรูปคลาสของโปรแกรมโดยมีที่มา 2 แบบ คือมาจากการโปรแกรมของผู้เขียนโปรแกรมนั้น และบางส่วนจะเป็นคลาสที่ถูกกำหนดไว้ล่วงหน้าโดยระบบปฏิบัติการวินโดวส์

เมื่อทำงานตามโปรแกรมย่อยแล้ว โปรแกรมจะเข้าสู่การทำงานในลูปรอรับแมสเสจอีกครั้งหนึ่งเพื่อคอยแมสเสจที่จะเข้ามาในแมสเสจคิวต่อไป แต่ในระหว่างที่โปรแกรมกำลังทำงานอยู่ในโปรแกรมย่อยระบบปฏิบัติการวินโดวส์จะไม่สามารถควบคุมการทำงานของโปรแกรมนั้น ๆ ได้ เรียกลักษณะงานแบบนี้ว่านอนพรีเอมพ์ตีฟ

2.3.3.2 ฟังก์ชันการทำงานแบบอะซิงโครนัสของ WinSock

ในการออกแบบ WinSock นั้นครั้งแรกออกแบบมาเพื่อทำงานกับสภาวะแบบนอนพรีเอมพ์ตีฟของระบบปฏิบัติการวินโดวส์ ด้วยเหตุนี้จึงต้องเพิ่มฟังก์ชันการทำงานจาก Berkeley Sockey คือ

การทำงานแบบบล็อกกิ้ง (Blocking) และนอนบล็อกกิ้ง (Nonblocking)

ฟังก์ชันในการทำงานของ Berkeley Sockets หลายฟังก์ชันเป็นฟังก์ชันที่ไม่ทราบเวลาในการส่งค่ากลับที่แน่นอน เรียกลักษณะการทำงานแบบนี้ว่าการทำงานแบบบล็อก ซึ่งในสภาวะการทำงานของระบบปฏิบัติการยูนิกซ์การทำงานของฟังก์ชันแบบนี้จะไม่เป็นปัญหาที่ร้ายแรง เพราะระบบปฏิบัติการจะทำการจัดการในส่วนที่จะพรีเอมพ์ (preempt) โปรแกรมที่บล็อก และให้โปรแกรมอื่นทำงานแทน แต่ในทางกลับกันบนระบบปฏิบัติการวินโดวส์ ระบบปฏิบัติการไม่สามารถจะทำการพรีเอมพ์โปรแกรมได้ จึงต้องรอให้โปรแกรมนั้นคืนการทำงานให้แก่ระบบปฏิบัติการเท่านั้น WinSock จึงจำเป็นต้องฟังก์ชันการทำงานแบบอะซิงโครนัส เพื่อที่จะแก้ปัญหาดังกล่าว เพราะว่าการทำงานบนระบบเครือข่ายไม่สามารถกำหนดเวลาได้แน่นอน ฟังก์ชันที่ทำงานแบบอะซิงโครนัสคือ ฟังก์ชันที่เมื่อเรียกทำงานแล้วจะส่งค่ากลับทันที แต่อาจจะยังไม่มีการทำงานเกิดขึ้นในทันที และเมื่อทำงานเสร็จแล้วจะมีแมสเสจส่งกลับมาหาโปรแกรมที่เรียกใช้ฟังก์ชันนั้น เพื่อบอกถึงการทำงานที่เสร็จสิ้น และฟังก์ชันที่เป็นฟังก์ชันแบบอะซิงโครนัสจะมีคำนำหน้าว่า WSAAsync นำหน้าชื่อฟังก์ชัน ซึ่งจะเห็นได้ว่าการทำงานแบบอะซิงโครนัสถูกออกแบบมาเพื่อให้เข้ากับการทำงานบนระบบปฏิบัติการวินโดวส์เป็นอย่างดี

2.3.4 ไฟล์ (File) ที่ใช้ในการพัฒนาโปรแกรมที่ใช้ WinSock. API

ในการพัฒนาโปรแกรมที่ใช้ WinSock API (และให้คอมไพล์เลอร์ภาษา C หรือ C++) มีไฟล์ที่จำเป็นดังนี้

- WINSOCK.H เป็นไฟล์ที่ประกาศซึ่งฟังก์ชัน และโครงสร้างข้อมูลที่จำเป็นของ WinSock API
- WINSOCK.LIB หรือ WSOCK32.LIB เป็นไฟล์ที่ใช้ในการเชื่อม (Link) กับโปรแกรมเพื่อให้ได้โปรแกรมซึ่งทำงานได้
- WINSOCK.DLL ไดนามิคลิงค์ไลบรารี (Dynamic Linked Library) ที่เป็นที่เก็บฟังก์ชันของ WinSock

2.4 หลักการพัฒนาโปรแกรมด้วย Microsoft Visual C++

Microsoft Visual C++ (ต่อไปจะขอเรียกสั้น ๆ ว่า Visaul C++) เป็นโปรแกรมเครื่องมือที่ช่วยในการพัฒนาโปรแกรมบนวินโดวส์โดยใช้ภาษา C++ เป็นภาษาในการพัฒนา ด้วยคุณลักษณะของ C++ นี้เองทำให้การพัฒนาโปรแกรมบนวินโดวส์ทำได้ง่ายกว่าการใช้ภาษา C ธรรมดา เนื่องจากภายใน Visual C++ จะมีออพเจคสำเร็จมาให้ใช้งานได้ทันทีเพียงแค่เรียกออพเจคนั้นมาใช้เท่านั้น นอกจากนี้ Visual C++ ยังช่วยให้การพัฒนาโปรแกรมง่ายขึ้นด้วยการติดต่อกับผู้ใช้แบบวิชวล (visual) เราสามารถฝัง (map) ฟังก์ชันเข้ากับแมสเสจของวินโดวส์ได้อย่างง่ายดาย แม้การติดต่อจะไม่สะดวกสบายเท่ากับเครื่องมือแบบวิชวลตัวอื่น ๆ เช่น Microsoft Visual Basic หรือ Delphi ก็ตาม แต่ก็ช่วยลดความยุ่งยากในการจัดการส่วนต่าง ๆ ที่กล่าวมาแล้วข้างต้นไปได้มาก เครื่องมือที่สำคัญของ Visual C++ ได้แก่ MFC (Microsoft Foundation Class) ได้

ถูกออกแบบมาสำหรับใช้งานกับ Visual C++ โดยเฉพาะ ซึ่ง MFC นี้จะประกอบด้วยคลาส (class) ต่าง ๆ ซึ่งช่วยให้การเขียนโปรแกรมบนวินโดวส์ง่ายขึ้น เช่นออพเจคจัดการลิสต์บ๊อกซ์, ไดอะลอกบ๊อกซ์ เป็นต้น โดยจะทำงานร่วมกับ

- AppWizard เพื่อใช้ในการสร้างเฟรมเวิร์ค (framework) ของโปรแกรม
- ClassWizard จะเป็นเครื่องมือช่วยในการจัดการเกี่ยวกับคลาสต่าง ๆ รวมถึงการเชื่อมแมสเสจ (message) ต่าง ๆ เข้ากับฟังก์ชัน (function) ออพเจคใด ๆ
- AppStudio เป็นเครื่องมือที่ใช้ในการช่วยออกแบบส่วนติดต่อกับผู้ใช้ (user interface)

เมื่อรวม AppWizard, AppStudio และ ClassWizard เข้าไว้ด้วยกันจะเรียกรวมว่า Visual Workbench ซึ่งหมายถึงเครื่องมือต่าง ๆ ที่ช่วยในการพัฒนาโปรแกรมบนวินโดวส์

โดยทั่วไปแอพพลิเคชั่น (application) ที่สร้างจาก Visual C++ จะประกอบไปด้วยออพเจคหลัก 2 ตัวคือ

1. ออพเจคเอกสาร (document object) มีหน้าที่จัดการเกี่ยวกับข้อมูลที่แอพพลิคชันนั้นใช้ รวมถึงการอ่าน ,บันทึกไฟล์ และอื่น ๆ
2. ออพเจควิว (view object) มีหน้าที่จัดการเกี่ยวกับส่วนแสดงผลบนหน้าจอ

โดยการแยกโปรแกรมออกเป็นออพเจคหลัก 2 ส่วนจะช่วยให้เราแยกโปรแกรมที่ซับซ้อนออกเป็นส่วนซึ่งจะมีขนาดเล็กลง และทำให้ง่ายต่อการพัฒนา และการแยกออกเป็นออพเจคจัดการข้อมูล และจัดการส่วนแสดงผลทำให้เราสามารถพัฒนาส่วนแสดงผล และส่วนจัดการข้อมูลไปพร้อมกันได้อย่างเป็นอิสระ

2.5 กติกาหมากรุกไทยเบื้องต้น

2.5.1 กระดานของหมากรุกไทย

กระดานหมากรุกไทยส่วนใหญ่มักจะทำจากไม้หนาพอประมาณ โดยมีขนาดความกว้าง-ยาวดังนี้

- ความยาว 19 นิ้ว
- กว้าง 16 นิ้ว
- โดยส่วนที่เป็นตารางวัดจากปลายสุดของด้านยาวเข้ามาข้างละ 15 นิ้ว จะได้สี่เหลี่ยมจัตุรัสที่มีขนาดเท่าความยาวของด้านกว้างคือ 16 x 16 นิ้ว
- แบ่งรูปสี่เหลี่ยมจัตุรัสนั้นออกเป็น 64 ช่อง โดยแต่ละช่องตารางที่แบ่งได้จะมีขนาด 2 x 2 นิ้ว

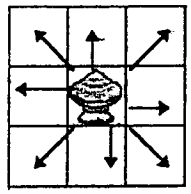
2.5.2 ตัวหมากรุกไทย

แต่ละฝ่ายจะมีตัวหมากฝ่ายละ 16 ตัว โดยประกอบไปด้วย

1. ขุน 1 ตัว
2. โคน 2 ตัว
3. เม็ด 1 ตัว
4. ม้า 2 ตัว
5. เรือ 2 ลำ
6. เบี้ย 8 ตัว

2.5.2.1 ขุน

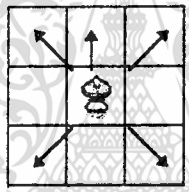
เปรียบได้กับแม่ทัพหรือขุนพล ขุนพลมีรูปลักษณะของหมากจะคล้ายเป็นรูปเจดีย์ใหญ่กว่าโคน และเม็ด ซึ่งมีลักษณะคล้ายกัน วิธีการเดิน ขุนจะมีสิทธิ์เดินได้รอบ ๆ ตัวทั้ง 8 ทิศ แต่เดินได้เพียง 1 ตาเท่านั้น ดังรูป



ภาพที่ 2.12 ตาเดินของขุน

2.5.2.2 โคน

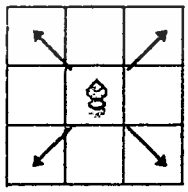
รูปลักษณะของโคนเหมือนขุน แต่มีขนาดเล็กกว่า แต่ยังมีขนาดใหญ่กว่าเม็ด วิธีเดินการเดิน โคนมีการเดินคล้ายกับขุน แต่สามารถเดินได้ 5 ทิศรอบ ๆ ตัวเท่านั้นคือ เดินหน้าตรง, เดินหน้าเฉียงซ้าย และขวา, เดินเฉียงไปด้านหลังซ้าย และขวา และสามารถเดินได้ทีละตาเท่านั้นดังรูป



ภาพที่ 2.13 ตาเดินของโคน

2.5.2.3 เม็ด

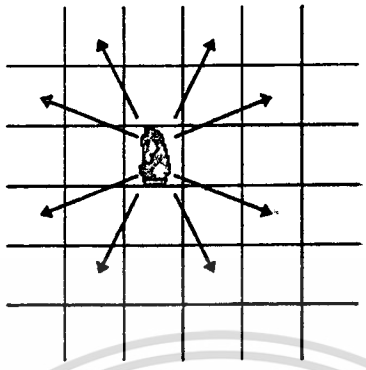
ตัวหมากมีลักษณะการเดินคล้ายโคน แต่เดินได้น้อยกว่าคือ เดินได้ 4 ทิศรอบตัว ได้แก่ ในแนวเฉียงทั้ง 4 ทิศ และสามารถเดินได้ทีละช่องเช่นกัน



ภาพที่ 2.14 ตาเดินของเม็ด

2.5.2.4 ม้า

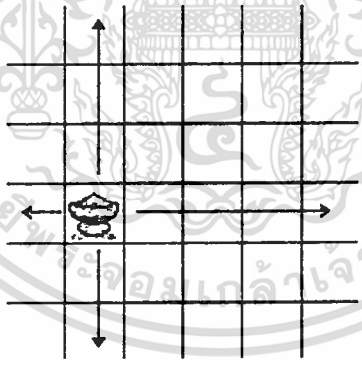
รูปลักษณะเป็นรูปม้าตั้งแต่คอถึงหัว การเดิน ม้ามีการเดินที่ค่อนข้างพิเศษคือ เดินเฉียงข้ามไป 2 ตา (ให้ดูรูปประกอบ) และสามารถเดินข้ามหมากอื่น ๆ ได้ แม้จะมีหมากอื่น ๆ ขวางอยู่



ภาพที่ 2.15 ตาเดินของม้า

2.5.2.5 เรือ

ลักษณะของตัวหมากเหมือนกับหัวเสา หรือยอดเสาธง หรือคล้ายขุนที่ตัดเอาท่อนล่างออก การเดิน เรือเดินได้ในทิศตรงทั้ง 4 ทิศ คือหน้า, หลัง, ซ้าย และขวา โดยไม่จำกัดจำนวนช่องที่เดิน แต่เรือไม่สามารถเดินข้ามหากอื่น ๆ ได้ถ้ามีหมากอื่น ๆ ขวางทางอยู่

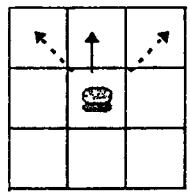


ภาพที่ 2.16 ตาเดินของเรือ

2.5.2.6 เบี้ย

ลักษณะของตัวหมากเหมือนดังรูป คือมีลักษณะเหมือนฝาขวด การเดิน เบี้ยเดินได้ 1 ทิศเท่านั้นคือ ตรงไปข้างหน้าทีละช่อง แต่สามารถกินหมากอื่นได้ในแนวเฉียงไปข้างหน้า ทั้งข้างซ้าย และขวาเท่านั้น

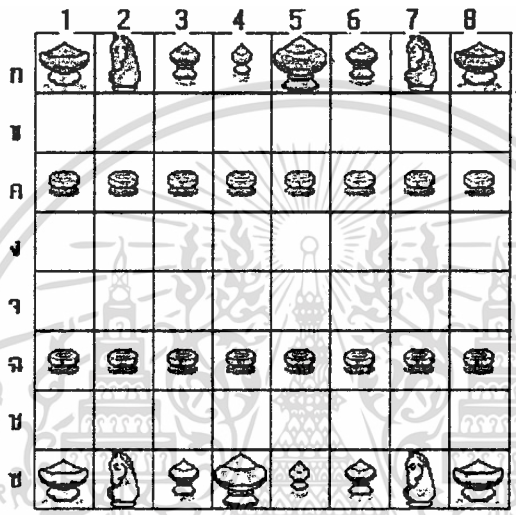
หมายเหตุ เมื่อเบี้ยเดินไปถึงช่วงที่เป็นช่วงเริ่มต้นของเบี้ยอีกฝ่าย เบี้ยตัวนั้นจะเปลี่ยนสภาพการเดินเป็นเบี้ยหงาย และมีสิทธิ์เดินได้เหมือนเม็ด



ภาพที่ 2.17 ตาเดินของเบี้ย

2.5.3 การตั้งหมาก

เมื่อเริ่มเล่นทั้ง 2 ฝ่ายจะต้องตั้งตัวหมากตามตำแหน่งดังภาพ



ภาพที่ 2.18 การตั้งหมาก

2.5.4 การแพ้, ชนะ และการนับศักดินา

การแพ้ชนะเกิดขึ้นในกรณีที่ฝ่ายหนึ่งทำให้ขุนของอีกฝ่ายจนได้ก็จะเป็นฝ่ายชนะ (โดยปกติในการเล่นหมากรุกจะไม่มีการกินขุนของอีกฝ่าย) แต่ถ้าในกรณีที่ขุนของอีกฝ่ายอับทั้ง 2 ฝ่ายจะเสมอกัน

แต่เพื่อให้การเล่นในแต่ละกระดานจบลงภายในเวลาอันสมควรจึงมีการตั้งกติกาการนับศักดินาของกระดาน และศักดินาของกระดานดังนี้

การนับศักดินามีด้วยกัน 2 วิธีคือ การนับศักดินาของหมาก และศักดินาของกระดาน โดยการนับทั้ง 2 นี้จะเริ่มนับกันได้ก็ต่อเมื่อเบี้ย (เฉพาะที่ยังไม่หงายเท่านั้น) ของทั้ง 2 ฝ่ายหมดไปจากกระดานแล้วเท่านั้น วิธีการนับแต่ละแบบเป็นดังนี้

2.5.4.1 การนับศักดินาของกระดาน

ผู้เล่นทั้ง 2 ฝ่ายจะต้องมีฝ่ายใดฝ่ายหนึ่งเป็นผู้ไล่ และอีกฝ่ายเป็นผู้หนี โดยผู้หนีจะเป็นฝ่าที่มีหมากน้อยกว่า การนับศักดินาของกระดานถือเอาจำนวนตารางคือ 64 ช่องมาเป็นจำนวนนับ โดยเริ่มนับตั้งแต่ 1 ไปจนครบ 64 ตาหากยังไม่มีการแพ้-ชนะจะถือว่าขุนจน

2.5.4.2 การนับศักดินาหมาก

ในกรณีนี้นอกจากจะต้องไม่มีเบี้ย (คว่ำ) แล้วหมากของฝ่ายหนีจะต้องมีขุนเพียงตัวเดียวจึงเริ่มนับได้ และจำนวนการนับตาเดินจะพิจารณาตามตัวหมากของฝ่ายไล่โดยจะกล่าวต่อไป และจะเริ่มนับจากจำนวนหมากบนกระดาน เช่น หากบนกระดานมีทั้งหมด 4 ตัวก็จะนับเริ่มต้นที่ 5

การพิจารณาจำนวนตาในการนับ

1. ฝ่ายไล่มี เรือ 2 ลำ จะนับ 8 ครั้ง
2. ฝ่ายไล่มี เรือ 1 ตัว จะนับ 16 ครั้ง
3. ฝ่ายไล่มี ม้า 2 ตัว จะนับ 32 ครั้ง
4. ฝ่ายไล่มี ม้า 1 ตัว จะนับ 64 ครั้ง
5. ฝ่ายไล่มี โคน 2 ตัว จะนับ 22 ครั้ง
6. ฝ่ายไล่มี โคน 1 ตัว จะนับ 44 ครั้ง
7. ฝ่ายไล่มี เม็ด 1 ตัว หรือเหลือเบี้ยหงายจำนวน 3 ตัวขึ้นไปไม่มีเม็ดจะต้องนับ 64 ครั้ง

บทที่ 3
การออกแบบ

3.1 หลักในการออกแบบ

จุดมุ่งหมายของโปรแกรมหมากรุกไทยนี้ต้องการให้ผู้เล่นสามารถเล่นหมากรุกผ่านระบบเครือข่ายได้ ซึ่งโปรโตคอลที่ใช้ได้แก่โปรโตคอล TCP/IP (Transmission Control Protocol/Internet Protocol) โดยมีหลักในการออกแบบดังนี้

1. พัฒนาได้ง่าย
2. แก้ไขได้ง่าย
3. โครงสร้างของโปรแกรมไม่ซับซ้อนเกินไป

เมื่อพิจารณาแล้วพบว่าหลักการโปรแกรมเชิงวัตถุ (object oriented programming) เป็นวิธีที่นำไปสู่เป้าหมายในการออกแบบไว้ได้เป็นอย่างดี เนื่องจากในหลักของการโปรแกรมเชิงวัตถุนี้จะแบ่งส่วนต่าง ๆ ของโปรแกรมออกเป็นส่วนต่าง ๆ เรียกว่าออพเจค (object) ซึ่งแต่ละออพเจคจะมีความเป็นอิสระกัน ทำให้สะดวกในการพัฒนา และแก้ไข เนื่องจากเมื่อแก้ไขออพเจคตัวใดตัวหนึ่งแล้วจะไม่มีผลกระทบกับออพเจคอื่น และโครงสร้างของโปรแกรมก็จะไม่ซับซ้อน สามารถทำความเข้าใจได้ง่ายเนื่องจากได้แยกโปรแกรมออกเป็นส่วน ๆ แล้ว

เมื่อได้หลักในการพัฒนาแล้วสิ่งที่ต้องพิจารณาต่อไปได้แก่ ความสามารถในโปรแกรมหมากรุกไทยจะต้องมีเพื่อให้สามารถทำงานได้ตามต้องการ

1. สามารถตรวจเช็คกติกาการเล่นของหมากรุกไทยได้
2. เนื่องจากเป็นการเล่นระหว่างผู้เล่น 2 คนผ่านระบบเครือข่าย (network) ดังนั้นต้องสามารถติดต่อสื่อสารระหว่างผู้เล่นทั้ง 2 ฝั่งได้
3. ต้องแสดงผลได้ตามที่ต้องการ
4. สามารถทำการแสดงผลออกทางหน้าจอได้
5. สามารถรับคำสั่งในการเดินจากผู้เล่นได้

จากความต้องการข้างต้นนี้สามารถออกแบบโปรแกรมหมากรุกไทยแบ่งออกออพเจคหลักได้เป็น 4 ออพเจค ได้แก่

1. ออพเจคหลักที่ใช้แสดงผล และประมวลผล ได้แก่ CNyView
2. ออพเจคจัดการการติดต่อ ได้แก่ CWinSock
3. ออพเจคจัดการกติกาทั่วไปบนกระดาน ได้แก่ CBoard
4. ออพเจคจัดการกติกาในการสำหรับหมากแต่ละตัว ได้แก่ MetaPiece, CCone, CBear, CKhun, CMar, CRyur, CMed

รายละเอียดของออพเจคต่าง ๆ จะกล่าวถึงต่อไปในหัวข้อ 3.2 โครงสร้าง และความสัมพันธ์ระหว่างออพเจคของโปรแกรมต่อไป

3.2 โครงสร้าง และความสัมพันธ์ระหว่างออพเจคของโปรแกรม

3.2.1 ออพเจค CNyView

ออพเจค CNyView จะเป็นออพเจคหลักของโปรแกรม มีหน้าที่ดังต่อไปนี้

1. เป็นส่วนหลักในการแสดงผล
2. รับคำสั่งจากผู้เล่น
3. เริ่ม และยุติการเล่น
4. เริ่ม และยุติการติดต่อ โดยจะทำการเรียกใช้ใช้ออพเจค CWinSock
5. เช็คกติกาพิเศษนอกเหนือไปจากการตรวจว่าเดินได้ถูกต้องตามกติกาหรือไม่ เช่น ตรวจดูการแพ้ชนะ ตรวจดูตำแหน่งของขุนว่าถูกต้องหรือไม่ เป็นต้น
6. ตรวจกติกาในการเดิน โดยใช้ออพเจคของตัวหมากเอง เช่น CBear, CCone, CKhun เป็นต้น
7. ฟังก์ชันที่ใช้ในการสื่อสารอื่น ๆ

ออพเจค CNyView นั้นเป็นได้รวบรวมออพเจคต่าง ๆ เข้าไปเป็นสมาชิกของออพเจค CNyView เมื่อให้ออพเจค CNyView เรียกใช้ โดยออพเจคต่าง ๆ ดังต่อไปนี้

3.2.2 ออพเจค CWinSock

ออพเจคนี้ทำการรวบรวมฟังก์ชันที่ใช้ในการเริ่ม และยุติการติดต่อของ WinSock API เอาไว้ นอกจากนี้ยังมีออพเจคอีก 2 ตัวที่จำเป็นต้องใช้คือ CStreamSocket และ CDatagramSocket ซึ่งเป็นส่วนจัดการการสื่อสารแบบสตรีม (stream) และดาต้าแกรม (datagram) ตามลำดับ ซึ่งทั้งสองออพเจคนี้จะมีฟังก์ชันไว้สำหรับการอ่าน หรือส่งข้อมูลผ่านระบบเครือข่าย ในโครงงานชิ้นนี้ได้เลือกใช้ออพเจคสตรีมเนื่องจากลักษระการสื่อสารแบบสตรีมนี้จะมีการตรวจสอบความผิดพลาดที่อาจเกิดขึ้นได้ ซึ่งจะช่วยลดภาระในการออกแบบโปรโตคอลของโปรแกรมนี้ลงได้ส่วนหนึ่ง สำหรับรายละเอียดเพิ่มเติมจะอยู่ที่บทที่ 2 หัวข้อความรู้เบื้องต้นเกี่ยวกับระบบเครือข่าย TCP/IP

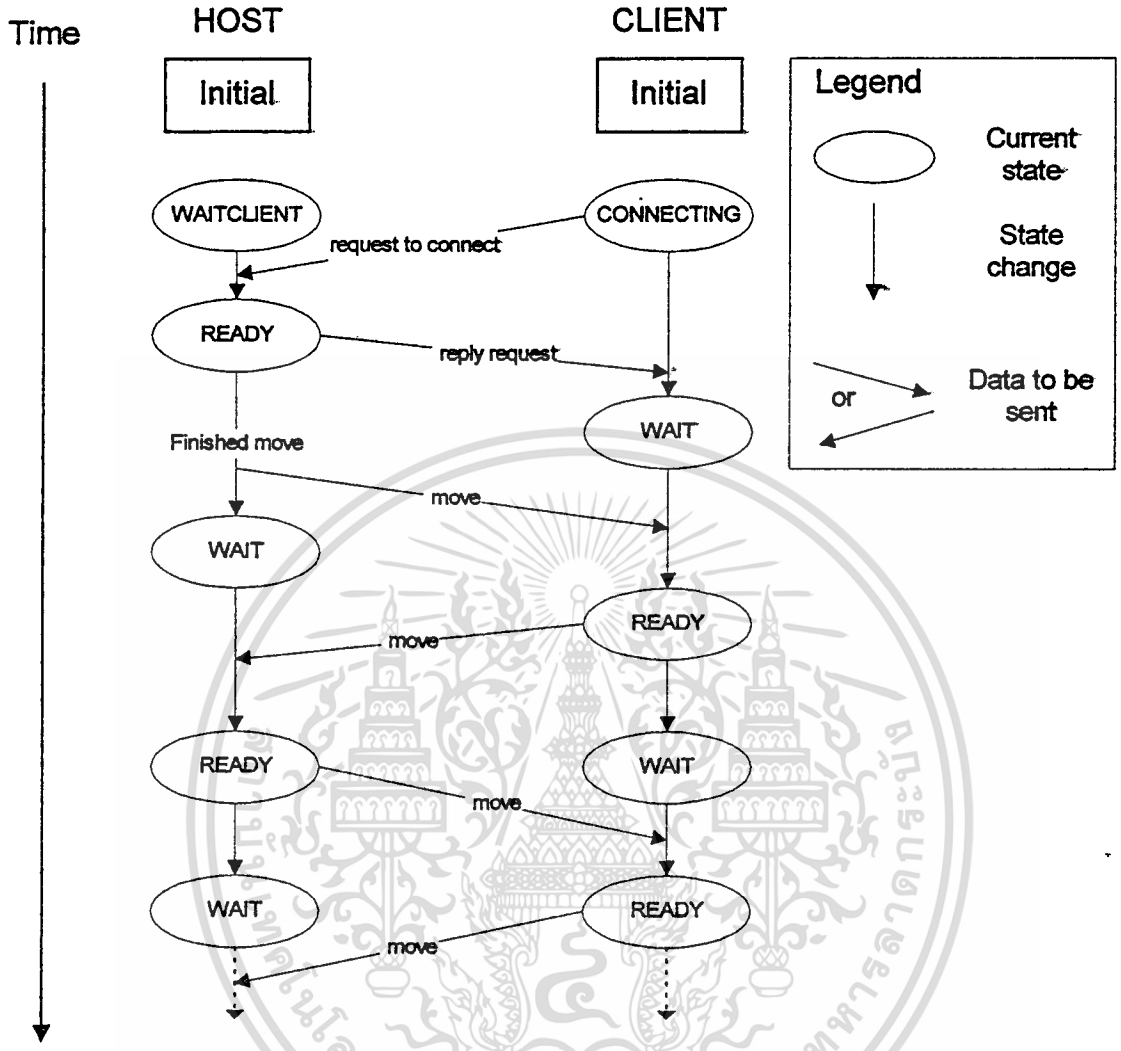
3.2.3 ออพเจค CBoard

ออพเจคนี้จะเก็บข้อมูลของหมากทุกตัวที่มีอยู่บนกระดาน เช่น ตำแหน่ง, ตาผู้เล่นคนปัจจุบัน, สีของผู้เล่น เป็นต้น นอกจากนี้ยังมีฟังก์ชันที่ใช้ตรวจสอบกติกาบางส่วนด้วย

3.2.4 ออพเจคของตัวหมาก (CBear, CCone, CKhun, CMar, CMed, CRyur)

จะเก็บข้อมูลของตัวหมากเอง เช่น สี, ตำแหน่งบนกระดาน, ชนิด และรูปของตัวหมากเอง นอกจากนี้ยังรวมไปถึงฟังก์ชันที่ใช้ในการตรวจสอบการเดินของตัวหมากนี้ว่าเป็นไปตามกติกาหรือไม่ด้วย เช่นหมากขุนจะเดินได้ 8 ทิศรอบตัว แต่ได้ทีละหนึ่งช่องเท่านั้น เป็นต้น

3.3 การออกแบบโปรโตคอล



ภาพที่ 3.1 โปรโตคอลที่ใช้ในการติดต่อ

การเริ่มเล่นจะทำได้โดยต้องมีฝ่ายใดฝ่ายหนึ่งทำหน้าที่เป็นโฮสต์ (host) คอยการติดต่อจากผู้เล่นอีกฝ่ายหนึ่ง และผู้เล่นอีกฝ่ายหนึ่งจะเป็นไคลเอนต์ (client) ทำหน้าที่ติดต่อไปยังโฮสต์ โดยในการเริ่มต้นติดต่อจะต้องมีการตรวจสอบชนิดของข้อมูลที่ส่งไปมาด้วยว่าเป็นชนิดของข้อมูลที่กำหนดไว้หรือไม่ เนื่องจากอาจมีโปรแกรมที่มีการทำงานต่างกันร้องขอการติดต่อมา ถ้าหากทั้งสองฝ่ายทำการเชื่อมต่อโดยไม่มีการตรวจสอบชนิดของข้อมูลก่อน จะทำให้การสื่อสารไม่ตรงกันได้ จึงจำเป็นต้องสร้างโปรโตคอลสำหรับโปรแกรมหมากรุกนี้ขึ้นมา

3.3.1 ลำดับขั้นตอนในการติดต่อของฝ่ายโฮสต์

1. ทำการเริ่มต้น (initial) การทำงานของโฮสต์
2. รอการเชื่อมต่อกับไคลเอนต์
3. เมื่อเชื่อมต่อสำเร็จแล้วเข้าสู่สถานะ (state) WAITCLIENT คอยไคลเอนต์ส่งข้อมูลชนิด CONN
4. เมื่อได้รับข้อมูลที่ต้องการจากไคลเอนต์แล้วเริ่มต้นการเล่นหมากรุก

5. เปลี่ยนไปสู่สถานะ READY
6. เมื่อเล่นเสร็จแล้วส่งข้อมูลการเล่นไปให้ไคลเอนต์แล้วเข้าสู่สถานะ WAIT
7. เมื่อได้รับข้อมูลการเล่นจากไคลเอนต์ ให้ย้อนกลับไปทำตั้งแต่ขั้นตอนที่ 5 อีกครั้ง

3.3.2 ลำดับขั้นตอนในการติดต่อของฝ่ายไคลเอนต์

1. ทำการเริ่มต้นการทำงานของไคลเอนต์
2. เมื่อเชื่อมต่อสำเร็จแล้ว เข้าสู่สภานะ CONNECTING
3. ส่งข้อมูล CONN ไปยังโฮสต์ แล้วรอรับข้อมูล CONN จากโฮสต์
4. เข้าสู่สถานะ WAIT คอยฝั่งโฮสต์ส่งข้อมูลการเดินของโฮสต์
5. เข้าสู่สถานะ PLAY
6. เมื่อเล่นเสร็จแล้วให้ส่งข้อมูลการเล่นของตัวเองไปให้โฮสต์ แล้วเข้าสู่สถานะ WAIT
7. เมื่อได้รับข้อมูลการเล่นจากฝั่งโฮสต์ ให้ย้อนกลับไปทำตั้งแต่ขั้นตอนที่ 4 อีกครั้ง

ชนิดของข้อมูลการเล่นที่ส่งระหว่างทั้ง 2 ฝั่งนั้นได้แก่ ตำแหน่งเดิมของหมากที่เดิน ตำแหน่งใหม่ของหมากตัวนี้ และเวลาที่ใช้ไปของฝ่ายส่ง กระบวนการประมวลผลกติกาทั้งหมดจะแยกทำที่แต่ละฝ่าย โดยฝ่ายที่ทำการเล่นจะประมวลผลกติกาว่าถูกต้องตามกติกาหรือไม่หากถูกต้องจึงสามารถส่งข้อมูลการเล่นไปให้อีกฝ่ายหนึ่งได้ได้ ส่วนฝ่ายที่คอยนั้นเมื่อได้รับข้อมูลการเล่นของฝ่ายตรงข้ามก็จะทำการเดินตามข้อมูลที่ได้รับมา โดยจะทำการตรวจสอบเพียงผลการเล่นว่ารู้แพ้-ชนะหรือยังเท่านั้นเนื่องจากความถูกต้องในการเดินนั้นได้ถูกตรวจสอบมาจากฝ่ายที่ส่งมาแล้ว
เนื่องจากการตรวจสอบผลแพ้-ชนะสามารถทำได้โดยอาศัยข้อมูลการเดินที่ได้รับมา ดังนั้นการยุติการเล่นเนื่องจากผลแพ้-ชนะเกิดขึ้นนั้นไม่จำเป็นต้องทำการส่งการติดต่อเพื่อยุติการเล่นซึ่งจะช่วยลดปัญหาความผิดพลาดที่อาจเกิดขึ้นในการส่งข้อมูลผ่านระบบเครือข่ายได้ ฝ่ายที่เล่นจะทราบผลทันทีเมื่อทำการเดินเสร็จ และเมื่อฝ่ายที่คอยได้รับข้อมูลการเล่นของอีกฝ่ายหนึ่งก็จะทราบได้ทันทีว่ารู้ผลแพ้-ชนะแล้ว
ในกรณีที่การติดต่อถูกยกเลิกไม่ว่าจะเป็นด้วยกรณีใด ๆ ก็ตามอีกฝ่ายหนึ่งจะรู้ได้ทันทีโดยโปรโตคอลในชั้นของโปรแกรมไม่จำเป็นต้องมีการตรวจสอบการติดต่อด้วยตัวเอง เนื่องจากโปรโตคอลในระดับชั้นทรานสปอร์ต (Transport) เป็นชนิดสตรีมซึ่งสามารถตรวจจับความผิดพลาดเมื่อการติดต่อถูกยกเลิกได้เองจึงทำให้โปรโตคอลในชั้นของโปรแกรมมีความซับซ้อนน้อยลง โดยจะแบ่งภาระบางอย่างไปให้โปรโตคอลในชั้นทรานสปอร์ต (คือโปรโตคอลชนิดสตรีม) จัดการแทน

3.4 การออกแบบส่วนติดต่อกับผู้ใช้ (User Interface)

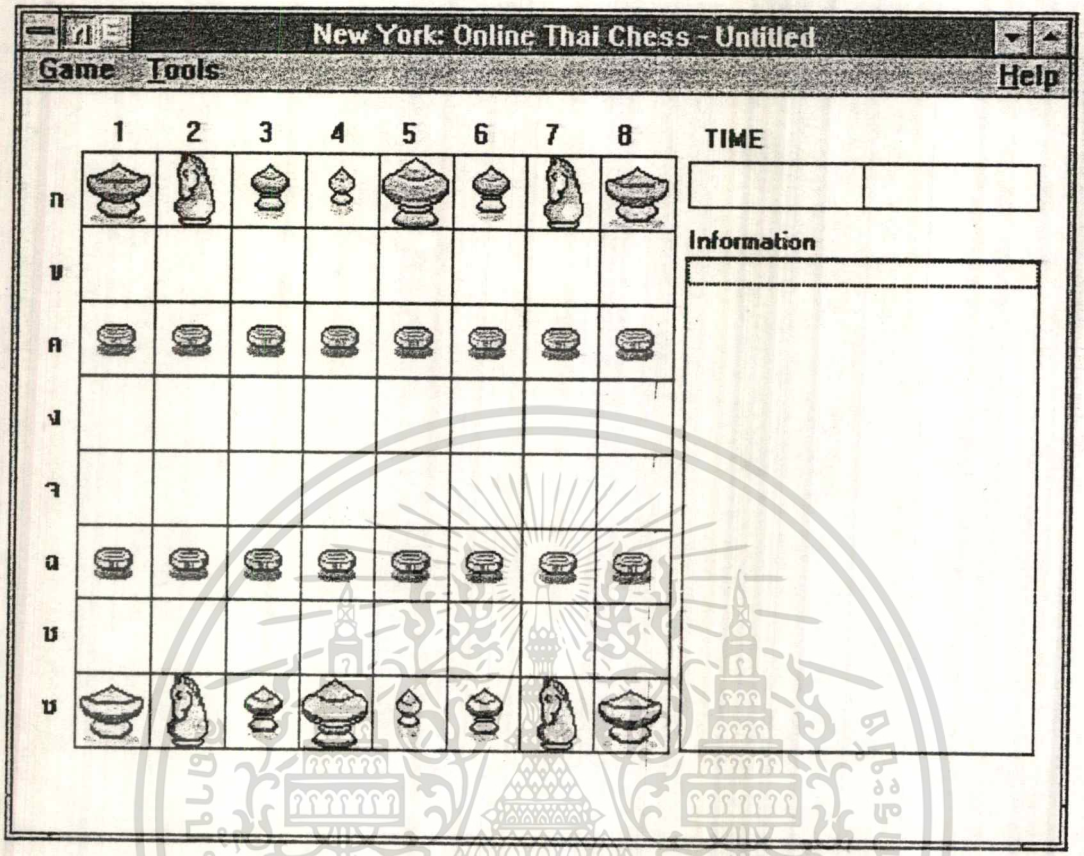
ส่วนติดต่อกับผู้ใช้แบ่งออกเป็น 5 ส่วนคือ

3.4.1 ส่วนติดต่อหลัก

เมื่อเรียกโปรแกรมหมากรุกขึ้นมาครั้งแรกจะแสดงหน้าต่าง (window) หลักดังภาพที่ 2 ภายในหน้าต่างหลักนี้จะประกอบไปด้วยส่วนหลัก 3 ส่วนดังนี้

1. ส่วนกระดาน

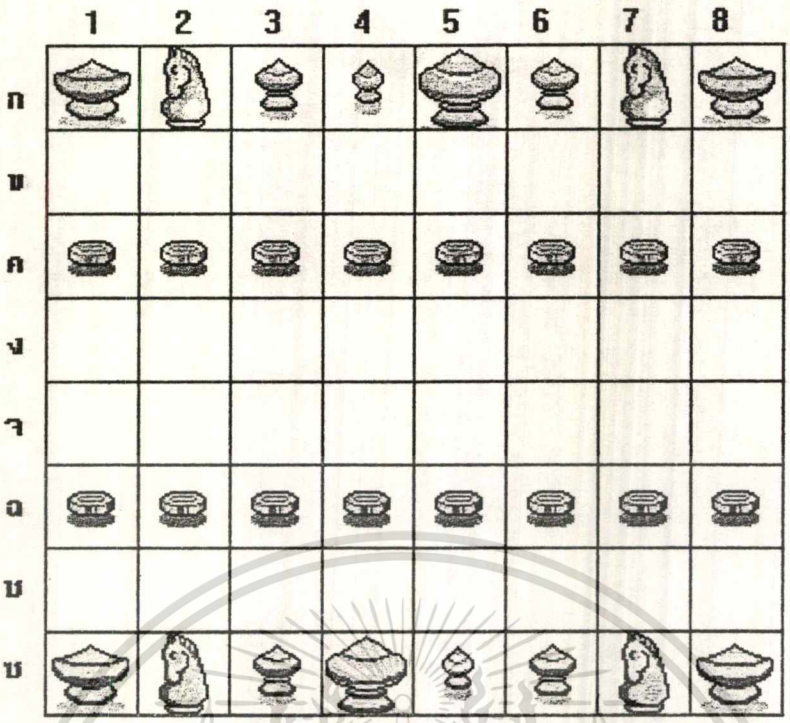
2. ส่วนแสดงเวลา
3. ส่วนแสดงข่าวสาร



ภาพที่ 3.2 หน้าต่างหลักของโปรแกรม

3.4.1.1 ส่วนกระดาน

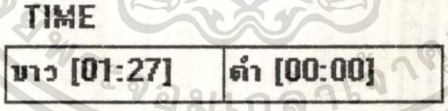
จะเป็นส่วนหลักในการติดต่อรับคำสั่งการเดินหมากจากผู้เล่น วิธีเล่นจะให้ผู้เล่นทำการเลือกหมากของฝ่ายตนที่ต้องการ โดยจะแสดงหมากตัวที่เลือกโดยการกลับสี (inverse color) ให้เป็นตรงกันข้ามกับสีเดิม จากนั้นให้ผู้เล่นเลือกตำแหน่งปลายทางที่ต้องการ เมื่อเลือกเสร็จแล้วโปรแกรมจะทำการตรวจสอบการเดินตามกติการที่ได้กำหนดไว้ ถ้าถูกต้องตามกติกาก็จะทำการย้ายหมากตัวที่ถูกเลือกไปยังตำแหน่งใหม่ แต่ถ้าไม่ถูกต้องตามกติกาก็จะไม่แสดงผลใด ๆ ทั้งสิ้น
ถ้าหากต้องการยกเลิกตัวหมากที่เลือกไปแล้วให้คลิกปุ่มขวาของเมาส์ (mouse) บนกระดานก็จะแสดงสีของหมากดังเดิม เมื่อคลิกซ้ายอีกครั้งก็จะเป็นการเริ่มต้นเลือกใหม่ แต่เมื่อเดินเสร็จแล้วจะไม่สามารถยกเลิกที่เล่นไปได้



ภาพที่ 3.3 แสดงกระดาน

3.4.1.2 ส่วนแสดงเวลา

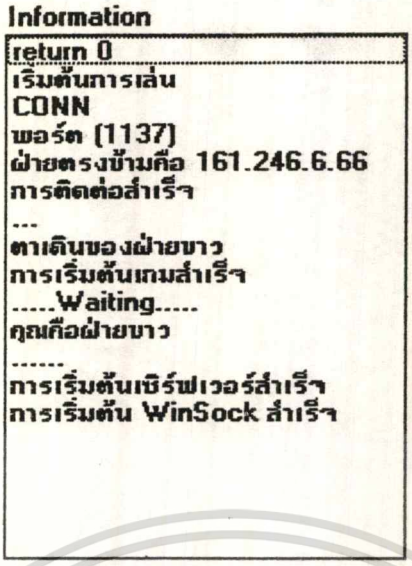
ส่วนนี้จะคอยแสดงเวลาของผู้เล่นแต่ละฝ่ายที่ได้ใช้ในการเดิน ในการเดินของแต่ละฝ่ายจะจำกัดไว้ให้เดินทั้งหมดไม่เกินฝ่ายละ 1 ชั่วโมง แต่ไม่จำกัดเวลาในการเดินของแต่ละครั้งเอาไว้กำหนดเพียงให้เวลาเดินทั้งหมดไม่เกิน 1 ชั่วโมง โดยนาฬิกาของแต่ละฝ่ายจะเริ่มเดินก็ต่อเมื่อขณะนั้นเป็นตาเดินของฝ่ายนั้นอยู่ ถ้าหากเป็นตาเดินของอีกฝ่ายหนึ่งนาฬิกาของฝ่ายตนเองจะไม่เดิน หากฝ่ายใดฝ่ายหนึ่งใช้เวลาเกินจากที่กำหนดไว้นี้ฝ่ายนั้นก็จะถูกปรับให้แพ้ทันที และการเล่นก็จะยุติลง



ภาพที่ 3.4 นาฬิกาจับเวลา

3.4.1.3 ส่วนแสดงข่าวสาร

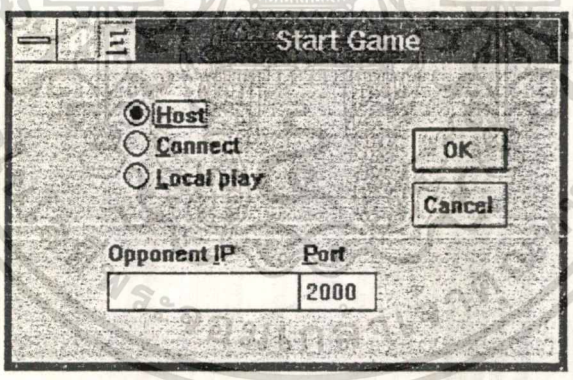
ส่วนแสดงข่าวสารจะคอยแสดงข่าวสารของตัวโปรแกรมที่ส่งให้กับผู้ใช้ เช่นตำแหน่งที่เลือก, ความผิดพลาดที่เกิดขึ้นดังภาพ



ภาพที่ 3.5 ส่วนแสดงข่าวสาร

3.4.2 การเริ่มต้นของฝ่ายเซิร์ฟเวอร์

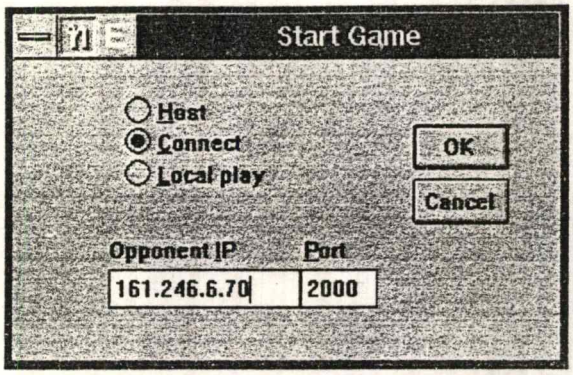
สำหรับฝ่ายที่ต้องการเริ่มต้นเป็นเซิร์ฟเวอร์นั้น เมื่อเลือกเมนู Game เพื่อทำการเริ่มต้นเล่นเกมแล้ว ให้เลือก Host ภายในไดอะลอกบ๊อกซ์ (dialog box) จากนั้นเลือกปุ่ม OK เพื่อทำการรอให้ฝ่ายไคลเอนต์ทำการติดต่อเข้ามา Port จะหมายถึงหมายเลขของพอร์ตที่ต้องการใช้ในการติดต่อ จากภาพหมายเลขของพอร์ตคือ 2000



ภาพที่ 3.6 การเริ่มต้นของฝ่ายเซิร์ฟเวอร์

3.4.3 การเริ่มต้นของฝ่ายไคลเอนต์

ฝ่ายที่ต้องการจะเล่นเกมเป็นชนิดไคลเอนต์ เมื่อเลือกเมนู Game เพื่อทำการเริ่มต้นเล่นเกมแล้ว ให้เลือก Connection ภายในไดอะลอกบ๊อกซ์ จากนั้นให้ใส่หมายเลข หรือชื่อของโฮสต์ภายในกรอบข้อความ Opponent IP address ส่วน Port เป็นเช่นเดียวกับในส่วนของเซิร์ฟเวอร์ ดังภาพ



ภาพที่ 3.7 การเริ่มต้นของฝ่ายไคลเอนต์

3.4.4 การเล่นผ่านระบบเครือข่าย

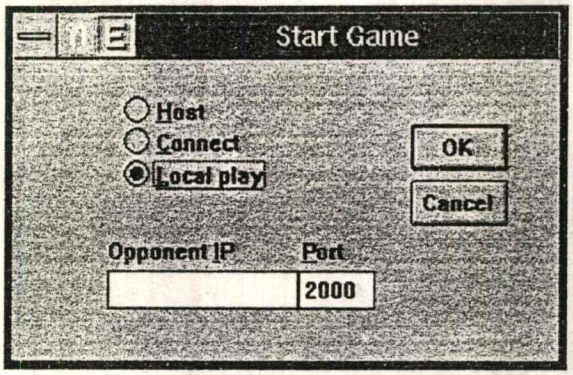
สำหรับการเล่นผ่านระบบเครือข่ายเมื่อทำการติดต่อกันสำเร็จแล้วฝ่ายโฮสต์จะเป็นฝ่ายเริ่มเล่น โดยในการเล่นครั้งแรกจะแยกได้เป็น 2 ขั้นตอนคือ

1. ฝ่ายโฮสต์ได้รับการติดต่อจากฝ่ายไคลเอนต์เรียบร้อยแล้ว ก็จะเริ่มต้นการเล่นได้ทันที โดยฝ่ายโฮสต์จะเป็นฝ่ายสีขาว ในการแสดงผลจะอยู่บริเวณส่วนล่างของหน้าต่าง สำหรับวิธีการเล่นนั้นให้เลือกตัวหมากที่ต้องการเดิน แล้วเลือกตำแหน่งปลายทางที่ต้องการ ถ้าหากตำแหน่งปลายทางที่เดินนั้นถูกต้องตามกติกาก็จะทำการแสดงผลการเดินนั้น และจะส่งข้อมูลการเล่นไปยังฝ่ายไคลเอนต์เพื่อให้ข้อมูลการเล่นของทั้งสองฝ่ายตรงกัน แล้วรอให้ฝ่ายไคลเอนต์เล่นเสร็จและส่งข้อมูลการเล่นมา
2. ฝ่ายไคลเอนต์เมื่อติดต่อกับฝ่ายโฮสต์สำเร็จแล้วจะต้องทำการรอให้ฝ่ายโฮสต์เดินเสร็จเสียก่อนจึงจะเริ่มเดินได้ โดยสังเกตจากในส่วนแสดงข่าวสารจะบอกว่าขณะนี้ฝ่ายใดเป็นฝ่ายเล่นอยู่ ฝ่ายไคลเอนต์จะเป็นฝ่ายสีดำ ส่วนในการแสดงผลจะอยู่บริเวณส่วนล่างของหน้าต่าง สำหรับวิธีการเดินนั้นก็เป็นเช่นเดียวกับฝ่ายโฮสต์คือให้เลือกหมากที่ต้องการ และตำแหน่งปลายทางที่ต้องการหากถูกต้องตามกติกาก็จะทำการเดินให้พร้อมกับส่งข้อมูลการเล่นไปยังฝ่ายโฮสต์ แต่ถ้าไม่ถูกต้องก็จะไม่ทำการเดินให้ จากนั้นก็จะรอให้ฝ่ายโฮสต์เล่นเสร็จ และส่งข้อมูลการเล่นมา

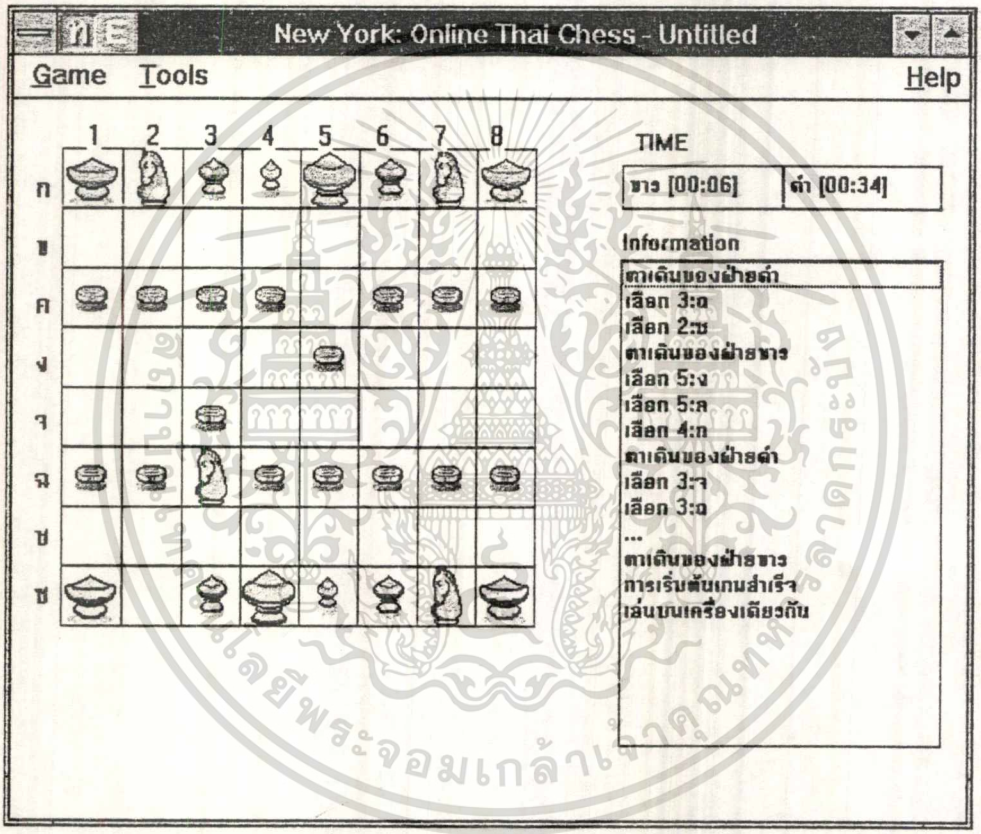
เมื่อฝ่ายไคเอนต์ส่งข้อมูลการเล่นไปให้ยังฝั่งโฮสต์แล้ว ก็จะเปลี่ยนตาเล่นเป็นของโฮสต์ จากนั้นก็จะผลัดกันเล่นจนกว่าจะรู้ผลแพ้-ชนะ หรือถูกสั่งให้ยุติการเล่น โดยที่ฝ่ายที่เดินเสร็จแล้วจะทำการรอให้อีกฝ่ายเล่นเสร็จ และส่งข้อมูลการเล่นมาให้จึงสามารถเล่นต่อได้

3.4.5 การเล่นโดยไม่ผ่านระบบเครือข่าย

โปรแกรมหมากรุกนี้สามารถเล่นบนเครื่องเดียวกันโดยไม่ต้องผ่านระบบเครือข่ายได้ วิธีเริ่มให้ทำเลือก Play local บนไดอะลอกบ็อกซ์ แล้วคลิก OK สำหรับการแสดงผลก็จะเหมือนกับการเล่นผ่านระบบเครือข่ายเพียงแต่ฝ่ายขาวจะอยู่ส่วนล่างของหน้าต่าง และฝ่ายดำจะอยู่ส่วนบนของหน้าต่าง และวิธีการเล่นก็เหมือนกับการเล่นผ่านเครือข่าย โดยฝ่ายขาว (ที่อยู่บริเวณด้านล่างของหน้าต่าง) จะเป็นฝ่ายเริ่มเล่นก่อน เมื่อเดินเสร็จแล้วจะเปลี่ยนไปตาเดินของฝ่ายดำ (อยู่บริเวณด้านบนของหน้าต่าง) เมื่อฝ่ายดำเล่นเสร็จก็จะเปลี่ยนมาเป็นตาเดินของฝ่ายขาวสลับเปลี่ยนเช่นนี้ไปเรื่อย ๆ จนกว่าการเล่นจะรู้ผลแพ้-ชนะ หรือสั่งให้ยุติการเล่น



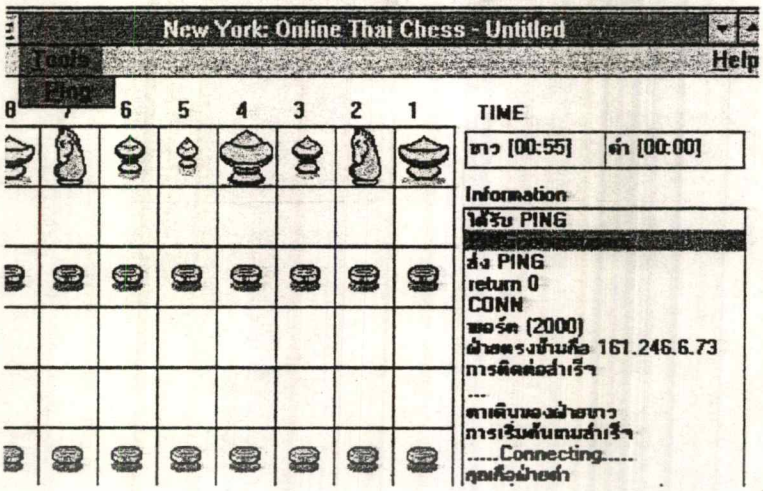
ภาพที่ 3.8 แสดงการเริ่มต้นของการเล่นไม่ผ่านระบบเครือข่าย



ภาพที่ 3.9 ตัวอย่างการเล่นโดยไม่ผ่านระบบเครือข่าย

3.4.6 การตรวจสอบสภาพการเชื่อมต่อ

เนื่องจากเล่นหมากรุกผ่านระบบเครือข่ายนี้การติดต่ออาจเกิดความผิดพลาดขึ้นได้โดยที่อีกฝ่ายไม่รู้ถึงความผิดพลาดนั้น ดังนั้นจึงจำเป็นต้องมีเครื่องมือที่ช่วยในการตรวจสอบการติดต่อกับฝ่ายตรงข้าม วิธีใช้ให้เลือก Ping จากเมนู Tools ก็จะทำการตรวจสอบสภาพการติดต่อ เมื่อฝั่งตรงข้ามได้รับก็จะตอบกลับมา หากการติดต่อยังไม่ถูกยกเลิกก็จะแสดงข้อความขึ้นมาบอกว่าฝั่งตรงข้ามได้ตอบกลับมาแล้ว แต่ถ้าหากไม่สามารถส่งข้อมูลไปยังฝั่งตรงข้าม หรือฝั่งตรงข้ามไม่สามารถส่งข้อมูลตอบได้ก็จะไม่แสดงข้อความการตอบกลับของฝั่งตรงข้ามขึ้นมาดังแสดงในภาพ



ภาพที่ 3.10 ตัวอย่างการตรวจสอบสภาพการติดต่อโดยใช้ Ping

3.5 การออกแบบออพเจคหมาก และออพเจคกระดาน

ในการเล่นหมากรุกจริง ๆ นั้นอุปกรณ์ที่จำเป็นต้องใช้สามารถแบ่งออกได้เป็นส่วนใหญ่ ๆ 2 ส่วนคือ

1. ตัวหมาก
2. กระดาน

เมื่อนำหมากรุกมาเล่นบนเครื่องคอมพิวเตอร์ก็ยังจำเป็นต้องใช้ส่วนประกอบทั้ง 2 ส่วนนี้ด้วย ดังนั้นจึงนำมาสร้างออพเจคหลักได้ 2 ออพเจคคือ MetaPiece และ CBoard

3.5.1 ออพเจค MetaPiece

ออพเจค MetaPiece นี้จะเป็นตัวออพเจคบรรพบุรุษ (ancestor) ของตัวหมากทุกชนิด คือ ขุน, เรือ, โคน, เบี้ย, เม็ด และ ม้า ข้อมูล และฟังก์ชันหลักที่เก็บในออพเจค MetaPiece คือ

1. ชนิดของตัวหมาก (เพื่อใช้อ้างอิงต่อไป)
2. สีของตัวหมาก
3. ตำแหน่งของตัวหมาก
4. ฟังก์ชันใช้ตรวจสอบว่าลักษณะการเดิน

ฟังก์ชันในข้อที่ 4 นี้จะเป็นเพียงโครงของฟังก์ชันเพื่อให้ผู้ใช้สามารถตรวจสอบความถูกต้องในการเดินได้โดยไม่จำเป็นต้องตรวจสอบชนิดของหมากก่อน ซึ่งเป็นการใช้คุณสมบัติของภาษา C++ ส่วนตรวจสอบกติกาในการเดินนี้ไม่ซับซ้อนจนเกินไป และสะดวกต่อการแก้ไขดังภาพ

```
int MetaPiece::VerifyMove( CPoint Target, int reverse)
{
        return 0;
}
```

ภาพที่ 3.11 แสดงโครงของฟังก์ชัน VerfiMove

ออพเจคลูกของ MetaPiece ได้แก่

1. CBear เป็นออพเจคใช้แทนหมากเบี้ย
2. CCone เป็นออพเจคใช้แทนหมากโคน
3. CMar เป็นออพเจคใช้แทนหมากม้า
4. CMed เป็นออพเจคใช้แทนหมากเม็ด
5. CKhun เป็นออพเจคใช้แทนหมากขุน
6. CRyur เป็นออพเจคใช้แทนหมากเรือ

ออพเจคลูกของออพเจค MetaPiece เหล่านี้จะมีส่วนประกอบเหมือนกันทุกประการ และเหมือนกับออพเจค MetaPiece ด้วย ต่างกันตรงที่ฟังก์ชันที่ใช้ในการตรวจสอบกติการของหมากแต่ละตัวนั้นจะต่างกันไปตามชนิดของตัวหมาก โดยทำการแก้ไขเพียงฟังก์ชันที่เป็นโครงร่างในภาพที่ 4 เท่านั้น

เนื่องจากฟังก์ชันที่ใช้ตรวจสอบการเดินนี้ถูกใส่ไว้ในออพเจคตัวหมาก ดังนั้นความสามารถในการตรวจกติกาของตัวหมากจึงทำได้เพียงให้ตรวจสอบความถูกต้องของลักษณะการเดินเท่านั้น เช่น เรือจะต้องเดินไปในทิศทางตรงเท่านั้น หรือ ขุนสามารถเดินได้ 8 ทิศรอบตัว แต่ได้ทีละหนึ่งช่องเท่านั้น เป็นต้น ไม่สามารถตรวจสอบกติการที่จำเป็นต้องนำหมากตัวอื่นมาร่วมพิจารณาด้วยได้ เช่น การเดินของเรือนั้นมีการกระโดดข้ามหมากตัวอื่นหรือไม่ เป็นต้น ส่วนของกติการที่ไม่สามารถตรวจสอบได้นั้นได้ถูกนำไปรวมไว้ในออพเจค CNyView แทน

3.5.2 ออพเจค CBoard

ออพเจค CBoard นี้จะทำการรวบรวมออพเจคหมากที่มีอยู่ (ใช้ในการเล่น) เปรียบได้กับกระดานหมากรุกที่มีตัวหมากวางอยู่ ข้อมูล และฟังก์ชันหลักที่เก็บได้แก่

1. ตาของผู้เล่น
2. บอกว่าผู้เล่นเป็นฝ่ายขาว หรือดำ
3. ตรวจสอบ และจัดการการหงายเบี้ย

บทที่ 4

การทดลอง และผลการทดลอง

4.1 โปรโตคอล

ในโปรแกรมหมากรุกขั้นทดลองนั้นได้ทดลองให้ทำการสื่อสารโดยไม่มีการกำหนดโปรโตคอล ทำให้ไม่ได้ตรวจชนิดของข้อมูลที่ได้รับเข้ามา เมื่อทดลองให้โปรแกรมอื่นทำการร้องขอการติดต่อมายังโฮสต์ปรากฏว่าสามารถทำการติดต่อกันได้ โดยที่ตัวโปรแกรมไม่แยกได้ว่าไคลเอนต์ที่ติดต่ออยู่นั้นเป็นโปรแกรมชนิดใด และมีการติดต่อแบบใด ทำให้โปรแกรมทั้งสองนี้ติดต่อกันได้ แต่ไม่สามารถทำงานอื่นได้ต่อไป จึงจำเป็นต้องออกแบบโปรโตคอลขึ้นมาสำหรับการเล่นหมากรุกที่มีชนิดข้อมูลที่ใช้ในการสื่อสารไม่ซ้ำกับโปรโตคอลชนิดอื่นมิฉะนั้นอาจเกิดความผิดพลาดเช่นนี้อีกได้
นอกจากนี้ในการสื่อสารผ่านระบบเครือข่ายอาจจะมีการสูญเสียการติดต่อโดยที่อีกฝ่ายไม่รู้ถึงความผิดพลาดนี้อาจจะต้องทำการรออีกฝ่ายไปเรื่อย ๆ แบบไม่มีจุดหมายได้ จึงจำเป็นต้องพัฒนาเครื่องมือเพื่อใช้ตรวจสอบสภาพวะการติดต่อนี้ด้วยเพื่อใช้ในกรณีที่ไม่แน่ใจว่าการติดต่อนั้นมีปัญหาหรือไม่

4.2 การตรวจกติการเล่น

ในการตรวจสอบการเล่นสามารถแยกออกได้เป็นส่วน ๆ ดังนี้

1. ขุน เมื่อสร้างส่วนตรวจเช็คกติการการเดินของขุนแล้ว ก็ทำการทดลองเดินตาเดินที่สามารถเดินได้ และไม่สามารถเดินได้ คือตรวจตาเดิน 8 ช่องรอบตัวเอง และทดลองเดินไปยังช่องที่สามารถถูกกินได้ รวมทั้งยังตรวจกรณีที่ขุนถูกฝ่ายตรงข้ามกินซึ่งทำให้รู้ผลแพ้-ชนะด้วย
2. เรือ ตรวจสอบการเดินในแนวตรงทั้ง 4 ทิศโดยไม่มีการข้ามหมากตัวอื่นที่อยู่ในทางนั้น
3. ม้า ตรวจสอบการเดินในแนวเฉียงข้ามไป 2 ตาทั้ง 6 ทิศ
4. โคน ตรวจสอบการเดินไปข้างหน้า 1 ช่องทั้ง 3 ทิศ และการเดินเฉียงไปข้างหลัง 2 ทิศ
5. เม็ด ตรวจสอบการเดินเฉียง 1 ช่องรอบตัวทั้ง 4
6. เบี้ย ตรวจสอบการเดินตรงไปข้างหน้าได้ 1 ตา และการกินเฉียงด้านหน้าทั้ง 2 ทิศ นอกจากนี้ยังต้องตรวจสอบการหงายเบี้ยว่ามีอำนาจเท่าเม็ดหรือไม่ด้วย

บทที่ 5
บทวิจารณ์ และสรุป

5.1 โปรโตคอล

จากการทดลองพบว่าโปรแกรมประยุกต์ทุกตัวที่ทำงานผ่านระบบเครือข่ายนั้นจำเป็นต้องมีโปรโตคอลเป็นของตัวเอง ซึ่งอาจจะออกแบบขึ้นมาใหม่ หรือนำโปรโตคอลที่มีอยู่แล้วมาดัดแปลงก็ได้ เนื่องจากจะต้องให้โปรแกรมเดียวกันนี้ที่อยู่ต่างเครื่อง หรือต่างสถานที่กันสามารถติดต่อกันได้โดยที่โปรแกรม หรือผู้ใช้คนอื่นที่ไม่ได้ใช้โปรแกรมของเราไม่สามารถเข้ามาติดต่อกันได้ มิฉะนั้นแล้วโปรแกรมที่ทำงานอาจเกิดผลที่ไม่สามารถคาดหมายได้เนื่องจากเป็นการสื่อสารคนละความหมายกัน หมายเลขพอร์ตก็เป็นส่วนสำคัญในการสื่อสาร ถ้าหากใช้ต่างพอร์ตกันก็จะไม่สามารถทำการติดต่อกันได้
สำหรับโปรแกรมหมากรุกนี้ได้ใช้โปรโตคอลที่ได้ออกแบบขึ้นมาใหม่เอง โดยอาศัยหลักการติดต่อสื่อสารเบื้องต้นมาเป็นแนวทางในการออกแบบ โดยเป็นเพียงโปรโตคอลง่าย ๆ ไม่ซับซ้อน ไม่สามารถตรวจจับความผิดพลาดที่ซับซ้อนได้ และขนาด และจำนวนของข้อมูลที่ทำการส่งมีไม่มากนัก ผลดีที่ได้รับคือค่าความสิ้นเปลืองที่ต้องเสียไปจะต่ำ ทำให้การติดต่อมีความเร็วสูง แต่ข้อเสียคือความน่าเชื่อถือจะต่ำ

5.2 Microsoft Visual C++

Visual C++ นับเป็นเครื่องมือที่มีความสามารถสูงตัวหนึ่ง การใช้จะเป็นการผสมระหว่างการทำงานแบบวิชวล กับแบบการโปรแกรมเชิงวัตถุ และการโปรแกรมในแบบเดิม ทำให้ผู้พัฒนาสามารถสร้างงานที่มีประสิทธิภาพสูงได้ โดยใช้ขั้นตอนน้อยลง ไม่ซับซ้อนมาก และใช้เวลาต่ำลงกว่าใช้วิธีเดิม แต่ปัญหาที่พบคือมันยังไม่มีความสามารถในการจัดการโปรแกรมหลายรุ่นได้ ผู้พัฒนาจะต้องทำการสำรองข้อมูลของโปรแกรมเก่าเอง เนื่องจากในการบันทึกมันจะบันทึกทับของเดิมไป ดังนั้นหากเกิดความผิดพลาดขึ้นผู้พัฒนาจำเป็นต้องทำการเรียกคืนด้วยตนเอง วิธีแก้คือผู้พัฒนาจะต้องทำการสำรองข้อมูลของโปรแกรมแต่ละรุ่นเป็นระยะ ๆ ตามความเหมาะสม นอกจากนี้ตัวเครื่องมือต้องการใช้ทรัพยากรสูงของระบบ เช่น หน่วยความจำ, หน่วยประมวลผลกลาง (CPU) และฮาร์ดดิสก์ (harddisk)

5.3 โปรแกรมหมากรุก

5.3.1 การออกแบบ

เนื่องจากใช้หลักการออกแบบแบบการโปรแกรมเชิงวัตถุ ปัญหาในช่วงแรกจึงเป็นด้านการออกแบบออพเจคที่จำเป็นสำหรับโครงงานนี้ ออพเจคแต่ละชนิดของหมากรุกจริง ๆ นั้นมีหน้าที่การทำงานคล้าย ๆ กันจึงทำให้ออพเจคที่ออกแบบมาในขั้นแรกมีความซ้ำซ้อนกันสูง เมื่อได้ออพเจคในชั้นแรกออกแล้วจึงต้องนำมาวิเคราะห์อีกครั้งหนึ่งเพื่อให้ได้ออพเจคที่แยกจากกันเป็นอิสระต่อกัน แต่ในการทำงานจริงนั้นในการทำงานระยะแรก ๆ นั้นไม่สามารถตรวจพบความซ้ำซ้อนได้

หมดเมื่อมีการแก้ไขเพิ่มเติมจึงจำเป็นต้องมีการวิเคราะห์เป็นระยะ ๆ แต่ระยะเวลาในการทำงานนั้นมีไม่มากพอจึงทำให้ไม่สามารถดอยวิเคราะห์ออพเจคเป็นระยะได้ ออพเจคที่ได้จึงยังคงมีความซ้ำซ้อนกันอยู่

5.3.2 การพัฒนา

ข้อจำกัดของโครงงานนี้คือสามารถทำการติดต่อผ่านระบบเครือข่าย TCP/IP ได้เท่านั้น สำหรับการพัฒนาต่อไปนั้นอาจพัฒนาให้สามารถส่งข้อมูลผ่านระบบเครือข่าย IPX/SPX (Internet Packet Exchange/Sequenced Packet Exchange)หรือผ่านโมเด็มได้

นอกจากนี้ส่วนติดต่อกับผู้ใช้ยังไม่สวยงามนัก ยังคงเป็นเพียงภาพ 2 มิติธรรมดา ในขั้นต่อไปอาจพัฒนาให้เป็นภาพ 3 มิติ, ภาพเคลื่อนไหว หรือเพิ่มส่วนที่เป็นเสียงเพิ่มเข้าไปได้

สุดท้ายในส่วนของโปรโตคอลนั้นยังไม่สามารถรองรับความผิดพลาดที่เกิดขึ้นได้อย่างสมบูรณ์ หากพบความผิดพลาดขึ้นมันก็จะยกเลิกการติดต่อทันที ในการพัฒนาต่อไปอาจทำให้มันสามารถทำการเชื่อมต่ออีกครั้งได้โดยที่ผู้ใช้ไม่สามารถถึงความผิดพลาดที่เกิดขึ้น แต่จะใช้งานต่อไปได้เรื่อย ๆ นอกจากนี้อาจเพิ่มเติมความสามารถของโปรโตคอลด้านอื่น ๆ อีก เช่น การเซิร์ฟเวอร์เพื่อใช้เป็นศูนย์กลางในการติดต่อทำให้สามารถซ่อนการทำงานส่วนของโฮสต์ และไคลเอนต์จากผู้ใช้ได้โดยให้ผู้ใช้ทำการติดต่อไปที่ศูนย์กลางนี้แทน

ในการจับเวลาของการเล่นนั้นยังมีข้อผิดผลาดอยู่และจะมีค่าความผิดผลาดมากหากเวลาในการส่งข้อมูลระหว่างการเล่นช้ามาก ๆ

ภาคผนวก

ตัวอย่างของโปรแกรมบางส่วน(file list)

ในภาคผนวกนี้จะแสดงบางส่วนของโปรแกรมโดยแสดงถึงตัวอย่างของคลาส (class) ต่างของตัวหมาก ได้แก่ CBear, CCone, CKhun, CMar, CMed, CRyur

Mpiece.h : header file

```
// and all its descendant class
// Remark:        Khun has not moved to let the opponent eat
//                :        Cone and Bear have to know theirs direction
// MetaPiece window
//
class MetaPiece : public CWnd
{
// Construction
public:
        MetaPiece(int Type, int Color, CPoint point);
// Attributes
public:
        int iType;                              // type of piece
        int iColor;                             // color of piece
        CBitmap Pic;                    // picture of each piece
        CPoint Pos;                             // current position of this piece
// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);  // check for rule of each piece
        void LoadPicture(void); // load piece's picture
// Implementation
public:
        virtual ~MetaPiece();
protected:
        // Generated message map functions
        //{{AFX_MSG(MetaPiece)
                // NOTE - the ClassWizard will add and remove member functions here.
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//////////////////////////////////////////////////////////////////////
// Bear class
//
class CBear : public MetaPiece
{
// Construction
public:
        CBear(int Type, int Color, CPoint point);

// Attributes
public:

// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CBear();
};

//////////////////////////////
// Cone class
//
class CCone : public MetaPiece
{
// Construction
public:
        CCone(int Type, int Color, CPoint point);

// Attributes
public:
```

```
// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CCone();
};

/////////////////////////////////////////////////
// Khun window
//
class CKhun : public MetaPiece
{
// Construction
public:
        CKhun(int Type, int Color, CPoint point);

// Attributes
public:

// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CKhun();
};

/////////////////////////////////////////////////
```

```
// Mar window
//
class CMar : public MetaPiece
{
// Construction
public:
        CMar(int Type, int Color, CPoint point);

// Attributes
public:

// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CMar();
};

/////////////////////////////////////////////////
// Piece window
//
class CMed : public MetaPiece
{
// Construction
public:
        CMed(int Type, int Color, CPoint point);

// Attributes
public:
```

```
// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CMed();
};

/////////////////////////////////////////////
// Ryur class
//
class CRyur : public MetaPiece
{
// Construction
public:
        CRyur(int Type, int Color, CPoint point);

// Attributes
public:

// Operations
public:
        virtual int VerifyMove(CPoint Target, int reverse);

// Implementation
public:
        virtual ~CRyur();
};
```

Mpiece.cpp : implementation file of MetaPiece class

```
#include "stdafx.h"
#include "ny.h"
//#include "mpiece.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// MetaPiece

MetaPiece::MetaPiece(int Type, int Color, CPoint point)
{
        iType = Type;
        iColor = Color;
        Pos = point;
        LoadPicture();
}

MetaPiece::~MetaPiece()
{
}

BEGIN_MESSAGE_MAP(MetaPiece, CWnd)
        //{{AFX_MSG_MAP(MetaPiece)
                // NOTE - the ClassWizard will add and remove mapping macros here.
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
//////////////////////////////////////////////////////////////////////

// MetaPiece message handlers


//////////////////////////////////////////////////////////////////////

// MetaPiece general functions

//


void MetaPiece::LoadPicture(void)

{

        switch (iType)

        {

                case KHUN:      if (iColor==BLACK) Pic.LoadBitmap(IDB_BKHUN);

                                        else Pic.LoadBitmap(IDB_WKHUN);

                                        break;

                case MAR:       if (iColor==BLACK) Pic.LoadBitmap(IDB_BMAR);

                                        else Pic.LoadBitmap(IDB_WMAR);

                                        break;

                case MED:       if (iColor==BLACK) Pic.LoadBitmap(IDB_BMED);

                                        else Pic.LoadBitmap(IDB_WMED);

                                        break;

                case RYUR:      if (iColor==BLACK) Pic.LoadBitmap(IDB_BRYUR);

                                        else Pic.LoadBitmap(IDB_WRYUR);

                                        break;

                case CONE:      if (iColor==BLACK) Pic.LoadBitmap(IDB_BCONE);

                                        else Pic.LoadBitmap(IDB_WCONE);

                                        break;

                case BEAR:      if (iColor==BLACK) Pic.LoadBitmap(IDB_BBEAR);

                                        else Pic.LoadBitmap(IDB_WBEAR);

                                        break;

                default:  MessageBox( "Unknown Piece type.!",

                                         "Error", MB_ICONEXCLAMATION || MB_OK);

                                        break;
```

```
        };
}

int MetaPiece::VerifyMove(CPoint Target, int reverse)
{
        return 0;
}
```

Bear.h

```
#ifndef MetaPiece
//          #include "mpiece.h"
#endif
// Piece window
class CBear : public MetaPiece
{
// Construction
public:
        virtual CBear(int Type, int Color, CPoint point);
// Attributes
public:
// Operations
public:
        virtual int VerifyMove(CPoint Target);
// Implementation
public:
        virtual ~CBear();
};
```

Bear.cpp : implementation file of Bear class

```
#include "stdafx.h"
#include "ny.h"
//#include "bear.h"
//#include "mpiece.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

CBear::CBear(int Type, int Color, CPoint point):
        MetaPiece(Type, Color, point)
{
}

CBear::~CBear()
{
}

int CBear::VerifyMove(CPoint Target, int reverse)
// return  0 : neither moving and eating
//                      1 : moving
//                      2 : eating
//                      3 : either moving and eating
{
        // check for Bear moving or eating
        int dx, dy;
        dx = Target.x - Pos.x;
        dy = Target.y - Pos.y;
        if (reverse==0)
```

```
    {
        if ( dx==0 && dy==-1 )
            return 1;
        else if ( abs(dx)==1 && dy==-1 )
            return 2;
    }
    else
    {
        if ( dx==0 && dy==1 )
            return 1;
        else if ( abs(dx)==1 && dy==1 )
            return 2;
    }
    return 0;
}
```

Cone.h

```
#ifdef MetaPiece
//        #include "mpiece.h"
#endif
// Piece window
class CCone : public MetaPiece
{
// Construction
public:
        CCone(int Type, int Color, CPoint point);
// Attributes
public:
// Operations
public:
        virtual int VerifyMove(CPoint Target);
// Implementation
public:
        virtual ~CCone();
};
```

Cone.cpp : implementation file of Cone class

```
#include "stdafx.h"
#include "ny.h"
//#include "cone.h"
//#include "mpiece.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

CCone::CCone(int Type, int Color, CPoint point):
        MetaPiece(Type, Color, point)
{
}

CCone::~CCone()
{
}

int CCone::VerifyMove(CPoint Target, int reverse)
// return  0 : neither moving and eating
//                      1 : moving
//                      2 : eating
//                      3 : either moving and eating
{
        // check for Cone moving or eating
        int dx, dy;
        dx = Target.x - Pos.x;
        dy = Target.y - Pos.y;
```

```
        if (reverse==0)
        {       // forward move
                if (dx==0 && dy==-1)
                        return 3;
        }
        else
        {       // backward move
                if (dx==0 && dy==1)
                        return 3;
        }

        // diagonal move
        if ( abs(dx)==1 && abs(dy)==1 )
                return 3;

        return 0;
}
```

Khun.h

```
#ifdef MetaPiece
//        #include "mpiece.h"
#endif


// Piece window
class CKhun : public MetaPiece
{
// Construction
public:
        CKhun(int Type, int Color, CPoint point);
// Attributes
public:
// Operations
public:
        virtual int VerifyMove(CPoint Target);
// Implementation
public:
        virtual ~CKhun();
};
```

Khun.cpp : implementation file of Khun class

```
#include "stdafx.h"
#include "ny.h"
//#include "khun.h"
//#include "mpiece.h"
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

CKhun::CKhun(int Type, int Color, CPoint point):
        MetaPiece(Type, Color, point)
{
}

CKhun::~CKhun()
{
}

int CKhun::VerifyMove(CPoint Target, int reverse)
// return  0 : neither moving and eating
//                      1 : moving
//                      2 : eating
//                      3 : either moving and eating
{
        // check for KHUN moving or eating
        BOOL validX, validY;

        if ( (abs(Target.x-Pos.x) == 1) || (Target.x==Pos.x) )
```

```
        validX = TRUE;
else
        validX = FALSE;

if ( (abs(Target.y-Pos.y)==1) || (Target.y==Pos.y) )
        validY = TRUE;
else
        validY = FALSE;

if ( validX && validY )
        return 3;           // either move or eat

return 0; // neither move nor eat
```

Mar.h

```
#ifdef MetaPiece
//        #include "mpiece.h"
#endif

// Piece window
class CMar : public MetaPiece
{
// Construction
public:
        CMar(int Type, int Color, CPoint point);
// Attributes
public:
// Operations
public:
        virtual int VerifyMove(CPoint Target);
// Implementation
public:
        virtual ~CMar();
};
```

Mar.cpp : implementation file of Mar class

```
#include "stdafx.h"
#include "ny.h"
//#include "mar.h"
//#include "mpiece.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

CMar::CMar(int Type, int Color, CPoint point):
        MetaPiece(Type, Color, point)
{

}

CMar::~CMar()
{

}

int CMar::VerifyMove(CPoint Target, int reverse)
// return  0 : neither moving and eating
//                      1 : moving
//                      2 : eating
//                      3 : either moving and eating
{
        // check for Mar moving or eating
        int dx, dy;
        dx = Target.x - Pos.x;
```

```
dy = Target.y - Pos.y;


if ( (abs(dx)==2 && abs(dy)==1) ||
        (abs(dx)==1 && abs(dy)==2) )
        return 3;
return 0;
```

Ryur.h

```
#ifdef MetaPiece
//        #include "mpiece.h"
#endif


// Piece window
class CRyur : public MetaPiece
{
// Construction
public:
        CRyur(int Type, int Color, CPoint point);

// Attributes
public:

// Operations
public:
        virtual int VerifyMove(CPoint Target);
// Implementation
public:
        virtual ~CRyur();
};
```

Ryur.cpp : Implementation file of Ryur class

```
#include "stdafx.h"

#include "ny.h"

//#include "ryur.h"

//#include "mpiece.h"

#ifdef _DEBUG

#undef THIS_FILE

static char BASED_CODE THIS_FILE[] = __FILE__;

#endif


CRyur::CRyur(int Type, int Color, CPoint point):

        MetaPiece(Type, Color, point)

{

}

CRyur::~CRyur()

{

}

int CRyur::VerifyMove(CPoint Target, int reverse)

// return  0 : neither moving and eating

//                  1 : moving

//                  2 : eating

//                  3 : either moving and eating

{

        // check for Ryur moving or eating

        int dx, dy;

        dx = Target.x - Pos.x;

        dy = Target.y - Pos.y;

        if ( (dx==0 && dy!=0) || (dx!=0 && dy==0) )

                return 3;

        return 0;

}
```

กิติกรรมประกาศ

อขอบคุณ

. อ. สมศักดิ์ วลัยรัชต์ที่เอื้อเฟื้อข้อมูล และให้คำแนะนำ
. อ. นภัทร สระเอี่ยม ที่ช่วยเอื้อเฟื้อหนังสือ
. สมาชิกชมรมหมากกระดานทุกท่านที่ให้ความกระจ่างในเรื่องกติกาหมากรุกไทย
. น้อง ๆ ที่ช่วยทำรายงาน

หนังสืออ้างอิง

1. เมืองใต้, "วิธีเล่นหมากรุกไทย และหมากกล", เสริมวิทย์บรรณาคาร, 176 หน้า, 2535
2. W.Richard Stevens, "Unix Network Programming", Prentice-Hall, 772 p.,1991
3. Arthur Dumas, "Prarmming WinSock", SAMS Publishing, 345 p.,1995
4. Ori Gurewich and Nathan Gurewich, "Master Visual C+ 1.5", SAMS Publishing, 1352 p.,1994
5. David J.Kruglinski, "Inside Visual C++", Microsoft PRESS, 598 p.,1993