เกมส์ออนไลน์บนโทรศัพท์มือถือด้วยจาวา

GAME ONLINE ON MOBILE WITH J2ME

นาย สุรวิช สินทัตตโสภณ
นาย สุรศักดิ์ ธีรเวชานนท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

เกมส์ออนไลน์บนโทรศัพท์มือถือด้วยจาวา

GAME ONLINE ON MOBILE WITH J2ME

โดย

นาย สุรวิช สินทัตตโสภณ

นาย สุรศักดิ์ ธีรเวชานนท์

อาจารย์ที่ปรึกษา

อ. ธวัชชัย สุทธิทศธรรม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เกมส์ออนไลน์บนโทรศัพท์มือถือด้วยจาวา

GAME ONLINE ON MOBILE WITH J2ME

ผู้จัดทำ

1. นาย สุรวิช สินทัตตโสภณ รหัสประจำตัว 42010402
2. นาย สุรศักดิ์ ธีรเวชานนท์ รหัสประจำตัว 42010403

_____ อาจารย์ที่ปรึกษา

(อ. ธวัชชัย สุทธิทศธรรม)

เกมส์ออนไลน์บนโทรศัพท์มือถือด้วยจาวา

นาย สุรวิช สินทัตตโสภณ 42010402
นาย สุรศักดิ์ ธีรเวชานนท์ 42010403
อ. ธวัชชัย สุทธิทศธรรม อาจารย์ที่ปรึกษา
ปีการศึกษา 2545

บทคัดย่อ

ในปัจจุบันอุตสาหกรรมอุปกรณ์สื่อสารไร้สายขยายตัวอย่างต่อเนื่อง ตลาดอุปกรณ์สื่อสารไร้สายจะเติบโตยิ่งขึ้นไปอีกเมื่อมีการนำระบบเครือข่ายอุปกรณ์สื่อสารไร้สายรุ่นที่ 3 (3rd Generation : 3G) เข้ามาใช้งาน ซึ่งเป็นระบบสื่อสารไร้สายที่รับส่งกลุ่มข้อมูลโดยใช้แบนด์วิดท์สูงขึ้น ทุกวันนี้ผู้คนนิยมรับข่าวสาร จำพวก ข่าว ความเคลื่อนไหวตลาดหุ้น และสภาพอากาศ โดยตรงจากอินเตอร์เน็ตผ่านโทรศัพท์มือถือหรือพีดีเอ (PDA) ทำให้อุปกรณ์อุตสาหกรรมอุปกรณ์สื่อสารไร้สายเติบโตตามลำพังการท่องโลกอินเตอร์เน็ตด้วยโทรศัพท์มือถือดูจะไม่เพียงพออีกต่อไป ผู้ใช้งานต้องการให้แอพพลิเคชั่นบนอุปกรณ์สื่อสารไร้สายทำงานโต้ตอบ และสามารถกำหนดค่าความต้องการได้มากขึ้น ซึ่งสิ่งนี้จะช่วยยกระดับการดำเนินชีวิต และธุรกิจให้ง่ายกว่าเดิม จาวา 2 สำหรับอุปกรณ์ขนาดเล็ก (J2ME) เป็นหนทางหนึ่งที่สามารถตอบสนองความต้องการเหล่านี้ได้

แอพพลิเคชั่นบนโทรศัพท์มือถือ มีความสำคัญในการพัฒนาให้สนองความต้องการของผู้ใช้ ไม่ว่าจะเพื่อความบันเทิง แอพพลิเคชั่นเกมส์ก็เช่นเดียวกันถูกพัฒนามาเพื่อดึงดูดความสนใจและสามารถพัฒนาเกมส์บนโทรศัพท์มือถือให้ตรงความต้องการของผู้ใช้ได้

GAME ONLINE ON MOBILE WITH J2ME

Mr. Surawich Sintattasopon

Mr. Surasak Teewachanon

Mr. Twatchai Sutitosatham Advisor

ABSTRACT

At present wireless device industry has increase slightly and may be more than this if the 3rd Generation (3G) with a high bandwidth has come in the market. Today most of people get news, financial and also weather via internet through the mobile phone or PDA so only devices that we have now is not enough. User want application that interactive and can define value. This will make life and business easy. Java 2 Micro Edition is the best choice .

Applications on mobile phone is an important developed factor to fulfill user's entertainment need. As well as game application which developed to attract user's attention and capable to evolve game on mobile phone response satisfy user need.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลายๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ ธวัชชัย สุทธิทศธรรม อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความช่วยเหลือ เอาใจใส่ และแนะนำมาตลอด ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก และที่ขาดไม่ได้ เพื่อนที่ร่วมทำวิทยานิพนธ์เล่มนี้กันมา คอยช่วยเหลือซึ่งกันและกัน

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจเอาใจใส่เสมอมา ในทุกๆด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกพระคุณอันสุดประมาณและขอกราบขอบพระคุณมา ณ ที่นี้

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันเทคโนโลยีต่างๆ มีความก้าวหน้า เช่น คอมพิวเตอร์ได้รับการพัฒนาเขียนโปรแกรมให้สามารถใช้ประโยชน์ได้ตามความเหมาะสมตามแต่ลักษณะการใช้งาน อุปกรณ์ขนาดเล็กก็เช่นเดียวกันซึ่งต้องการการพัฒนาอย่างมาก เพียงแต่อุปกรณ์ขนาดเล็กมีข้อจำกัดในการพัฒนามีมากกว่าแต่อย่างไรก็ตามในปัจจุบันอุปกรณ์ขนาดเล็กก็ได้พัฒนาอย่างมาก และเพิ่มขีดความสามารถต่างๆไม่ว่าจะเป็น ส่วนติดต่อผู้ใช้ (Graphic User Interface) จำพวกกราฟิกมีความสวยงามน่าใช้ขึ้น เสียงที่มีเครื่องดนตรีต่างๆ (Polyphonic) และที่สำคัญคือสามารถให้นักพัฒนาหรือผู้ใช้ได้เขียนโปรแกรมเพิ่มขีดความสามารถ

ในปัจจุบันการรับรู้ข่าวสารและข้อมูล การติดต่อสื่อสารจึงเป็นเรื่องสำคัญ อุปกรณ์ไร้สายได้สนองความต้องการของผู้ใช้ ไม่ว่าจะเป็น พีดีเอ (PDA) โทรศัพท์มือถือ (Mobile Phone) ในโลกแห่งเทคโนโลยี (Technology) การใช้โทรศัพท์มือถือในการสนทนาแลกเปลี่ยนข่าวสารคงจะไม่เพียงพอต่อความต้องการอีกต่อไป ผู้ใช้ต้องการแอพพลิเคชั่น (Application) บนโทรศัพท์มือถือทำงานโต้ตอบ และสามารถกำหนดค่าความต้องการได้มากขึ้น ซึ่งสิ่งนี้จะช่วยในการยกระดับการดำเนินชีวิต และการทำธุรกิจให้ง่ายกว่าเดิม

ในการสร้างแอพพลิเคชั่นบนโทรศัพท์มือถือ ผู้ผลิตได้ติดตั้งไลบรารี่ (Library) เฉพาะระบบไว้ให้ ภาษาที่ใช้ในการเขียนแอพพลิเคชั่นก็มีหลายภาษา ตั้งแต่ ซีพลัสพลัส (C++) , วิชั่วเบสิก (Visual Basic) จนไปถึงภาษาที่มีสคริปต์เฉพาะตัว แอพพลิเคชั่นที่เขียนขึ้นจึงเขียนเพื่อใช้กับอุปกรณ์หนึ่งไม่สามารถนำไปใช้กับอุปกรณ์อื่นๆได้ ภาษาจาวา (Java) นั้นได้สามารถที่จะนำไปใช้กับแพลตฟอร์ม (Platform) ต่างๆได้

บริษัทซันไมโครซิสเต็มส์ ออกแพลตฟอร์มใหม่เพิ่มเติมที่นำไปใช้กับอุปกรณ์ขนาดเล็กคือ J2ME (Java 2 Micro Edition) ซึ่งได้รับการออกแบบโดยเน้นกลุ่มผู้ใช้อุปกรณ์อิเล็กทรอนิกส์ ซึ่งรวมถึงอุปกรณ์ไร้สายเช่น โทรศัพท์มือถือ และปาล์มพีดีเอ

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 ศึกษาภาษา J2ME โดยภาษา J2ME จะลดความสามารถของภาษาจาวา ที่ใช้บนเครื่องพีซี เพื่อกำหนดขอบข่ายของงาน ในบางครั้งนั้นนักพัฒนามีข้อจำกัดในการพัฒนาแอพพลิเคชั่น ตลอดจนความเป็นไปได้ของแอพลิเคชั่นที่จะนำไปใช้กับผู้ใช้

1.2.2 เพื่อศึกษาและพัฒนาแอพพลิเคชั่นทางด้านเครือข่ายในโทรศัพท์มือถือ และสนองความต้องการผู้ใช้ในปัจจุบัน

1.2.3 เพื่อนำเสนอแนวคิดถึงแอพพลิเคชั่นแบบเครือข่าย และหลักการติดต่อสื่อสารแบบซ็อกเก็ต (Socket) เพื่อเป็นแนวทางในการนำไปใช้ในการติดต่อสื่อสารแบบอื่นๆ

1.3 ขอบเขตของงานวิจัย

ขอบเขตของงานวิจัยนี้จะใช้ภาษา J2ME ซึ่งตัวภาษานี่จะต่างกับภาษาจาวาทั่วไปโดยที่จะลดความสามารถต่างๆให้น้อยลงและในงานวิจัยนี้จะใช้ตัวอิดิเตอร์ (Editor) ในการนำเสนอการทำงานและแสดงถึงแอพพลิเคชั่นที่พัฒนาขึ้น และต้องการนำเสนอในด้านเครือข่ายที่มีเซิร์ฟเวอร์ (Server) และไคลเอนต์ (Client) ให้เซิร์ฟเวอร์รับการติดต่อการทำงานจากไคลเอนต์และจะกระจายไปที่ไคลเอนต์ทุกเครื่องที่เข้ามาอยู่ในเครือข่าย ซึ่งแอพพลิเคชั่นที่พัฒนานั้นจะเป็นเกมส์ที่ใช้เครือข่ายซ็อกเก็ตในการติดต่อ ซึ่งแต่ละการกระทำของผู้เล่นจะส่งไปประมวลผลที่เซิร์ฟเวอร์และกระจายไปให้ผู้เล่นทุกคนเห็น

1.4 วิธีดำเนินงาน

14.1 เป้าหมาย

- รู้โครงสร้างพร้อมทั้งวิธีการเขียนโปรแกรมด้วย J2ME
- สร้างแอพพลิเคชั่นขึ้นมาโดยใช้ความรู้จากการศึกษา J2ME

14.2 ขั้นตอนการดำเนินงาน

- หาข้อสรุปและขอบเขตของโครงงาน
- ศึกษาหาความรู้เกี่ยวกับโครงงาน และส่วนที่เกี่ยวข้อง
- ทำการออกแบบแอพพลิเคชั่น
- สร้างแอพพลิเคชั่นที่ได้ทำการออกแบบไว้
- ทดสอบแอพพลิเคชั่นที่ได้สร้างขึ้น

บทที่ 2
จาวาสำหรับอุปกรณ์ไร้สาย

2.1 ความเป็นมา

หลังจากที่บริษัทซัน ไมโครซิสเต็มส์ออกจาวา 2 สแตนดาร์ทอิดิชั่น (Java 2 Standard Edition : J2SE) และจาวา 2 เอ็นเทอร์ไพร์สอิดิชั่น (Java 2 Enterprise Edition : J2EE) ก็ได้ออกแพลตฟอร์มใหม่เพิ่มเติม คือ J2ME ซึ่งได้รับการออกแบบโดยเน้นกลุ่มผู้ใช้อุปกรณ์และเครื่องใช้อิเล็กทรอนิกส์

2.2 ภาพรวม

ภาษาจาวาได้กลายเป็นแพลตฟอร์มการพัฒนาแอพพลิเคชั่นแบบเชิงวัตถุ (Object-oriented) ในอุปกรณ์และงานต่างๆ อย่างเต็มตัว นับตั้งแต่แอพพลิเคชั่นเซิร์ฟเวอร์ระดับองค์กร เครื่องคอมพิวเตอร์แบบตั้งโต๊ะทั่วไป จนถึงแอพพลิเคชั่นฝังตัวสำหรับอุปกรณ์ขนาดเล็ก

แพลตฟอร์มจาวา 2 ที่ใช้งานอยู่ในขณะนี้มี 3 รุ่นด้วยกัน

- จาวา 2 เอ็นเทอร์ไพร์สอิดิชั่น ใช้งานกับแอพพลิเคชั่นบนเครื่องเซิร์ฟเวอร์สำหรับองค์กรที่รองรับระบบงานใหญ่ๆ และไคลเอนต์จำนวนมาก
- จาวา 2 สแตนดาร์ทอิดิชั่น ใช้งานกับแอพพลิเคชั่นบนเครื่องคอมพิวเตอร์แบบตั้งโต๊ะทั่วไป
- J2ME ใช้งานกับแอพพลิเคชั่นรุ่นใหม่ซึ่งเน้นกลุ่มผู้ใช้อุปกรณ์อิเล็กทรอนิกส์และอุปกรณ์ฝังตัว (Embeded)

2.3 J2ME คืออะไร

J2ME เป็นแพลตฟอร์มจาวาที่ออกแบบมาเพื่อใช้กับแอพพลิเคชั่นที่ทำงานบนอุปกรณ์ขนาดเล็ก เช่น โทรศัพท์มือถือ พีดีเอ โทรศัพท์พร้อมจอภาพที่ต่อกับอินเตอร์เน็ต โทรทัศน์ดิจิตอลขนาดเล็ก อุปกรณ์บันเทิงและระบบนำทางในรถยนต์ สวิตช์ในระบบเครือข่าย คอมโพเน็นท์ของเครื่องอำนวยความสะดวกภายในบ้าน ฯลฯ

J2ME ได้นำโครงสร้างแบบโมดูลที่มีความยืดหยุ่นสูงเข้ามาใช้ เพื่อให้สามารถสนับสนุนการทำงานอุปกรณ์หลากหลายประเภท J2MEกำหนดชั้นของซอฟต์แวร์ (Software) ไว้ 3 เลเยอร์ (Layer) ด้วยกัน โดยเลเยอร์ทั้งหมดจะอยู่เหนือชั้นระบบปฏิบัติการของอุปกรณ์ ดังนี้

2.3.1 เลเยอร์จาวาเวอร์ชวลแมชชีน (Java Virtual Machine) เป็นเลเยอร์ของจาวาเวอร์ชวลแมชชีนปรับแต่งให้เข้ากับระบบปฏิบัติการของอุปกรณ์ และรองรับแต่ละ คอนฟิกูเรชั่น (Configuration) ของ J2ME และเวอร์ชวลแมชชีน ของ J2MEได้แก่ (Connected Virtual Machine : CVM) และ (Kilobyte Virtual Machine : KVM) ทั้ง CVM และ KVM ต่างสนับสนุนยูติลิตี้ JavaCodeCompact หรือคลาส Prelinker preloader และ ROmnizer ยูติลิตี้นี้จะโยงจาวาคลาสเข้ากับเวอร์ชวลแมชชีน ช่วยลดระยะเวลา

เริ่มใช้งาน VM ได้มากขึ้น ตัวอย่างเช่น เรียก KVM และ CLDC ขึ้นมาก่อนโหลดไปยังหน่วยความจำแบบ ROM ของอุปกรณ์ คลาส CLDC จะถูกโยงเข้าใน KVM โดยตรง เพื่อปรับปรุงประสิทธิภาพการทำงาน เรา เรียกขั้นตอนนี้ว่าการโหลดเข้าหน่วยความจำแบบ ROM (ROMized)

2.3.2 เลเยอร์คอนฟิกูเรชั่น เป็นเลเยอร์ของคอนฟิกูเรชั่นของ J2ME ซึ่งกำหนดคลาสไลบรารี่ อุปกรณ์ใช้งานทั่วไปหรือกลุ่มอุปกรณ์ที่มีความต้องการหน่วยความจำและหน่วยประมวลผลใกล้เคียงกัน คอนฟิกูเรชั่นในจาวาสำหรับอุปกรณ์ขนาดเล็ก มี 2 ประเภทด้วยกัน คือ CDC (Connected Device Configuration) และ CLDC (Connected Limited Device Configuration)

2.3.3 เลเยอร์โพรไฟล์ (Profile) เป็นเลเยอร์ที่สร้างเหนือเลเยอร์คอนฟิกูเรชั่น โดยกำหนดคลาสไลบรารี่เพื่อสนองตอบความต้องการของตลาดเฉพาะกลุ่ม ตัวอย่างของโพรไฟล์ในเลเยอร์นี้ ได้แก่ PDA Profile , MID Profile , Foundation Profile และ Personal Profile

2.4 คอนฟิกูเรชั่นใน J2ME

คอนฟิกูเรชั่นและโพรไฟล์เป็นองค์ประกอบหลักของ J2ME โดยมีจุดสำคัญ คือ เพื่อปรับแต่งเวอร์ชวลแมชชีนและคลาสไลบรารี่ให้เหมาะสมกับอุปกรณ์แต่ละประเภท คอนฟิกูเรชั่น คือ ชุดที่มีคุณสมบัติขั้นต่ำของ จาวาเวอร์ชวล แมชชีน และจาวาคลาสสำหรับอุปกรณ์แต่ละประเภท เป็นตัวแทนของอุปกรณ์ที่มีลักษณะคล้ายคลึงกัน อาจกล่าวได้ว่า คอนฟิกูเรชั่นเป็นตัวกำหนดคุณสมบัติหรือไลบรารี่ขั้นต่ำของแพลตฟอร์มจาวาที่นักพัฒนาคาดว่าจะต้องมีในทุกอุปกรณ์ ขณะที่คลาสไลบรารี่ที่กำหนดในคอนฟิกูเรชั่นจะมีในทุกอุปกรณ์ที่อยู่ในกลุ่มเดียวกัน

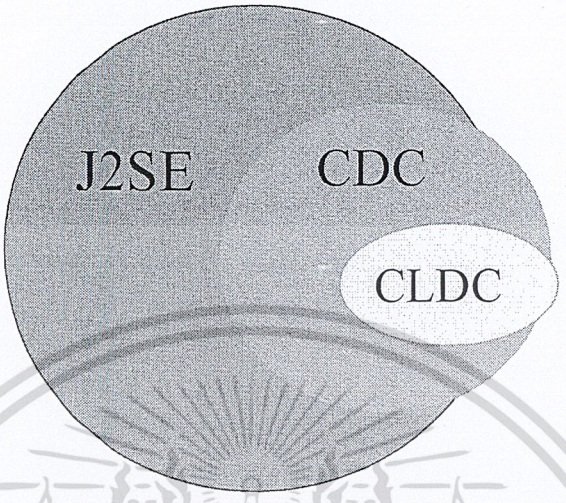
ปัจจุบันคอนฟิกูเรชั่นใน J2ME แบ่งเป็น 2 ประเภทคือ CDC (Connected Decive Configuration) และ CLDC (Connected Limited Device Configuration)

2.4.1 CDC เป็นอุปกรณ์ใช้ร่วมกับอุปกรณ์อื่น ติดตั้งตายตัว และใช้เชื่อมต่อข้อมูล โดยปรกติมักมีส่วนติดต่อกับผู้ใช้หลากหลายแบบ มีหน่วยความจำประมาณ 2-16 เม็กกะไบต์ ใช้หน่วยประมวลผลแบบ 32 บิตหรือมากกว่า เชื่อมต่อเครือข่ายที่มีแบนด์วิดท์ (Bandwidch) สูงอย่างต่อเนื่องโดยอาศัยพอร์ต TCP/IP ตัวอย่างอุปกรณ์ประเภทนี้ ได้แก่ โทรทัศน์ดิจิตอลขนาดเล็ก อินเตอร์เน็ตทีวี โทรศัพท์พร้อมจอภาพที่เชื่อมต่อกับอินเตอร์เน็ต อุปกรณ์สื่อสารที่มีความซับซ้อนสูง อุปกรณ์บันเทิงและระบบนำทางในรถยนต์

2.4.2 CLDC เป็นอุปกรณ์ส่วนบุคคล พกพาได้ และใช้เชื่อมต่อข้อมูล โดยปรกติมักมีส่วนติดต่อกับผู้ใช้แบบง่ายๆเมื่อเทียบกับระบบบนเครื่องคอมพิวเตอร์ตั้งโต๊ะ มีหน่วยความจำประมาณ 128 กิโลไบต์ – 1 เม็กกะไบต์ ใช้หน่วยประมวลผลแบบ 16 หรือ 32 บิต เชื่อมต่อกับเครือข่ายที่มีแบนด์วิดท์ต่ำเป็นระยะเวลาสั้นๆ โดยไม่อาศัยพอร์ต TCP/IP ตัวอย่างอุปกรณ์ประเภทนี้ ได้แก่ โทรศัพท์มือถือแบบไม่ซับซ้อนมากนัก เพจเจอร์รับส่งข้อความ เครื่องปาล์มโอเอสแบบพกพา

ในชั้นของคอนฟิกูเรชั่น มีคลาส 2 ประเภทด้วยกัน คือ คลาสที่นำมาจาก J2SE และที่ออกแบบเฉพาะอุปกรณ์ขนาดเล็ก คลาสที่นำมาจาก J2SE จะมีคุณสมบัติอย่างเดียวกันกับคลาสใน J2SE หรือเป็นซับคลาสของ J2SE เช่น แพ็คเก็จ java.io และ java.util จากรูป 2.1 แสดงความสัมพันธ์ระหว่าง J2SE และ

คลาสไลบรารี่ของ CDC และ CLDC จากภาพจะเห็นว่าคลาสใน CLDC ส่วนใหญ่ทำงานร่วมกับคลาสใน CDC ได้ดีเช่นเดียวกับกรณีของคลาสใน CLDC และคลาสใน J2SE



รูป 2.1 แสดงความสัมพันธ์ระหว่าง J2SE และคลาสไลบรารี่ใน CDC และ CLDC

คลาสที่ไม่ได้นำมาจาก J2SE และออกแบบเพื่อใช้เฉพาะอุปกรณ์มักทำงานร่วมกับ J2SE ได้ไม่ค่อยดีใน CLDC คลาสเหล่านี้จะอยู่ในกลุ่มกรอบการติดต่อสื่อสารทั่วไป (Generic Connection Framework) โดยระบุไว้ในแพ็คเก็จ javax.microedition.io

คอนฟิกูเรชั่นยังระบุคุณสมบัติของเวอร์ชวลแมชชีน ในเลเยอร์ด้านล่างอีกด้วย ในโครงสร้างปัจจุบัน CDC และ CLDC มีเวอร์ชวลแมชชีนที่ปรับแต่งมาเฉพาะตัวอยู่แล้ว เวอร์ชวลแมชชีนของ CDC คือ C Virtual Machine (CVM) มีคุณสมบัติครบถ้วนเหมือน Java 2 Virtual Machine แต่ขนาดเล็กกว่า ออกแบบมาสำหรับอุปกรณ์ที่สลับซับซ้อน CVM มีความต้องการหน่วยความจำ 256 กิโลไบต์ ขณะที่หน่วยความจำแบบ ROM ของ CDC มีขนาด 1 เม็กกะไบต์ เวอร์ชวลแมชชีนของ CLDC คือ K Virtual Machine (KVM) แม้จะมีขนาดเล็กแต่มีความสามารถในการทำงานสูง ออกแบบมาเพื่อใช้งานกับอุปกรณ์ที่มีทรัพยากรจำกัด K ใน KVM หมายถึง กิโล โดยเรียกตามหน่วยความจำที่นับเป็นกิโลไบต์ ส่วนคอมพิวเตอร์แบบตั้งโต๊ะนั้นหน่วยนับเป็นเม็กกะไบต์ KVM เหมาะจะนำมาใช้กับหน่วยประมวลผล RISC/CISC แบบ 16/32 บิต ซึ่งมีหน่วยความจำทั้งหมดไม่กี่ร้อยกิโลไบต์เท่านั้นประมาณ 128 กิโลไบต์ ปัจจุบัน KVM มีความต้องการหน่วยความจำอยู่ในช่วง 40 – 80 กิโลไบต์

2.5 โพรไฟล์ของ J2ME

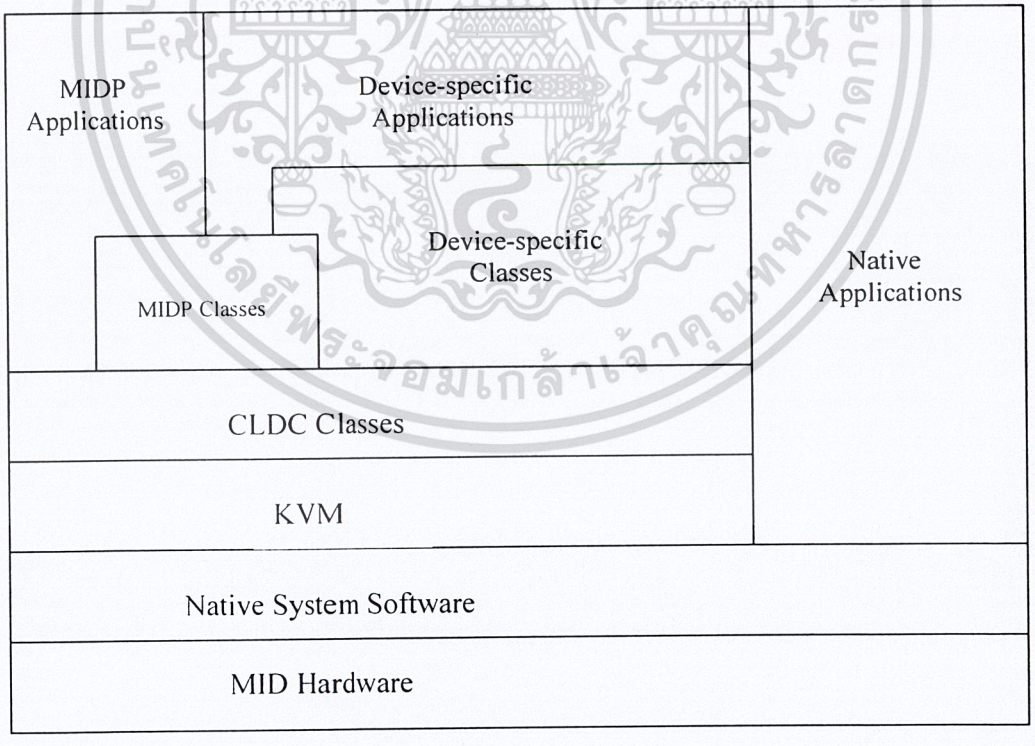
โพรไฟล์จะกำหนดชุดของ API ที่ต้องใช้เพิ่มเติม ตลอดจนกำหนดคุณสมบัติที่เป็นที่ต้องการของตลาดเฉพาะกลุ่มหรืออุปกรณ์เฉพาะประเภท คลาสไลบรารี่ในโพรไฟล์ช่วยให้นักพัฒนาสร้างฟังก์ชั่นเฉพาะอุปกรณ์ เช่น ส่วนติดต่อกราฟิกกับผู้ใช้การเชื่อมต่อเครือข่าย หน่วยเก็บข้อมูล Persistent Storage

ฯลฯ ตามปรกติแล้วจะไม่สามารถนำคลาสไลบรารี่ที่สร้างเพื่อใช้งานกับโพรไฟล์หนึ่งไปใช้กับโพรไฟล์อื่นได้ ขณะนี้มีบางโพรไฟล์ได้ถูกกำหนดแล้ว แต่ก็มีหลายโพรไฟล์ที่ยังอยู่ในขั้นตอนการดำเนินการอยู่ โพรไฟล์ Foundation และ RMI ซึ่งสร้างบน CDC กำลังเผยแพร่อยู่ ส่วนที่สร้างบน CLDC มีโพรไฟล์เดียว คือ MID Profile (Mobile Information Device Profile : MIDP) มันจะเตรียมส่วนติดต่อผู้ใช้และหน่วยเก็บข้อมูล Persistent Storage ความสามารถด้านเครือข่าย แบบจำลอง API สำหรับแอพพลิเคชั่น ไว้ให้อุปกรณ์ไร้สาย เช่น โทรศัพท์ที่ไม่ซับซ้อนมากนักและเพจเจอร์รับส่งข้อความ ส่วนโพรไฟล์ พีดีเอ (PDAP) กำลังได้รับการวิจัยอยู่

อุปกรณ์หนึ่งๆอาจมีโพรไฟล์ใช้งานมากกว่า 1 ชนิด และบางโพรไฟล์ใช้งานเฉพาะบางอุปกรณ์หรือแอพพลิเคชั่นเท่านั้น ตัวอย่างเช่น โพรไฟล์บน CDC ส่วนใหญ่ เช่น RMI และ Personal จะสร้างไว้เหนือ Profile Foundation หากไม่มี Profile Foundation และ CDC รองรับแอพพลิเคชั่นที่เขียนก็ไม่สามารถทำงานได้

2.6 J2ME สำหรับอุปกรณ์ไร้สาย

J2ME ได้ให้กำเนิดแอพพลิเคชั่นยุคใหม่บนอุปกรณ์ไร้สาย ช่วยให้เกมส์แบบหลายผู้เล่นที่ทำงานผ่านอินเตอร์เน็ตการทำธุรกรรมทางโทรศัพท์มือถือ แอพพลิเคชั่นสำหรับองค์กรทั้งไคลเอนต์และเซิร์ฟเวอร์ เกิดขึ้นได้บนโทรศัพท์มือถือและเพจเจอร์รับส่งข้อความ MIDP CLDC และ KVM ได้กลายมาเป็นรากฐานในการพัฒนาจาวาแอพพลิเคชั่นบนอุปกรณ์ไร้สายยุคใหม่ มี 3 ปัจจัยในการสร้างแพลตฟอร์มสำหรับ แอพพลิเคชั่นไร้สาย คือ MIDP CLDC KVM



รูป 2.2 โครงสร้างของเลเยอร์ต่างๆ

โครงสร้างของเลเยอร์ต่างๆจากล่างขึ้นบน ดังนี้

1. เลเยอร์ฮาร์ดแวร์ MID หมายถึง ตัวโทรศัพท์มือถือ

2. เลเยอร์ซอฟต์แวร์ของระบบที่ติดตั้งมากับอุปกรณ์ หมายถึง ระบบปฏิบัติการและไลบรารี่ของระบบที่บริษัทผู้ผลิตให้มา

3. เลเยอร์ KVM เป็นส่วนที่เตรียม Runtime Environment ไว้ให้แอพพลิเคชั่นบนอุปกรณ์ไร้สาย

4. เลเยอร์ CLDC เป็นส่วนที่ตรียม API หลักของจาวาให้แอพพลิเคชั่นบนอุปกรณ์ไร้สาย

5. เลเยอร์ MIDP เป็นส่วนที่เตรียมไลบรารี่สำหรับส่วนติดต่อกราฟิกกับผู้ใช้ และหน่วยเก็บข้อมูล Persistent Storage ระบบเครือข่าย และไทม์เมอร์

นอกจากคลาสไลบรารี่สำหรับ MIDP ผู้ผลิตอาจเตรียมคลาสไลบรารี่เฉพาะอุปกรณ์ไว้ให้นักพัฒนา เพื่อดึงความสามารถของฟังก์ชั่นที่มีอยู่แล้วไปใช้งานให้เกิดประโยชน์สูงสุด อาทิ การโทรศัพท์ การแชร์ข้อมูลกับแอพพลิเคชั่นที่ติดตั้งมาในเครื่อง จำพวก ปฏิทิน สมุดจดที่อยู่ การตรวจสอบข้อมูลอุปกรณ์ เช่น อายุแบตเตอรี่ ความแรงของสัญญาณ ฯลฯ แม้ว่าการนำคลาสเฉพาะอุปกรณ์ที่ผู้ผลิตเตรียมไว้มาใช้งาน จะช่วยเพิ่มความสามารถแก่แอพพลิเคชั่นบนอุปกรณ์ไร้สาย แต่ไม่สามารถเคลื่อนย้ายไปสู่อุปกรณ์อื่นที่ใช้ MIDP ได้ เนื่องจากคลาสที่นำมาใช้อยู่นอกเหนือขอบเขตของ MIDP

2.7 ความต้องการของระบบ

2.7.1 อุปกรณ์ไร้สายจะทำงานสนับสนุน J2ME ได้ดีเมื่อมีคุณสมบัติตรงตามข้อกำหนด หากต้องการให้ KVM ทำงานได้อย่างมีประสิทธิภาพ โดยใช้ไลบรารี่ CLDC จะต้องมีคุณสมบัติของระบบขั้นต่ำ ดังนี้

- มีหน่วยความจำ 160 – 512 กิโลไบต์ สำหรับสร้างแพลตฟอร์มจาวา
- มีหน่วยประมวลผลแบบ 16-32 บิตความเร็ว 25 เม็กกะเฮิรซ์
- ใช้พลังงาน โดยมากมักทำงานโดยใช้แบตเตอรี่
- เชื่อมต่อกับเครือข่ายได้ในช่วงสั้นๆ อาศัยระบบไร้สายเป็นส่วนใหญ่ แบนด์วิดท์ค่อนข้างจำกัด ความเร็ว 9600 ไบต์ต่อวินาทีหรือน้อยกว่า
- มีหน่วยความจำชั่วคราวขนาด 32 กิโลไบต์ สำหรับเก็บจาวา รันไทม์และหน่วยความจำของอ็อบเจกต์

2.7.2 การใช้งาน MIDP ฮาร์ดแวร์จะต้องมีคุณสมบัติดังนี้

2.7.2.1 การแสดงผล

- หน้าจอขนาด 96*54
- ความลึกของสี 1 บิต
- สัดส่วนของภาพ 1:1

2.7.2.2 การรับข้อมูลเข้า

- ใช้กลไกการป้อนข้อมูลอย่างไรอย่างหนึ่ง อาทิ แป้นพิมพ์ หรือจอสัมผัส

2.7.2.3 หน่วยความจำ

- หน่วยความจำถาวรขนาด 128 กิโลไบต์สำหรับเก็บคอมโพเน็นท์ของ MIDP
- หน่วยความจำถาวรขนาด 8 กิโลไบต์สำหรับเก็บข้อมูลที่แอพพลิเคชั่นสร้างขึ้น
- หน่วยความจำชั่วคราวขนาด 32 กิโลไบต์ สำหรับเก็บจาวา รันไทม์ เช่น จาวา ฮีป

2.7.2.4 เครือข่าย

- เครือข่ายรับส่งข้อมูลผ่านระบบไร้สาย เชื่อมต่อได้ในช่วงสั้นๆ และมีแบนด์วิดท์จำกัด

บทที่ 3
หลักการเขียนโปรแกรม

3.1 CLDC

สภาพแวดล้อมการทำงานของจาวาใน MIDP มักมาพร้อมกับ K Virtual Machine (KVM) ไลบรารี่ของ CLDC และ MIDP และซอฟต์แวร์จัดการแอพพลิเคชั่น (Application Management Software : AMS) ไลบรารี่จาวาใน CLDC และ MIDP จะเป็นพื้นฐานรองรับการเขียนแอพพลิเคชั่นบนอุปกรณ์ไร้สาย ไลบรารี่ของ CLDC เป็นไลบรารี่ของเครือข่ายและระบบชั้นสูงที่ไม่จำกัดเฉพาะอุปกรณ์ใดอุปกรณ์หนึ่ง ประกอบด้วย คลาส 2 ประเภท คือ คลาสที่เป็นซับเซ็ตของ J2SE และคลาสที่เกี่ยวกับกรอบการติดต่อสื่อสารทั่วไป

คลาสประเภทแรกของ CLDC เป็นซับคลาสของไลบรารี่ J2SE กำหนดไว้ในแพ็กเกจ java.lang java.util และ java.io โดยคลาสระบบและคลาสชนิดข้อมูลเข้ากันได้กับ J2SE และ J2EE และเพื่อให้เข้ากันได้เคลื่อนย้ายข้ามแพลตฟอร์มได้สะดวก คลาสประเภทนี้จะใช้ชื่อคลาสและชื่อแพ็คเกจตามชื่อคลาสอย่างเดียวกันใน J2SE หรือตามชื่อซับคลาสใน J2SE ที่เกี่ยวเนื่องกัน โดยไม่มีการเพิ่มเมธ็อด public หรือ protected หรือฟิลด์ซึ่งไม่มีในคลาสที่เกี่ยวเนื่องกันใน J2SE ความหมายของคลาสและเมธ็อดจึงไม่เปลี่ยนไป

คลาสประเภทที่สองของ CLDC กำหนดไว้ในแพ็คเกจ javax.microedition.io คลาสเหล่านี้มีเฉพาะใน CLDC จึงไม่สามารถทำงานร่วมกับไลบรารี่ของ J2SE ได้ เป็นคลาสชั้นสูงเกี่ยวกับเรื่องทั่วไปของเครือข่าย กำหนดไว้ในแพ็กเกจ java.io และ java.net คลาสประเภทนี้จะเตรียมกรอบการติดต่อสื่อสารผ่านเครือข่ายให้กับอุปกรณ์ที่สนับสนุน J2ME โดยกรอบการสื่อสารนี้มักเรียกกันว่า Generic Connection Framework คลาสส่วนใหญ่ได้แก่อินเตอร์เฟสที่แทนการสื่อสารรูปแบบต่างๆ เช่น ซ็อกเก็ต ดาต้าแกรม ซีเรียล และ http ขึ้นอยู่กับว่าผู้ผลิตอุปกรณ์หรือผู้ให้บริการเครือข่ายจะติดตั้งอินเตอร์เฟสการเชื่อมต่อเหล่านี้เพียงบางตัวหรือครบทั้งชุด เมื่อพิจารณาจากความสามารถของอุปกรณ์และเครือข่าย

ข้อจำกัดของ CLDC CLDC และ KVM ออกแบบมาเพื่อใช้กับอุปกรณ์ที่มีทรัพยากรจำกัด จึงไม่สามารถสนับสนุนคุณสมบัติและฟังก์ชั่นได้หลากหลายเท่า J2SE คุณสมบัติบางอย่างที่มีใน J2SE ถูกดึงออกจาก CLDC และ KVM เพื่อลดขนาดและปรับปรุงประสิทธิภาพในการทำงาน โดยจะต้องศึกษาข้อจำกัดของ J2ME ก่อนที่จะเริ่มออกแบบและพัฒนาแอพพลิเคชั่น

3.2 ประเภทของข้อมูลพื้นฐาน

CLDC สนับสนุนเฉพาะซับเซตของชนิดข้อมูลที่นำมาจาก J2SE ซึ่งได้แก่ byte short int long char และ boolean แต่ไม่สนับสนุนข้อมูลประเภท float และ double เนื่องมาจากสาเหตุสองประการ คือ อุปกรณ์เป้าหมายที่นำ CLDC ไปใช้ ส่วนใหญ่ไม่มีฮาร์ดแวร์รองรับตัวเลขทศนิยม และหากจะใช้ซอฟแวร์

เข้าช่วยก็จะต้องเสียค่าใช้จ่ายสูงมาก นอกจากนี้ CLDC ยังกำหนดคลาส type wrapper สำหรับข้อมูลทุกประการ ได้แก่

java.lang.Boolean

java.lang.Byte

java.lang.Character

java.lang.Integer

java.lang.Long

java.lang.Short

คลาส type wrapper กำหนดไว้ในจาวา เนื่องจากจาวามีระบบย่อยหลายระบบซึ่งทำงานได้เฉพาะกับอ็อบเจกต์เท่านั้นในกรณีนี้ คุณสามารถสร้างออบเจ็กต์โดยใช้คลาส wrapper ซึ่งเก็บชนิดของข้อมูลที่นำมาจาก J2SE เอาไว้

3.3 ไลบรารี่ของ MIDP

ไลบรารี่ของ CLDC ช่วยสร้างฟังก์ชั่นที่ไม่จำกัดเฉพาะอุปกรณ์บางชนิด ไลบรารี่ของ MIDP กลับตรงกันข้ามได้แก่ ถี่จัดการแอพพลิเคชั่นบนอุปกรณ์ ส่วนการติดต่อกราฟิกกับผู้ใช้ทั้งแบบพื้นฐานและซับซ้อน พื้นที่เก็บข้อมูลแบบคงตัว และความสามารถเพิ่มเติมด้านเครือข่าย

3.4 คลาส

คลาสโปรแกรมจัดการแอพพลิเคชั่น เป็นคลาสที่ติดต่อกับโปรแกรมจัดการแอพพลิเคชั่นบนอุปกรณ์ถูกกำหนดไว้ในแพ็คเกจ java.microedition.midlet แอพพลิเคชั่นทั้งหลายที่เขียนใน MIDP จะต้องขยายคลาส MIDlet ที่อยู่ในแพ็คเกจออกไป และจะต้องนำเมธ็อดทั้ง 3 ซึ่งได้แก่ startApp() pauseApp() และ destroyApp() เข้ามาใช้

คลาสของส่วนติดต่อกราฟิกกับผู้ใช้ (GUI Class) เป็นชุดเครื่องมือ Abstract Windowing Toolkit ใน J2SE ออกแบบมาเพื่อใช้งานกับแอพพลิเคชั่นบนเครื่องคอมพิวเตอร์แบบตั้งโต๊ะและไม่สามารถนำมาใช้กับอุปกรณ์ไร้สายได้ เนื่องจากบริโภคทรัพยากรประเภทหน่วยความจำค่อนข้างมาก MIDP มีวิธีที่ต่างกันในการกำหนดไลบรารี่สำหรับแพ็คเกจส่วนติดต่อกราฟิกกับผู้ใช้ ด้วยการใช้ API ขั้นสูงซึ่งเน้นความสามารถในการเคลื่อนย้ายข้ามอุปกรณ์ และ API ขั้นพื้นฐานซึ่งเน้นองค์ประกอบกราฟิกเฉพาะอุปกรณ์ และ input event ทั่วไป คลาสที่เชื่อมต่อกับส่วนติดต่อกราฟิกกับผู้ใช้ และ event-handling กำหนดไว้ในแพ็คเกจ java.microedition.lcdui Screen ถือเป็นซูเปอร์คลาสของคอมโพเน็นท์ส่วนติดต่อกับผู้ใช้โดยใช้ API ชั้นสูง ประกอบไปด้วย Alert , Form , List , Textbox ฯลฯ Canvas และ Graphic เป็นคลาสหลักของ API ระดับต่ำ (Low Level) แอพพลิเคชั่นเกมส์เป็นแอพพลิเคชั่นที่ใช้ API ระดับต่ำในการสร้างส่วนติดต่อผู้ใช้

คลาสของพื้นที่เก็บข้อมูลแบบคงตัว ในบางครั้งแอพพลิเคชั่นที่เขียนบน MIDP จำเป็นต้องเก็บข้อมูลไว้บนอุปกรณ์อย่างถาวร คลาสที่กำหนดไว้ในแพ็คเกจ java.microedtion.rms ให้กลไกการเก็บ

ข้อมูลถาวรที่เรียกว่า RecordStore โดยยอมให้แอพพลิเคชั่นเขียนลบและปรับปรุงบันทึกข้อมูลในหน่วยเก็บข้อมูลถาวรในอุปกรณ์

คลาสของเครือข่าย แม้ว่ากรอบการติดต่อสื่อสารทั่วไปที่กำหนดไว้ใน CLDC จะประกอบไปด้วยชุดของอินเตอร์เฟสการเชื่อมต่อเครือข่าย แต่ก็ไม่มีโปรโตคอลอยู่เบื้องหลังอินเตอร์เฟสการเชื่อมต่อจริงๆ หากแต่ปล่อยไว้ให้เป็นหน้าที่ของ MIDP ในบรรดาอินเตอร์เฟสการเชื่อต่อเครือข่ายเหล่านี้ HttpConnection ถือเป็นอินเตอร์เฟสหลักที่ต้องมีใน MIDP เสมอ คลาสของอินเตอร์เฟสเหล่านี้กำหนดในแพ็คเกจ java.microedition.io

3.5 MIDlet

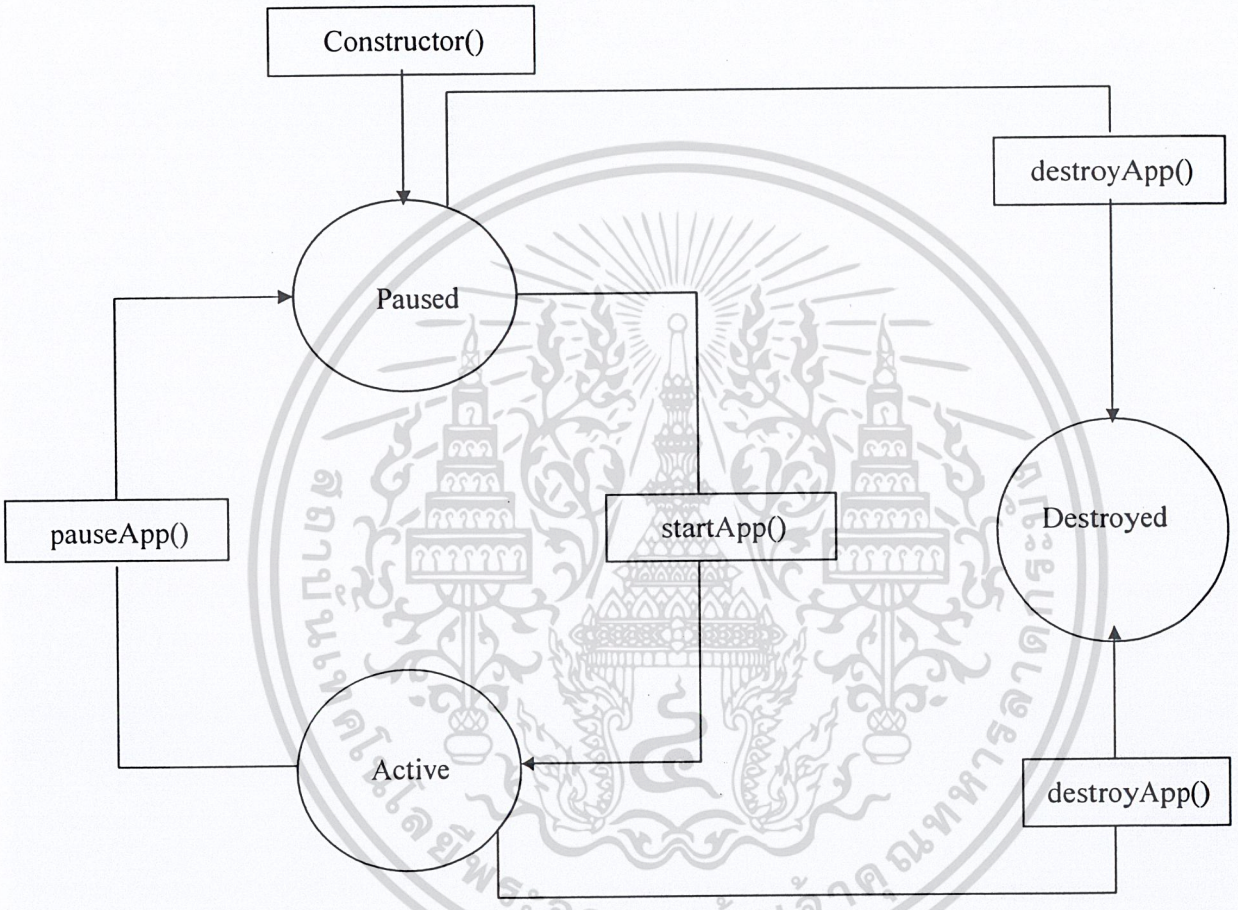
MIDlet หมายถึงแอพพลิเคชั่นบน MIDP MIDlet มีส่วนคล้ายกับจาวาแอปเพล็ต แม้จะไม่มีเมธ็อด main() แต่ MIDlet ก็นำคลาส java.microedtion.midlet.MIDlet ตลอดจนเมธ็อด startApp() pauseApp() และ destroyApp() เข้ามาใช้ นอกจากนี้ MIDlet ยังกำหนด constructor แบบ public ที่ไม่มีอาร์กิวเมนต์ใดๆอีกด้วย เราสามารถให้คำนิยามของคลาส javax.microedtion.midlet.MIDlet ได้ดังนี้

```
Public abstract คลาส MIDlet extends Object {
        protected MIDlet()
        protected abstract void startApp() throws MIDletStateChangeException
        protected abstract void pauseApp()
        protected abstract void destroyApp(boolean unconditional)
                throws MIDletStateChangeException
        public final String getAppProperty(String key)
        public final void notifyDestroyed()
        public final void notifyPaused()
        public final String getAppProperty(String key)
        public final void resumeRequest()
}
```

คลาส MIDlet ระบุเมธ็อดที่สามารถเรียกใช้โดยซอฟต์แวร์จัดการแอพพลิเคชั่น (AMS) เพื่อสั่งให้แอพพลิเคชั่น MIDlet เริ่มต้นและหยุดทำงาน

3.5.1 วงจรการทำงานของ MIDlet การกระทำของ MIDlet ประกอบไปด้วย 3 สถานะ คือกำลังทำงาน หยุดชั่วคราวและถูกทำลาย อาศัยซอฟต์แวร์จัดการแอพพลิเคชั่น เป็นตัวควบคุมการเปลี่ยนสถานะหนึ่งไปยังอีกสถานะหนึ่งควบคุมด้วยเมธ็อด startApp() pauseApp() และ destroyApp() ที่มาพร้อมกับ MIDlet จากรูปที่ 3.1 แสดงให้เห็นจุดเปลี่ยนระหว่างสถานะทั้ง 3 โดยการเรียกใช้เมธ็อดข้างต้น เมื่อ MIDlet พร้อมสั่งกระทำการ ซอฟต์แวร์จัดการแอพพลิเคชั่นจะสร้างตัวอย่าง MIDlet ขึ้นมาก่อนโดยใช้ constructor แบบ public ที่ไม่มีอาร์กิวเมนต์ใดๆ โดย MIDlet จะอยู่ในสถานะหยุดชั่วคราว จากนั้นซอฟต์แวร์จัดการแอพพลิเคชั่นจะเรียกเมธ็อด startApp() ขึ้นมา และ MIdlet จะเข้าสู่สถานะกำลังทำงาน

เปิดรับทรัพยากรที่ต้องการและเริ่มต้นการทำงาน ในสถานะนี้ MIDlet จะทำงานและดึงทรัพยากรที่ต้องการไว้ใช้งาน เมื่อซอฟต์แวร์จัดการแอพพลิเคชั่นไม่ต้องการให้ MIDlet ทำงานต่อไป ก็จะเรียกเมธ็อด pauseApp() จากนั้น MIDlet จะหยุดทำงานและเข้าสู่สถานะหยุดชั่วคราว คืนทรัพยากรที่ดึงมาใช้งานและเข้าสู่ภาวะไม่ทำงาน MIDlet สามารถกลับไปอยู่ที่สถานะกำลังทำงาน ได้เมื่อซอฟต์แวร์จัดการแอพพลิเคชั่นเรียกเมธ็อด startApp() ขึ้นมา



รูป 3.1 วงจรการทำงานของ MIDlet

ท้ายที่สุดเมื่อซอฟต์แวร์จัดการแอพพลิเคชั่นไม่ต้องการเรียกใช้งาน MIDlet หรือต้องการเคลียร์หน่วยความจำเพื่อให้โปรแกรมอื่นได้ใช้งาน ก็จะส่งสัญญาณเตรียมทำลาย MIDlet ทิ้งด้วยการเรียกเมธ็อด destroyApp() และเข้าสู่สถานะถูกทำลาย ซึ่ง MIDlet จะปล่อยทรัพยากรทั้งหมด ทำการจัดเก็บข้อมูลถาวรต่างๆ และหยุดการทำงานทั้งหมด หาก MIDlet อยู่ระหว่างสั่งกระทำขั้นตอนสำคัญอยู่ ก็อาจร้องขอไม่เข้าสู่สถานะถูกทำลายได้โดยเรียกใช้ MIDletStateChangeException อย่างไรก็ดีซอฟต์แวร์จัดการแอพพลิเคชั่นอาจปฏิเสธหรือยินยอมตามคำร้องขอนี้ก็ได้ ตัวแปรบูลีน unconditional ในลายเซ็นของเมธ็อด destroyApp() จะเป็นตัวกำกับว่าการร้องขอนี้สมควรหรือไม่ ถ้าตัวบ่งชี้มีค่าเท่ากับ true การร้องขอ

เป็นอันตกไป ตรงกันข้ามหากมีค่าเท่ากับ false ก็จะได้รับการตอบรับและจะเรียกเมธ็อด destroyApp() ขึ้นมาใหม่ในคราวต่อไป หาก MIDlet ต้องการเข้าสู่สถานะหยุดช่วงคราวหรือถูกทำลายด้วยตัวเอง ก็สามารถทำได้โดยเรียกเมธ็อด notifyPause() หรือ notifyDestroy() ตามลำดับ โดยเมธ็อดทั้ง 2 จะแจ้งให้ซอฟต์แวร์จัดการแอพพลิเคชั่นทราบว่า MIDlet ได้เข้าสู่สถานะหยุดชั่วคราว/ถูกทำลายแล้ว ในกรณีนี้ซอฟต์แวร์จัดการแอพพลิเคชั่นจะไม่เรียกเมธ็อด pauseApp() หรือ destroyApp() อีก MIDlet จะเข้าสู่สถานะหยุดทำงานได้ก็ต่อเมื่อผ่านสถานการณ์ทำงานมาก่อนแล้ว ในทางตรงกันข้าม หากต้องการเข้าสู่สถานะถูกทำลาย สามารถเข้าได้โดยตรงจากสถานะหยุดทำงานชั่วคราวและสถานะกำลังทำงาน นอกจากนี้ยังสามารถเข้าสู่สถานะหยุดการทำงานได้ในขณะที่กำลังทำงาน หรือมเอได้รับคำสั่งจากซอฟต์แวร์จัดการแอพพลิเคชั่น

3.6 ซอฟต์แวร์จัดการแอพพลิเคชั่น

ซอฟต์แวร์แอพพลิเคชั่นที่มาพร้อมกับ MIDP ทำหน้าที่ควบคุมการติดตั้ง สั่งกรทำการ และลบ MIDlet บางครั้งเราเรียกซอฟต์แวร์จัดการแอพพลิเคชั่นว่าเป็น ซอฟต์แวร์จัดการ MIDlet (MIDlet Management Software) หรือ โปรแกรมบริหารจาวาแอพพลิเคชั่น (Java Application Manager)

3.7 ข้อจำกัด

3.7.1 ข้อจำกัดในการจัดการข้อผิดพลาด CLDC นั้นรองรับความผิดพลาดในการทำงานในวงจำกัด โดยกำหนดคลาสแสดงข้อผิดพลาดไว้เพียง 2 คลาส คือ java.lang.VirtualMachineError และ java.lang.OutOfMemoryError ขณะที่ดึงคลาสส่วนใหญ่ออกไป เนื่องจาก

3.7.1.1 ในระบบอุปกรณ์ฝังตัว การคืนสู่ภาวะทำงานผิดพลาด เป็นความสามารถเฉพาะบางอุปกรณ์เท่านั้น ไม่ควรคาดหวังให้ผู้เขียนแอพพลิเคชั่นจัดการกับข้อบกพร่องเฉพาะอุปกรณ์เหล่านี้

3.7.1.2 ความผิดปรกติประเภท Error มักไม่สามารถแก้ไขได้ การบรรจุความสามารถในการจัดการข้อผิดพลาดอย่างเต็มรูปแบบทำให้ต้องเสียค่าใช้จ่ายมาก และยังต้องกำหนดลำดับความสำคัญไว้สูงทั้งที่อุปกรณ์ CLDC มีทรัพยากรค่อนข้างจำกัด

3.7.2 ข้อจำกัดของ KVM เนื่องจาก KVM เป็นเลเยอร์ Java Virtual Machine ที่รองรับไลบรารี่ของ CLDC คุณสมบัติใดที่ CLDC ไม่สนับสนุน ก็จะถูกดึงออกจาก KVM ด้วยเช่นกัน อาทิ ตัวเลขทศนิยม finalization และการจัดการข้อผิดพลาดเฉพาะอุปกรณ์

3.7.2.1 ไม่สนับสนุน Java Native Interface (JNI) ด้วยสาเหตุ 2 ประการคือในแบบจำลองความปลอดภัยของ CLDC ผู้เขียนแอพพลิเคชั่นไม่สามารถดาว์นโหลดไลบรารี่ใหม่ๆที่ยังมีฟังก์ชั่นเดิมของอุปกรณ์อยู่ และไม่สามารถเข้าถึงฟังก์ชั่นเดิมของอุปกรณ์ที่ไม่มีในไลบรารี่ของจาวาได้ นอกจากนี้ การนำ JNI มาใช้ยังเพิ่มภาระให้กับหน่วยความจำบนอุปกรณ์ CLDC

3.7.2.2 ไม่สนับสนุนโปรแกรมบรรจุคลาสที่ผู้ใช้งานกำหนดขึ้นเอง และโปรแกรมบรรจุคลาสใน KVM ก็ไม่อุญาติให้ผู้ใช้งานลบล้าง เขียนทับ หรือกำหนดค่าใหม่ด้วย

3.7.2.3 ไม่สนับสนุน Reflection , RMI และ การเรียงลำดับอ็อบเจกต์ ทำให้โปรแกรม CLDC ไม่สามารถตรวจสอบเนื้อหาในคลาสออบเจ็กต์ เมธ็อด ฯลฯ

3.7.2.4 ไม่สนับสนุน Thread Group หรือ Daemon คือ KVM สนับสนุนแอพพลิเคชั่นแบบทำงานคู่ขนาน (Multi-Thread) แต่ไม่รองรับกลุ่มงานย่อย (Thread Object) หรืองานเบื้องหลัง (Daemon Thread)

3.8 ระบบความปลอดภัยใน J2ME

วัตถุประสงค์สำคัญของการรักษาความปลอดภัยบนระบบคอมพิวเตอร์คือ การป้องกันทรัพยากรของระบบจากการเจาะเข้ามาทั้งที่ไม่ประสงค์ดีและโดยไม่ตั้งใจ การดาว์นโหลดเนื้อหาและแอพพลิเคชั่นตามข้อกำหนดของ CLDC ระบุไว้ ทำให้การรักษาความปลอดภัยบนระบบเครือข่ายทวีความสำคัญมากขึ้น จาวานั้นออกแบบมาโดยคำนึงถึงความปลอดภัยมาตั้งแต่ต้น โดย J2SE เองมีคุณสมบัติด้านความปลอดภัยที่หลากหลาย เช่น การตรวจไบต์โค้ด และโปรแกรมจัดการความปลอดภัย อย่างไรก็ดี เนื่องจากรหัสที่ J2SE ใช้ในการรักษาความปลอดภัยมีขนาดใหญ่เกินกว่าหน่วยความจำของอุปกรณ์ไร้สายที่มีทรัพยากรจำกัด จึงต้องลดทอนคุณสมบัติดังกล่าวลงให้เหมาะสมกับความต้องการระบบของอุปกรณ์ไร้สาย

3.9 การเวอริฟายและฟรีเวอริฟายไฟล์คลาส

เนื่องจากทรัพยากรที่มีจำกัดบนอุปกรณ์ไร้สายการเวอริฟายคลาสของ KVM จึงต้องแยกกันกระทำ โดยบางส่วนกระทำนอกตัวอุปกรณ์นั้น เรียกว่า พรีเวอริฟิเคชั่น

3.10 แบบจำลอง Sand Box

J2ME ได้นำแบบจำลองรักษาความปลอดภัยมาจาก J2SE หลักการของแบบจำลอง Sand Box ใน J2ME คือ ให้แอพพลิเคชั่นที่เขียนด้วยจาวาทำงานในสภาพแวดล้อมปิด ซึ่งแอพพลิเคชั่นสามารถเข้าถึงเฉพาะ API ที่กำหนดโดยคอนฟิกูเรชั่น โพรไฟล์ และคลาสเปิดที่อุปกรณ์นั้นได้รับอนุญาตให้ใช้ นอกจากนี้ แบบจำลอง Sand Box ยังหมายถึง

1. จาวาคลาสได้รับเวอริฟายและรับรองแล้วว่าเป็นจาวาแอพพลิเคชั่นที่ถูกต้อง

2. นักพัฒนาแอพพลิเคชั่นมีชุดจาวา API จำนวนจำกัดซึ่งกำหนดไว้ล่วงหน้าโดยโพรไฟล์ CLDC และคลาสเปิดที่ได้รับอนุญาตให้ใช้

3. การดาว์นโหลดและการจัดการจาวาแอพพลิเคชั่นบนอุปกรณ์ เกิดขึ้นในระดับโค้ดพื้นฐานที่อยู่ภายใน Virtual Machine และไม่มีโปรแกรมบรรจุคลาสที่ผู้ใช้งานกำหนดขึ้นเอง เพื่อป้องกันไม่ให้ผู้เขียนโปรแกรมเขียนทับกลไกมาตรฐานการโหลดคลาสของ Virtual Machine

4. ชุดฟังก์ชั่นเดิมของอุปกรณ์ที่เข้าถึง Virtual Machine ถูกปิดการใช้งาน ซึ่งหมายความว่านักพัฒนาแอพพลิเคชั่นไม่สามารถดาว์นโหลดไลบรารี่ใหม่ๆที่มีฟังก์ชั่นเดิมของอุปกรณ์มาใช้ได้ หรือไม่สามารถเข้าถึงฟังก์ชั่นเดิมของอุปกรณ์ซึ่งไม่มีในไลบรารี่ของจาวาที่กำหนดโดย CLDC Profile หรือคลาสเปิดที่ได้รับอนุญาตให้ใช้

บทที่ 4
การพัฒนาแอพพลิเคชั่นสำหรับอุปกรณ์ไร้สายด้วยจาวา

4.1 คอมโพเน็นท์หลักของส่วนติดต่อกับผู้ใช้

เจทูเอ็มอี สามารถทำงานบนอุปกรณ์ที่หลากหลาย ตั้งแต่สมาร์ทการ์ดและโทรศัพท์เคลื่อนที่ ไปจนถึงอุปกรณ์ set-top-box สำหรับโทรทัศน์และเครื่องใช้ไฟฟ้าในบ้าน อุปกรณ์เหล่านี้มีความสามารถในการแสดงผลและทรัพยากรที่แตกต่างกันค่อนข้างมาก บริษัทซัน ไมโครซิสเต็มส์ตระหนักถึงปัญหานี้จึงไม่ได้กำหนดคลาสส่วนติดต่อกับผู้ใช้ซึ่งสมบูรณ์แบบใน เจทูเอ็มอี หากแต่กำหนด API ของส่วนติดต่อกับผู้ใช้ในลักษณะโพรไฟล์ของอุปกรณ์เคลื่อนที่ (Mobile Information Device Profile : MIDP) หรือถ้าเป็น API ส่วนติดต่อกับผู้ใช้สำหรับพีดีเอ ก็จะระบุไว้ในโพรไฟล์ของพีดีเอ (Personal Digital Assistant Profile :PDAP) เป็นต้น

API ของส่วนติดต่อกับผู้ใช้ในเจทูเอสอีกำหนดไว้ในชุดเครื่องมือ Abstract Window Toolkit (AWT) ซึ่งออกแบบสำหรับเครื่องคอมพิวเตอร์ตั้งโต๊ะที่มีอุปกรณ์การชี้ (Pointer Device) จอภาพขนาดใหญ่พร้อมหน่วยความจำที่พอเพียง ขณะที่หน้าจอแสดงผลไร้สายมีขนาดเล็กกว่า และใช้แป้นกดเพื่อป้อนข้อมูลเข้าแทนที่จะเป็นตัวชี้ ดังนั้นกลุ่มผู้เชี่ยวชาญการออกแบบ MIDP จึงได้กำหนด API ของส่วนติดต่อกับผู้ใช้ขึ้นมาใหม่ทั้งหมดโดยไม่ได้อ้างอิงซับเซ็ตของ AWT เลยซึ่ง API ของส่วนติดต่อกับผู้ใช้เหล่านี้กำหนดไว้ในแพ็กเกจ javax.microedition.lcdui

API ของส่วนติดต่อกับผู้ใช้ใน MIDP มี 2 กลุ่มหลักๆ ด้วยกันคือ API ระดับสูงและระดับล่าง API ระดับสูงออกแบบให้เป็น Abstract เน้นความยืดหยุ่นในการโยกย้ายไปใช้งานในระบบอื่น แอพพลิเคชั่นที่ใช้ API ระดับสูงจะควบคุมวิธีแสดงผลได้น้อยมากและเข้าถึงได้เฉพาะอีเวนต์ระดับสูงเท่านั้น ในทางตรงกันข้ามการใช้ API ระดับล่างจะสามารถควบคุมการแสดงผลได้เต็มที่ สามารถเข้าถึงอุปกรณ์ป้อนข้อมูล และจัดการอีเวนต์พื้นฐานที่เกิดจากการตอบโต้ของผู้ใช้ได้ อย่างไรก็ดีโปรแกรมที่ใช้ API ระดับล่างนี้จะขาดความยืดหยุ่น ยึดติดกับอุปกรณ์มากจนเกินไป โดยจะต้องพิจารณาถึงข้อดีข้อเสียในการออกแบบโปรแกรมด้วย

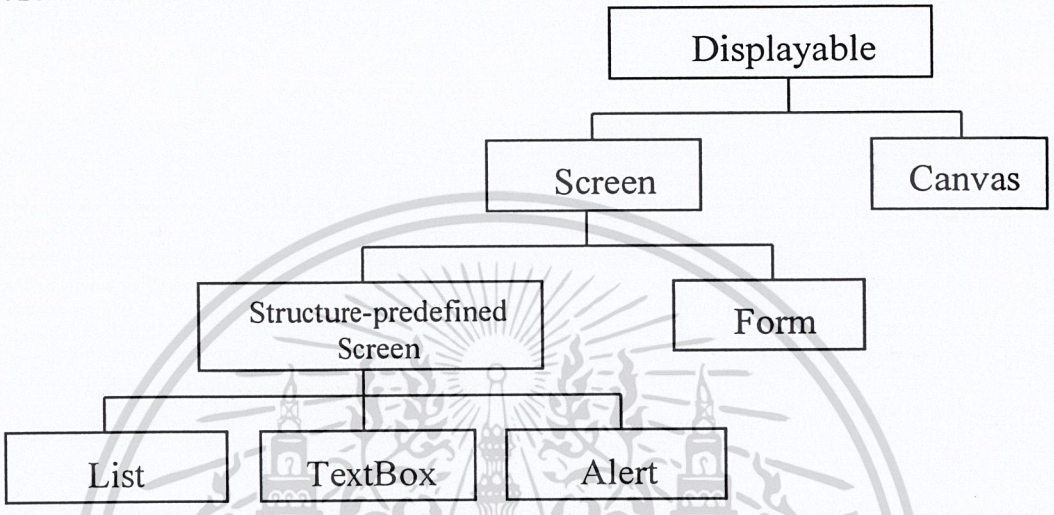
4.1.1 Displayable และ Display ฟังก์ชั่นหลักอย่างหนึ่งของส่วนติดต่อกับผู้ใช้ ก็คือ การแสดงข้อมูลบนหน้าจอใน J2ME MIDP ออบเจ็กต์ Displayable จะเก็บข้อมูลที่ต้องการนำเสนอ และออบเจ็กต์ Display จะนำออบเจ็กต์ Displayable นั้นไปแสดงบนจอให้ผู้ใช้เห็น

4.1.1.1 Displayable เป็นคลาสหลักของส่วนติดต่อกับผู้ใช้ของ MIDP สามารถเก็บภาพกราฟิกที่สร้างเฉพาะอุปกรณ์และนำไปแสดงบนจอ

MIDP กำหนด API ของส่วนติดต่อกับผู้ใช้ออกเป็น 2 กลุ่มคือ API ระดับสูงและระดับล่าง คลาส Screen เป็นคลาสย่อยของ Displayable ที่ใช้ API ระดับสูง ส่วนคลาส Canvas เป็นคลาสย่อยของ Displayable ที่ใช้ API ระดับล่าง

ขนาดและรูปร่างของหน้าจออุปกรณ์จะแตกต่างกันออกไป เนื่องจาก API ระดับสูงจะจัดการกับความแตกต่างของขนาดหน้าจอ ดังนั้น ในการเขียนแอพพลิเคชั่นจึงไม่ต้องพะวงเรื่องความแตกต่างของขนาดหน้าจอมากนัก แต่หากเป็นแอพพลิเคชั่นที่ใช้ API ระดับล่างจะต้องระวังเรื่องความแตกต่างของหน้าจอแสดงผลเหล่านี้

Displayable ทุกตัวต้องรีจิสเตอร์ Command และ CommandListener เพื่อที่จะสามารถรับอีเวนต์ของคอมมานด์ได้



รูปที่ 4.1 โครงสร้างลำดับขั้นของคลาส Displayable

4.1.1.1.1 Screen เป็นซับคลาสของ Displayable ซึ่งนำ API ระดับสูงมาใช้ประกอบด้วยซุปเปอร์คลาส Alert , Form , List และ TextBox อาจมีชื่อหรือแถบตัวอักษรวิ่ง (ticker tape) กำกับหรือจะไม่มีก็ได้

Screen เป็นคลาสแบบ Abstract เมธ็อดซึ่งกำหนดวิธีการแสดงผลและการโต้ตอบกับเนื้อหาในออบเจ็กต์ก็เป็นเมธ็อดแบบ Abstract ด้วย นำมาใช้โดยซับคลาสของ Screen มีเพียงเมธ็อดสำหรับปรับปรุงหน้าจอเวลาเมื่อต้องการแสดงผลเท่านั้นที่อยู่ในคลาส Screen อย่างไรก็ดี ไม่สามารถเพิ่มเติมอะไรในคลาส Screen ได้เอง เนื่องจากใช้เมธ็อดแบบ Abstract ซึ่งไม่สามารถเข้าถึงได้จากภายนอกแพ็กเกจ javax.microedition.lcdui

4.1.1.1.2 Canvas เป็นซับคลาสของ Displayable ซึ่งใช้ API ระดับล่างแอพพลิเคชั่นสามารถควบคุมได้ว่าจะแสดงผลอะไรและอย่างไรบ้าง และยังสามารถเข้าถึงอีเวนต์ระดับล่างเช่น การกดปุ่ม ได้ด้วย Canvas เป็นคลาส Abstract แอพพลิเคชั่นต้องสร้างซับคลาสย่อยเพื่อใช้งาน

4.1.1.2 Display เป็นตัวจัดการการแสดงผล และใช้เมธ็อดเหล่านี้สำหรับอ่านค่าพรอพเพอร์ตี้ของอุปกรณ์และนำออบเจ็กต์ไปแสดงผลบนหน้าจอ

ในโปรแกรม MIDlet หนึ่งๆ จะมีออบเจ็กต์ของ Display ได้เพียงตัวเดียวเท่านั้น แอพพลิเคชั่นสามารถอ้างถึงออบเจ็กต์ Display ได้ด้วยเมธ็อด getDisplay() และสามรถเรียกเมธ็อดนี้ได้ทุก

เมื่อตั้งแต่เริ่มเรียกใช้ MIDlet.startApp() ไปจนถึง MIDlet.destroyApp() ออบเจ็กต์ Display ทั้งหมดที่ได้จาก getDisplay() จะยังคงอยู่ตลอด จนกว่าจะเสร็จสิ้นโปรแกรม MIDlet

Displayable ปัจจุบัน แอพพลิเคชั่นจะมีออบเจ็กต์ Displayable ที่จะแสดงผลบนอุปกรณ์หรือมีการทำงานโต้ตอบกับผู้ใช้ได้เพียงออบเจ็กต์เดียวเท่านั้น โดยเรียก Displayable นี้ว่า Displayable ปัจจุบัน ผู้ใช้จะป้อนข้อมูลหรือโต้ตอบได้เฉพาะกับ Displayable ปัจจุบันเท่านั้น คลาส Displayable มีเมธ็อด getCurrent() สำหรับดึง Displayable ปัจจุบัน และมีเมธ็อดไว้สำหรับแสดงผลคือ setCurrent(Displayable nextDisplayable)

แอพพลิเคชั่น สามารถควบคุม Displayable ปัจจุบัน และ เรียก เม ธ็อด Display.setCurrent() ได้ตลอดเวลา เธรดของ MIDlet สามารถกำหนดหรือดึงค่า Displayable ปัจจุบันด้วย เมธ็อด Display.setCurrent() และ Display.getCurrent()

แอพพลิเคชั่นหน้าฉากและหลังฉาก แม้สามารถสั่งให้แอพพลิเคชั่นหลายตัวทำงานพร้อมกันบนอุปกรณ์หนึ่งๆ อุปกรณ์ก็จะแสดงผลได้เฉพาะ Displayable ปัจจุบันเท่านั้น ดังนั้น แอพพลิเคชั่นใดที่มี Displayable ปัจจุบันแสดงผลอยู่บนจอจะเรียกว่าเป็นแอพพลิเคชั่นหน้าฉาก (foreground) ซึ่งสามารถรับอีเวนต์การป้อนข้อมูลเข้าจากผู้ใช้ได้ แอพพลิเคชั่นที่เหลือซึ่งไม่ได้แสดงผลจะเรียกว่าแอพพลิเคชั่นหลังฉาก (background)

MIDlet สามารถวางตัวเองให้อยู่หลังฉากได้โดยใช้ Display.setCurrent(null) แต่การใช้เมธ็อดนี้ไม่ได้กำหนด Displayable ปัจจุบันให้เป็น null จริงๆ สมมติว่า Displayable ปัจจุบันคือ cD หลังจากแอพพลิเคชั่นเรียก Display.setCurrent(null) แล้ว การเรียกเมธ็อด getCurrent() ก็ยังได้ค่าเป็น cD แอพพลิเคชั่น จะกลับคืนสู่หน้าฉากโดยใช้ Display.setCurrent(Display.getCurrent())

แม้ว่า Displayable ปัจจุบันของแอพพลิเคชั่นจะไม่ได้วาดอะไรบนจอก็ตาม แอพพลิเคชั่นก็ยังรับรู้ว่า Displayable ใดเป็น Displayable ปัจจุบัน ซึ่งสำคัญมากแม้แต่กับแอพพลิเคชั่นหลังฉาก เนื่องจากเมื่อแอพพลิเคชั่นหลังฉากกลายเป็นแอพพลิเคชั่นหน้าฉาก ก็จะสามารถแสดงผล Displayable ที่ถูกต้องได้

โปรแกรม MIDlet แต่ละตัวจะมี Displayable ปัจจุบันของตัวเอง เมธ็อด getCurrent() จะให้ Displayable ปัจจุบันโดยใช้เมธ็อด setCurrent() ก็ไม่มีผลอะไรกับ Displayable ปัจจุบันของ MIDlet อื่นๆ

System Screen โดยปรกติแล้ว จะสามารถมองเห็นการแสดงผลบนหน้าจอปัจจุบันของ MIDlet ที่ทำงานหน้าฉากได้ อย่างไรก็ตามในบางสถานการณ์ระบบอาจจะสร้างหน้าจอชั่วคราขึ้นมาบังหน้าจอปัจจุบันของแอพพลิเคชั่นเอาไว้ คือ หน้าจอของระบบ(system screen) โดยจะแสดงขึ้นมาเมื่อระบบต้องการแสดงเมนูเพิ่มเติม นอกเหนือไปจากการแก้ไขฟิลด์ข้อความต่างๆใน Form

แม้ว่าหน้าจอของระบบ จะโผล่ขึ้นมาบังหน้าจอของแอพพลิเคชั่น Displayable ปัจจุบันก็ยังเป็นอันเดิมไม่เปลี่ยนแปลง การเรียกใช้เมธ็อด getCurrent() ในขณะที่มองเห็นหน้าจอของระบบ ก็ยังได้ค่า Displayable ปัจจุบันจองแอพพลิเคชั่น และไม่ใช่หน้าจอของระบบ

ถ้าจะดูว่า Displayable ปัจจุบัน ถูกบังด้วยหน้าจอของระบบหรือไม่ สามารถตรวจสอบ ได้จาก currentDisplayable.isShown()

4.1.2 Image คลาส Image ใช้เก็บข้อมูลรูปกราฟิก ออบเจ็กต์ Image จะอยู่ในหน่วยความจำที่แยกจากส่วนแสดงผล ออบเจ็กต์ Image สามารถวาดลงบน Canvas หรือวางใน Form , Alert , List หรือ ChoiceGroup ก็ได้

Image อาจเป็นรูปภาพที่แก้ไขได้ หรือแก้ไขไม่ได้ ขึ้นอยู่กับวิธีการสร้าง

4.1.2.1 รูปที่แก้ไขไม่ได้ มักมีที่มาจากการโหลดภาพจากไฟล์ จากแหล่งอื่นๆ ที่ใส่มาในแพ็กเกจ หรือจากเครือข่าย เมื่อสร้างรูปเสร็จแล้ว จะไม่สามารถแก้ไขได้อีก รูปที่ใส่ใน Alert , Choice หรือ Item จะต้องเป็นรูปที่แก้ไขไม่ได้เท่านั้น

4.1.2.2 รูปที่แก้ไขได้ จะสร้างในหน่วยความจำที่อยู่นอกจอ เสมือนกับว่าเป็นหน้าจอหรือ Canvas อีกอันหนึ่ง แอพพลิเคชั่นสามารถวาดรูปที่แก้ไขนี้ได้จากที่สร้างขึ้นมาแล้ว ด้วยเมธ็อด Image.createImage(int width,int height) โดยรูปที่แก้ไขได้จะใช้ในเทคนิคแบบบัฟเฟอร์ 2 ชั้น

พรอพเพอร์ตี้ของรูป ออบเจ็กต์ของ image จะมีพรอพเพอร์ตี้ 3 แบบด้วยกัน คือ height width และ isMutable แอพพลิเคชั่นสามารถอ่านค่าพรอพเพอร์ตี้ได้โดยใช้เมธ็อด getHeight() getWidth() และ isMutable() ตามลำดับ ถ้ารูปนั้นแก้ไขได้ แอพพลิเคชั่นจะสามารสร้างออบเจ็กต์ Graphics เพื่อสร้างรูปโดยใช้ getGraphics()

4.1.3 อีเวนต์และการจัดการอีเวนต์ ส่วนติดต่อกับผู้ใช้ประกอบด้วย 2 ส่วนหลักคือ หน้าจอที่ทำหน้าที่แสดงผล และตอบสนองการโต้ตอบของผู้ใช้ ส่วนติดต่อกับผู้ใช้ในอีเวนต์จะทำหน้าที่ประมวลผลอีเวนต์ที่เกิดขึ้น อีเวนต์และตัวจัดการอีเวนต์มีทั้งระดับสูงและระดับล่างเช่นเดียวกันกับ API ของ Displayable

4.2 การใช้ API ระดับสูงในการพัฒนาส่วนติดต่อกับผู้ใช้

API ระดับสูงเน้นความสามารถในการโยกย้ายไปใช้กับอุปกรณ์อื่นๆ ซับคลาสของ Screen มี 2 กลุ่ม กลุ่มแรก ได้แก่ List TextBox และ Alert ซึ่งมีโครงสร้างที่กำหนดไว้ล่วงหน้าแล้ว กับกลุ่มที่สอง ได้แก่คลาส Form ซึ่งกำหนดโครงสร้างโดยแอพพลิเคชั่น

4.2.1 List และ Choice เมื่อเริ่มการทำงาน แอพพลิเคชั่นมักจะแสดงเมนูซึ่งประกอบด้วยฟังก์ชั่นต่างๆ ให้คุณได้เลือกตามต้องการ List และ ChoiceGroup ต่างแสดงรายการทางเลือก โดยทั้งคู่ใช้อินเตอร์เฟส Choice ที่กำหนดไว้ในแพ็กเกจ javax.microedition.lcdui

4.2.1.1 List เป็น Screen ซึ่งมีโครงสร้างที่กำหนดไว้แล้วโดยใช้อินเตอร์เฟส Choice API ของแพ็กเกจ javax.microedition.lcdui กำหนด constructor ของ list ไว้ 2 แบบ โดยตัวแรกจะสร้าง List เปล่าๆขึ้นมา ซึ่งสามารถเพิ่มรายการของ choice เข้าไปได้ทีหลัง constructor ตัวที่สองจะสร้าง List ที่มีค่าเริ่มต้นอยู่ข้างใน อาร์เรย์ stringElements ต้องไม่เป็น null และสมาชิกทุกตัวในอาร์เรย์ต้องไม่เป็น nul ด้วยเช่นกัน ความยาวของอาร์เรย์ stringElements จะกำหนดจำนวนรายการใน List อาร์เรย์

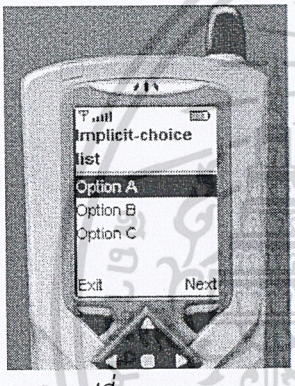
imageElements อาจมีค่าเป็น null ซึ่งหมายความว่ารายการใน List นั้นไม่มีภาพอยู่ ถ้าอาร์เรย์ imageElements ไม่เป็น null ก็จะต้องมีความยาวเท่ากับอาร์เรย์ stringElements รายการแต่ละรายการในอาร์เรย์ imageElements อาจจะเป็น null ได้ซึ่งหมายความว่าไม่มีภาพในรายการของ List ที่ตรงกัน

4.2.1.2 Choice ออบเจ็กต์ Choice มี 3 ประเภทได้แก่ implicit-choice (ใช้ได้กับ List เท่านั้น) exclusive-choice และ multiple-choice

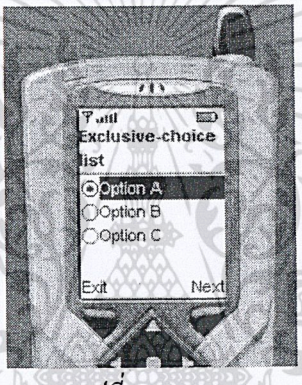
4.2.1.2.1 Exclusive-choice จะต้องเลือก รายการใดรายการหนึ่ง ในแต่ละครั้ง เว้นเสียแต่ว่าไม่มีรายการใดๆให้เลือก

4.2.1.2.2 Implicit-choice เป็นกรณีพิเศษของ exclusive-choice เมื่อรายการที่อยู่ในโฟกัสถูกเลือกโดยอ้อมเมื่อเริ่มต้น Command โดย choice ประเภทนี้ใช้ได้กับ List เท่านั้น

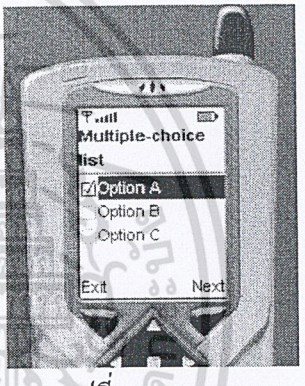
4.2.1.2.3 Multiple-choice สามารถเลือกหลายรายการพร้อมกันได้ในแต่ละครั้ง หรือจะไม่เลือกเลยก็ได้



รูปที่ 4.2 List แบบ Implicit-choice



รูปที่ 4.3 List แบบ exclusive-choice



รูปที่ 4.4 List แบบ multiple-choice

ออบเจ็กต์ Choice จะแสดงรายการทางเลือกแบบต่างๆ ให้ผู้ใช้เลือก แต่ละรายการประกอบด้วยสตริงข้อความและรูปซึ่งอาจจะมีหรือไม่มีก็ได้

จำนวนรายการในออบเจ็กต์ Choice สามารถดูได้โดยใช้เมธ็อด size() แต่ละรายการสามารถอ้างถึงได้ตามดัชนีซึ่งเป็นตัวเลขจำนวนเต็มมีค่าตั้งแต่ 0 จนถึง size() – 1 โดยที่ 0 อ้างถึงรายการแรกและ size() – 1 อ้างถึงรายการสุดท้าย

ภาพในรายการ Choice อาจเว้นว่างได้ถ้าแอพพลิเคชั่นนั้นไม่ได้กำหนดไว้ หรือแม้จะกำหนดไว้ก็อาจเลือกไม่แสดงภาพเลย หากอุปกรณ์นั้นๆ ไม่สามารถแสดงผลได้ ถ้าเลือกที่จะแสดงภาพภาพจะติดอยู่กับสตริงข้อความและถือเสมือนว่าทั้งภาพและข้อความนั้นเป็นหน่วยเดียวกัน ภาพไม่สามารถเปลี่ยนได้ แอพพลิเคชั่นสามารถกำหนดภาพให้กับรายการ elementNum choice ได้โดย Void set(int elementNum, String stringPart, Image imagePart) และอ่านภาพจาก elementNum choice ได้โดย Image getImage(int elementNum)

4.2.2 TextBox เป็น Screen ที่มีโครงสร้างกำหนดไว้แล้ว ผู้ใช้งานสามารถป้อนหรือแก้ไขข้อความใน TextBix ได้ การสร้างออบเจ็กต์ TextBox ทำได้โดยใช้ constructor ดังนี้

TextBox(String title, String text, int maxSize, int constraints)

ข้อความใน TextBox จะเก็บเป็นอาร์เรย์ของ char ดังนั้นจึงมีขนาด (ความจุ) ในการเก็บกลุ่มตัวอักษรไว้ในออบเจ็กต์อย่างจำกัด โดยสามรถกำหนดขนาดสูงสุดของ TextBox ในขณะสร้างออบเจ็กต์ หรืออาจใช้เมธ็อด setMaxSize() กำหนดหลังจากที่สร้างออบเจ็กต์ TextBox แล้ว การดูขนาดความจุทำได้โดยใช้เมธ็อด getMaxSize() เนื่องจากอุปกรณ์ไร้สายมักมีหน่วยความจำจำกัด ขีดจำกัดสูงสุดที่แท้จริงขึ้นอยู่กับค่าสูงสุดที่ MIDP กำหนดไว้ ดังนั้นขนาดสูงสุดจริงของ TextBox จะจำกัดอยู่ภายในกรอบที่กำหนดโดย MIDP

การตรวจสอบขนาดความจุสูงสุดเกิดขึ้นเมื่อสร้างออบเจ็กต์ TextBox เมื่อผู้ใช้งานแก้ไขข้อความภายใน TextBox และเมื่อโปรแกรมเรียกใช้เมธ็อดเพื่อแก้ไขเนื้อหาของ TextBox เมื่อใดก็ตามที่ข้อความใน TextBox มีขนาดเกินกว่าที่กำหนดไว้ จะเกิด IllegalArgumentException ขึ้น

ข้อความใน TextBox อาจมีความยาวเกินกว่าจะแสดงผลได้หมดในครั้งเดียว ในกรณีนี้ผู้ใช้สามารถเลื่อนหน้าจอขึ้น-ลง เพื่อดูหรือแก้ไขข้อความส่วนต่างๆ การเลื่อนหน้าจอนี้ไม่ทำให้เกิดอีเวนต์ application-visible แต่อย่างใด

4.2.3 Alert เป็น Screen ที่มีโครงสร้างกำหนดไว้แล้ว ใช้สำหรับแสดงข้อมูล (ข้อความและภาพ) ให้ผู้ใช้และจะคอยเป็นเวลาครู่หนึ่ง (กำหนดโดย timeout) ก่อนที่จะแสดงหน้าจอถัดไป

4.2.4 Form และ Item

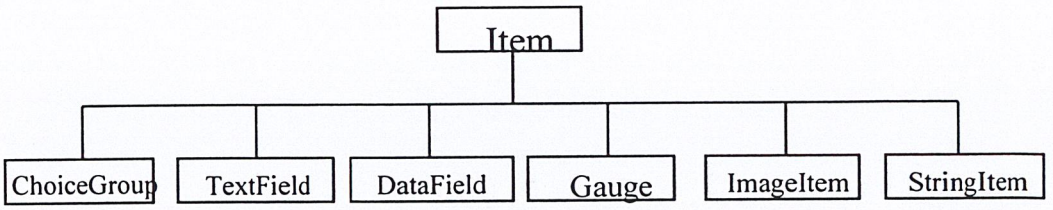
Form เป็นออบเจ็กต์ Screen ซึ่งสามารถบรรจุ Item ต่างๆเช่น ภาพ ข้อความ หรือ choice จำนวนเท่าใดก็ได้ การสร้างออบเจ็กต์ Form ทำได้ 2 แบบ คือ สร้าง Form เปล่าที่ไม่มี Item ใดๆโดยใช้ Form(String title) หรือสร้าง Form ที่มี Item เริ่มต้นโดยใช้ Form(String title,Item[] item)

สามารถดูจำนวนของ Item ใน Form ได้โดยใช้เมธ็อด size() การอ้างถึง Item ทำได้โดยใช้ดัชนีตัวเลข ซึ่งมีค่าตั้งแต่ 0 ถึง size() – 1 อ้างถึงตัวสุดท้าย

สามารถดึงออบเจ็กต์ Item จาก Form ได้โดยใช้ get(int itemNum)

Item หนึ่งจะอยู่ได้ใน Form เดียวเท่านั้น หากแอพพลิเคชั่นโปรแกรมพยายามใส่ Item ที่มีแล้วใน Form นี้หรือ ในฟอร์มอื่นๆ ลงไปในอีก Form หนึ่งก็จะเกิด IllegalStateException ขึ้น ดังนั้นจะต้องย้าย Item ออกจาก Form ที่อยู่เดิมก่อนจะสามารถใส่เข้าไปใน Form ใหม่ได้

4.2.4.1 Item เป็นซุปเปอร์คลาสสำหรับคอมโพเน็นต์ต่างๆที่สามารถใส่ลงไปใน Form ได้ แต่ละ Item จะมีป้ายข้อความกำกับ ซึ่งเป็นฟิลด์สตริงข้อความที่ติดอยู่กับ Item โดยปกติจะแสดงไว้ใกล้ๆ กับคอมโพเน็นต์ที่ปรากฏในหน้าจอ



รูปที่ 4.5 โครงสร้างลำดับขั้นของคลาส Item

4.2.4.1.1 StringItem ประกอบด้วยป้ายและสตริงข้อความที่แสดงผลอย่างเดียว ผู้ใช้ไม่สามารถแก้ไขค่าใน StringItem ได้โดยตรง ต้องทำผ่านแอพพลิเคชั่นเท่านั้นโดยใช้เมธ็อด setText() การสร้างออบเจ็กต์ของ StringItem ทำได้ดังนี้ StringItem(String label, String text)

4.2.4.1.2 ImageItem อาจมีออบเจ็กต์ Image ซึ่งสามารถเปลี่ยนแปลงได้ การสร้างออบเจ็กต์ ImageItem ทำได้ดังนี้ ImageItem(String label, Image img, int layout, String altText)

4.2.4.1.3 ChoiceGroup คือ Item ที่แสดงผลแบบ Choice มีลักษณะคล้ายกับ List แต่ว่าไม่มีแบบ implicit-choice การแสดงผลแบบ exclusive หรือ multiple-choice ของ ChoiceGroup จะเหมือนกับ List การสร้าง ChoiceGroup ทำได้ดังนี้ ChoiceGroup(String label, int choiceType)

4.2.4.1.4 Gauge เป็น Item ที่แสดงผลแบบกราฟแท่งโดยมีค่าระหว่าง 0 ถึง maxValue การสร้างออบเจ็กต์ Gauge ทำได้ดังนี้

Gauge(String label, □ oolean interactive, int maxValue, int initialValue)

4.2.4.1.5 DateField คือ Item ทีแก้ไขได้ ใช้สำหรับแสดงค่าวัน เดือน ปีและเวลา สามารถสร้างออบเจ็กต์ DateField โดยใช้ DateField(String label, int mode, TimeZone timeZone)

4.2.4.1.6 TextField เป็น Item ที่มีสตริงข้อความซึ่งผู้ใช้สามารถแก้ไขได้ การสร้างออบเจ็กต์ TextField ทำได้ดังนี้ TextField(String label, String text, int maxSize, int constraints)

ข้อแตกต่างระหว่าง TextField และ TextBox คือ TextBox เป็นซับคลาสของ Screen ดังนั้นจึงสามารถแสดงผลบนหน้าจอได้ทันที ทว่า TextField เป็นซับคลาสของ Item จะสามารถแสดงผลได้ก็ต่อเมื่อนำไปไว้ใน Form

4.2.4.2 Form เป็นคอมโพเน็นท์ที่สำคัญที่สุดในการพัฒนาส่วนติดต่อกับผู้ใช้สำหรับอุปกรณ์ MIDP เพราะช่วยให้ผู้พัฒนาสามารถสร้างหน้าจอได้อย่างยืดหยุ่น

ในการนำ MIDP มาใช้งาน ได้ระบุการวางโครงร่างของ Item ใน Form สำหรับอุปกรณ์ทั่วๆไปไว้ว่าต้องเป็นแนวตั้ง Item ที่โฟกัสได้เช่น TextField, DateField, Gauge หรือ ChoiceGroup จะขึ้นบรรทัดใหม่เสมอ สำหรับ StringItem และ ImageItem ซึ่งไม่เกี่ยวข้องกับการโต้ตอบของผู้ใช้จะวางในแนวนอน นอกเสียจากว่าสตริงหรือคำสั่ง Layout ของ ImageItem จะกำหนดให้ขึ้นบรรทัดใหม่ ข้อความใน StringItem หรือ ภาพใน ImageItem จะถูกตัดขึ้นบรรทัดใหม่(กรณีข้อความ) หรือถูกคลิปบางส่วนออก(กรณีภาพ) เพื่อให้พอดีกับความกว้างของหน้าจอ หากจำเป็นต้องเลื่อนจอ ทำได้เฉพาะในแนวตั้งเท่านั้น

4.2.5 อีเวนต์ระดับสูงและการจัดการอีเวนต์ โมเดลการจัดการอีเวนต์ระดับสูงเป็นแบบ delegation-based เหมือนกับใน AWT ของ J2SE ในโมเดลนี้ประกอบด้วย 2 ส่วนคือ แหล่งกำเนิดอีเวนต์และตัว event listener แหล่งกำเนิดอีเวนต์จะสร้างอีเวนต์แล้วส่งไปยัง event listener จากนั้นเมธ็อดที่จัดการอีเวนต์ของ listener ก็จะประมวลผลอีเวนต์นั้น

ออบเจ็กต์ของส่วนติดต่อกับผู้ใช้ใน MIDP จะใช้อีเวนต์ระดับสูง 2 ตัวคือ อีเวนต์ command และ ItemStateChanged ดังนั้นจึงมีอีเวนต์ listener 2 ตัวคือ commandListener และ ItemStateListener ตามลำดับ

4.2.5.1 Command เป็นโครงสร้างที่เก็บความหมายของแอกชั่นหรืออีเวนต์ แอกชั่นจริง (หรือการจัดการอีเวนต์) ซึ่งเกิดเมื่อมี command เกิดขึ้นได้กำหนดไว้ในอินเตอร์เฟสของ CommandListener เมื่อ command เริ่มต้นจะเกิดอีเวนต์แล้วส่งไปยังออบเจ็กต์ CommandListener ที่รีจิสเตอร์ไว้ ออบเจ็กต์ Command มีข้อมูล 3 อย่างคือ เลเบล , ชนิด และ ลำดับความสำคัญ

4.2.5.2 ItemStateChanged เมื่อ Form แสดงผลบนจอ ผู้ใช้สามารถเลื่อนโฟกัสจาก Item หนึ่งไปยังอีก Item หนึ่งภายใน Form ได้ โดยไม่เกินอีเวนต์ให้แอพพลิเคชั่นสามารถตรวจเห็นได้ แต่หากผู้ใช้แก้ไขเปลี่ยนแปลงสถานะของ Item ที่โต้ตอบได้ซึ่งอยู่ภายใน Form จะทำให้เกิด ItemStateChanged ขึ้น แต่ถ้าเป็นการเปลี่ยนแปลงค่า Item โดยแอพพลิเคชั่นด้วยเมธ็อด setString() ก็จะไม่ทำให้เกิดอีเวนต์ ItemStateChanged

4.3 การใช้ API ระดับล่างในการพัฒนาส่วนติดต่อกับผู้ใช้

API ระดับล่างจะช่วยให้แอพพลิเคชั่นสามารถควบคุมการแสดงผลบนจอและเข้าถึงอีเวนต์ระดับล่างได้ แต่ก็อาจทำให้แอพพลิเคชั่นที่พัฒนาด้วย API ระดับล่างที่นำคุณสมบัติเฉพาะอุปกรณ์มาใช้ขาดความยืดหยุ่น และไม่อาจย้ายไปใช้บนแพลตฟอร์มอื่นได้

API ระดับล่างประกอบด้วย 2 ส่วนคือ คลาส Canvas ซึ่งมีชุดคำสั่งทางกราฟิกและเป็นคลาสที่สร้างอีเวนต์ระดับล่างและคลาส Graphics ซึ่งใช้สร้างข้อความและรูปภาพหรือรูปร่างต่างๆ

4.3.1 พื้นฐานเกี่ยวกับ Canvas เป็นซับคลาสของ Displayable ซึ่งใช้ API ระดับล่าง คลาสนี้มีเมธ็อด paint สำหรับการวาดและเมธ็อดอื่นๆ สำหรับจัดการอีเวนต์ระดับล่างซึ่งแอพพลิเคชั่นสามารถเขียนใหม่ได้ เมธ็อด Paint() ในคลาส Canvas ถูกประกาศเป็น Abstract เช่นเดียวกับคลาส Canvas ดังนั้นแอพพลิเคชั่นจะต้องการขยายคลาส Canvas เพื่อใช้งานและจะต้องมีเมธ็อด Paint() เมื่อเป็นซับคลาสของ Canvas แต่เมธ็อดสำหรับจัดการอีเวนต์ระดับล่างไม่ได้ถูกประกาศเป็น Abstract จึงมีค่าว่าง(นั่นคือ จะไม่ทำงานใดๆ) ดังนั้น แอพพลิเคชั่นไม่จำเป็นต้องมีเมธ็อดการจัดการอีเวนต์ครบทั้งหมด หากแต่ต้องการเฉพาะอีเวนต์ที่แอพพลิเคชั่นสนใจเท่านั้น

ขนาดที่แท้จริงของพื้นที่แสดงผลจะแตกต่างกันไปตามอุปกรณ์แต่ละชนิด แอพพลิเคชั่นไม่ควรกำหนดขนาดที่ตายตัวแต่ควรเช็คขนาดของพื้นที่แสดงผลโดยใช้เมธ็อด getHeight() และ getWidth()

จุดประสงค์หลักในการใช้คลาส Canvas ก็เพื่อเข้าถึงและจัดการอีเวนต์ระดับล่าง แต่อีเวนต์ระดับสูงก็สามารถเกิดจากออบเจ็กต์ Canvas ได้ด้วยเช่นกัน แอพพลิเคชั่นสามารถเพิ่มคอมมานด์ไน

Canvas ได้โดยใช้เมธ็อด addCommand() เหมือนกับซับคลาสของ Displayable อื่นๆ และยังสามารถรีจิสเตอร์ CommandListner ใน Canvas ด้วยเมธ็อด setCommandListener() ได้เช่นกัน

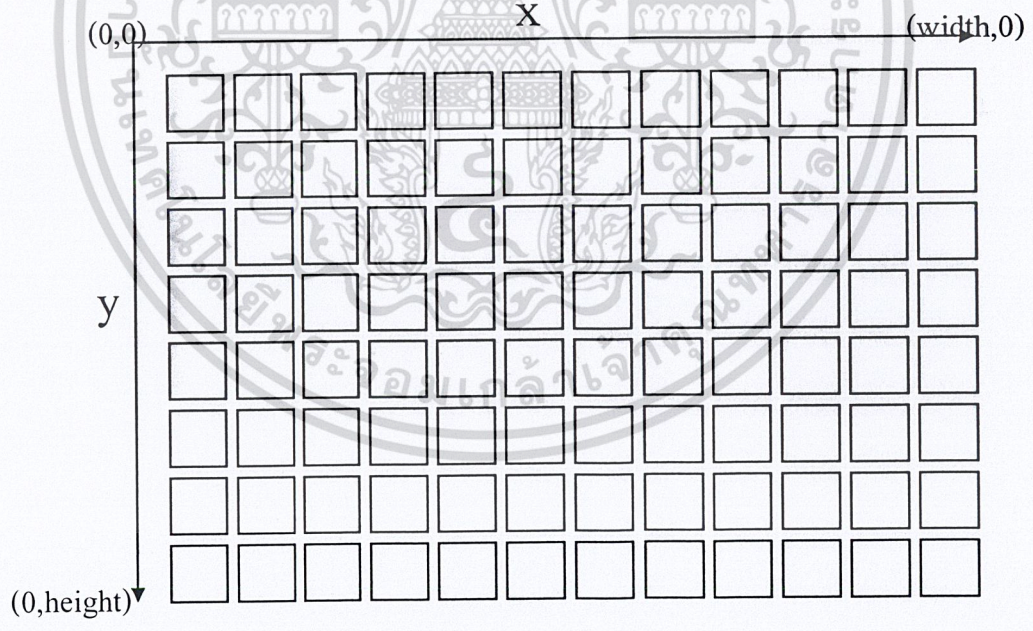
4.3.2 กราฟิก ทำหน้าที่สร้างภาพเรขาคณิต 2 มิติ คลาสนี้คล้ายกับ java.awt.Graphics ใน J2SE โหมดในการวาดของคลาส Graphics เท่านั้นที่เป็นแบบแทนที่พิกเซล พิกเซลที่มีอยู่เดิมบนจอจะถูกแทนที่ด้วยพิกเซลปัจจุบันที่ระบุในออบเจ็กต์ Graphics ในคลาส Graphics ไม่มีฟังก์ชั่นอื่นที่ทำหน้าที่รวมค่าของพิกเซล อาทิ คำสั่งเกี่ยวกับบิคแมพหรือการผสม/ ซ้อนภาพแบบแอลฟ่า (การทำให้ฉากหลังโปร่งใสเพื่อซ้อนกับอีกภาพ)

ออบเจ็กต์ Graphics สามารถวาดบนจอโดยตรงหรือจะวาดลงในบัฟเฟอร์ของรูปที่ไม่อยู่บนจอก็ได้ ฉากที่ออบเจ็กต์ Graphics วาดลงไปนั้นจะเป็นอะไรก็ขึ้นอยู่กับแหล่งกำเนิดของออบเจ็กต์ Graphics

วิธีเดียวที่จะดึงออบเจ็กต์ Graphics มาวาดได้ ทำได้โดยใช้เมธ็อด paint(Graphics g) ของออบเจ็กต์ Canvas แอพพลิเคชั่นสามารถวาดเนื้อหาบนออบเจ็กต์ Canvas ด้วยการใช้ออบเจ็กต์ Graphics นี้เท่านั้น

แอพพลิเคชั่นสามารถดึงออบเจ็กต์ Graphics เพื่อวาดลงในบัฟเฟอร์ของรูปได้โดยใช้เมธ็อด getGraphics() แอพพลิเคชั่นสามารถเก็บออบเจ็กต์ Graphics ที่ดึงมาได้ตลอดและเรียกใช้คำสั่งใสการวาดได้ตลอดเวลา

ระบบพิกัดของกราฟิกมีจุดปัก (เรียกว่าจุด anchor) เริ่มที่มุมบนซ้ายของพื้นที่วาด ค่า x จะเพิ่มจากซ้ายไปขวา และค่า y จะเพิ่มจากบนลงล่าง

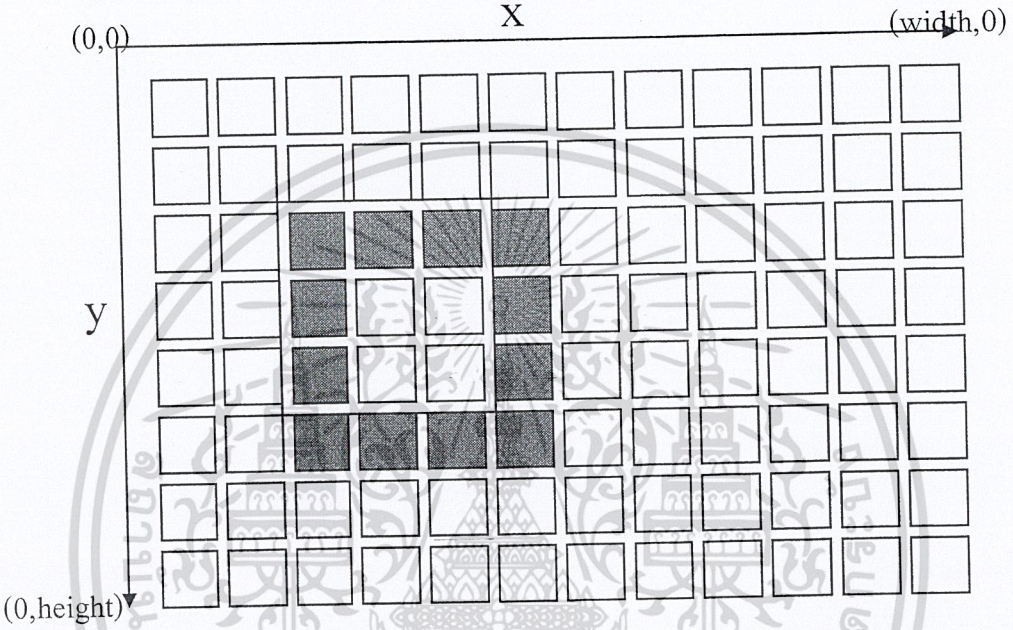


รูปที่ 4.6 ระบบพิกัดของ Graphic

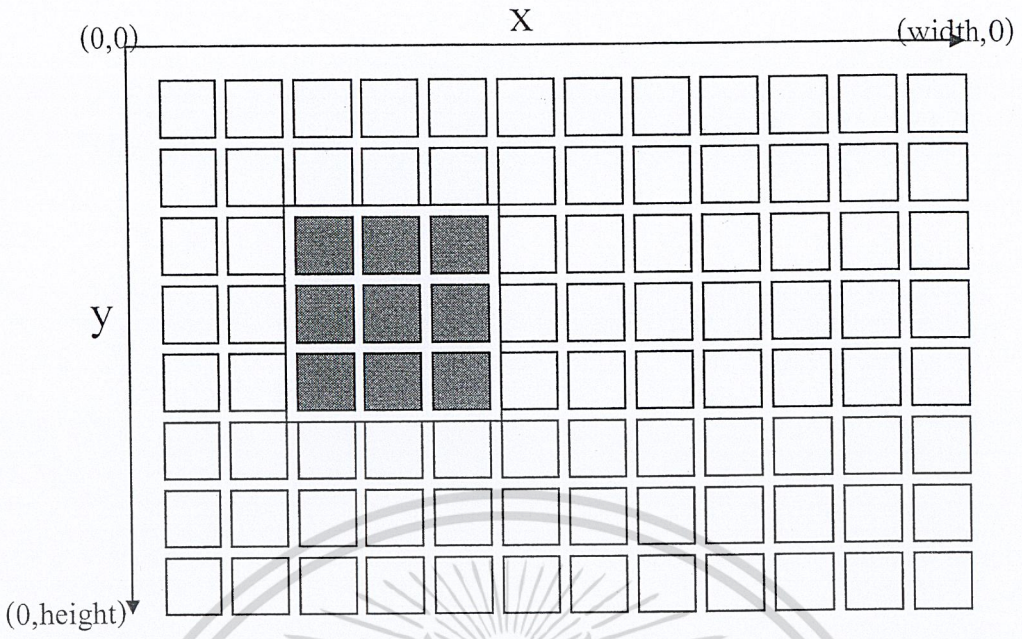
ระบบพิกัดจะอ้างถึงตำแหน่งระหว่างพิกเซล ไม่ใช่ตำแหน่งของพิกเซลเอง ตัวอย่างเช่น พิกเซลแรกที่มุมบนซ้ายสุดจะอยู่ในช่องสี่เหลี่ยมที่รอบด้วยพิกัด (0,0),(1,0),(0,1),(1,1)

คำสั่งวาดลายเส้นโครงร่างจะเดินไปตามพิกัดโดยเสมือนมีปากกาติดอยู่ข้างล่างและข้างขวาของเส้นทางที่เดินขนาดของปากกาคือกว้าง 1 พิกเซล ยาว 1 พิกเซล ดังนั้นพิกเซลที่อยู่ทางขวาหรือใต้พิกัดของเส้นทางวาดจะเป็นเป้าหมายของคำสั่ง ตัวอย่างเช่นการเรียก drawRect(2,2,3,3) จะได้ผลดังรูป 4.7

คำสั่งการเติมภาพจะแตกต่างจากคำสั่งการวาด เฉพาะพิกเซลที่ถูกปิดล้อมด้วยพิกัดของเส้นทางวาดเท่านั้นที่จะได้รับผลคำสั่ง ตัวอย่างเช่น fillRect(2,2,3,3) จะได้ผลดังรูป 4.8



รูปที่ 4.7 เส้นทางการวาด



รูปที่ 4.8 การเติมภาพ

คลาสกราฟิกจะใช้พารามิเตอร์สำหรับวาดรูปเหมือนๆกัน พารามิเตอร์เหล่านั้นได้แก่ สี ฟอนต์ และลายเส้น

API ระดับล่างใช้ระบบสีแบบ 24 บิต นั่นคือใช้ 8 บิตสำหรับแต่ละองค์ประกอบของสี คือ สีแดง สีเขียว และสีน้ำเงิน เนื่องจากอุปกรณ์บางชนิดอาจไม่สนับสนุนสีแบบ 24 บิต การนำไปใช้งานจึงต้องมีการจับคู่สีที่แอพพลิเคชั่นต้องการใหม่ให้เป็นสีที่อุปกรณ์สามารถแสดงผลได้ แอพพลิเคชั่นสามารถตรวจว่าอุปกรณ์สามารถแสดงผลสีหนึ่งๆ ได้หรือไม่ด้วยเมธ็อด Display.isColor() และดูจำนวนสี (ถ้า isColor() เป็น true) หรือระดับเฉดเทา (ถ้า isColor() เป็น false) ด้วยเมธ็อด Display.numColors()

โดยปกติแล้วก่อนที่แอพพลิเคชั่นจะวาดเนื้อหาบนจอหรือบนรูปภาพ จะต้องกำหนดสีของ foreground และ background ก่อนใน J2SE มีเมธ็อด setBackgrount() และ setForegrount() เพื่อใช้สำหรับกานี้ แต่ในคลาส Graphics มีเพียงเมธ็อด setColor() เท่านั้น การเซ็ตสี background ต้องใช้วิธี setColor() เพื่อให้ได้สีที่ต้องการ จากนั้นจึงใช้วิธีเติมสีรูปสี่เหลี่ยมด้วย fillRect() แล้วจึงค่อยเซ็ตสีที่ต้องการสำหรับ foreground

คลาส Graphics มีขั้นตอนสำหรับการวาดรูปและเติมรูปสี่เหลี่ยมอยู่ 2 ชุด ชุดแรกเป็นการวาดสี่เหลี่ยมที่มีมุมเหลี่ยม

```
void drawRect(int x, int y, int width, int height)
void fillRect(int x, int y, int width, int height)
```

ชุดที่ 2 เป็นการวาดสี่เหลี่ยมที่มีมุมโค้งมน

void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

คลาส Graphics มีเมธ็อดดังต่อไปนี้สำหรับวาดและเติมเส้นโค้งหรือรูปส่วนโค้งซึ่งระบุขอบเขตด้วยพิกัดของรูปสี่เหลี่ยม

void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

คลาส Grapics มีเมธ็อดสำหรับวาดข้อความดังต่อไปนี้

void drawChar(char character, int x, int y, int anchor)

void drawChars(char[] data, int offset, int length, int x, int y, int anchor)

void drawstring(String str, int x, int y, int anchor)

void drawSubstring(String str, int offset, int len, int x, int y, int anchor)

นอกจากสีและลายเส้นแล้ว ฟอนต์ก็เป็นพารามิเตอร์อีกตัวหนึ่งที่ออบเจ็กต์ Graphics สามารถกำหนดได้

คลาส Font เป็นตัวแทนของฟอนต์และเมตริกซ์ของฟอนต์ แอพพลิเคชั่นไม่สามารถสร้างออบเจ็กต์ของฟอนต์ใหม่ได้ ต้องใช้ฟอนต์ที่มีอยู่เดิม ซึ่งสามารถเรียกดูรายละเอียดต่างๆ ของฟอนต์ได้โดยใช้เมธ็อดดังนี้

Static Font getFont(int face, int style, int size)

การวาดข้อความจะขึ้นอยู่กับจุดปักหรือ anchor ใช้ลดปริมาณการคำนวณค่าเมื่อแอพพลิเคชั่นต้องการวางข้อความ

จุดปักมีค่าคงที่ 3 ตัวในแนวนอนได้แก่ LEFT,HCENTER และ RIGHT และ 4 ตัวในแนวตั้งได้แก่ TOP,BASELINE,VCENTER และ BOTTOM การกำหนดจุดปักต้องระบุด้วยค่าคงที่ในแนวนอน 1 ค่ารวมด้วยโอเปอเรเตอร์ OR กับค่าคงที่ในแนวตั้ง 1 ค่า อาทิ TOP,BASELINE และ BOTTOM แต่ไม่สามารถใช้ค่า VCENTER กับพารามิเตอร์ของการวาดข้อความที่จุดปัก ทั้งนี้ เนื่องจากการวางตำแหน่งตรงกลางในแนวตั้งไม่มีประโยชน์กำหนดยาก และนำไปใช้งานจริงได้ลำบาก

คลาส Graphics มีเมธ็อดเดียวสำหรับการวาดรูป

void drawImage(Image img, int x, int y. int anchor)

รูปที่วาดอาจจะแก้ไขได้หรือไม่ได้ ตำแหน่งจุดปักจะใช้ค่าคงที่เหมือนกับที่อธิบายในเรื่องการวาดข้อความ ข้อแตกต่างประการเดียว คือ จุดปัก BASELINE ของข้อความจะถูกแทนที่ด้วย VCENTER ของการวางรูปภาพ

4.3.3 อีเวนต์ระดับล่างและการจัดการกับอีเวนต์ คลาส Canvas ใช้ API ส่วนติดต่อกับผู้ใช้ระดับล่าง และรับผิดชอบการจัดการอีเวนต์ในระดับล่าง เช่น อีเวนต์ปุ่ม(คีย์) และอีเวนต์ตัวชี้หรือพอยน์เตอร์ (สำหรับอุปกรณ์บางรุ่น)

ใช้โมเดลการจัดการอีเวนต์แบบเดียวกับ AWT ของ J2SE นั่นคือ ทุกอีเวนต์จะถูกส่งไปที่ Canvas และแอพพลิเคชั่นจะทำหน้าที่คัดเอาอีเวนต์ที่ไม่ต้องการออกไป

4.3.3.1 อีเวนต์ Show และ Hide เมธ็อด showNotify() ถูกเรียกขึ้นมาก่อนออบเจ็กต์ Canvas จะแสดงผลบนจอ และเมธ็อด hideNotify() ถูกเรียกหลังจากที่ออบเจ็กต์ Canvas ถูกย้ายออกไปจากจอแล้ว การเปลี่ยนสถานการณ์มองเห็น/ไม่เห็นของ Canvas อาจเกิดจากซอฟต์แวร์แอพพลิเคชั่นที่ย้าย MIDlet สลับไปมาระหว่าง foreground กับ background หรืออาจเกิดจากข้อความของระบบโผล่ขึ้นมาบัง Canvas ดังนั้น showNotify() และ hideNotify() จึงไม่อยู่ภายใต้การควบคุมของ MIDlet และอาจเกิดอีเวนต์เหล่านี้ขึ้นได้บ่อยๆ

เมธ็อดของอีเวนต์ Key และ Pointer เมธ็อด paint() commandAction() เหล่านี้ จะถูกเรียกก็ต่อเมื่อสามารถมองเห็นออบเจ็กต์ Canvas ได้บนอุปกรณ์ นั่นคือจะเรียกเมธ็อดเหล่านี้ได้หลังจาก showNotify() ถูกเรียกขึ้นมาแล้วและก่อนจะเรียก hideNotify() เท่านั้น เมื่อ hideNotify() ถูกเรียก เมธ็อดของอีเวนต์ Key และ Pointer เมธ็อด paint() commandAction() จะไม่ถูกเรียกจนกว่า showNotify() ถัดไปจะถูกเรียก

4.3.3.2 อีเวนต์ Key แอพพลิเคชั่นจะได้รับอีเวนต์ Key เมื่อผู้ใช้กดปุ่มใดๆบนแป้น

4.3.3.2.1 Key Code และ Key Name ปุ่มแต่ละปุ่มบนจอจะมีค่ารหัสหรือ key code กำหนดไว้ซึ่งเป็นค่าที่ไม่ซ้ำกัน เว้นแต่ปุ่มที่มีค่า key code ซ้ำกันนั้นสามารถใช้แทนกันได้ MIDP กำหนดค่า key code ต่อไปนี้สำหรับปุ่มบนแป้นโทรศัพท์มาตรฐาน KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_STAR และ KEY_POUND

ค่ารหัสเหล่านี้มีค่าเทียบได้กับรหัสยูนิโค้ดของตัวอักษร เช่น KEY_NUM0 หรือปุ่มเลข 0 บนแป้นกด จะมีค่ารหัสเป็น 48 นอกจากปุ่มเหล่านี้แล้วบนแป้นกดอาจมีปุ่มอื่นๆ key code นอกเหนือจากที่ระบุไว้ข้างต้นจะมีค่าติดลบเพื่อให้แตกต่างจากปุ่มมาตรฐานที่เป็นค่าบวก

แต่ละปุ่มจะมีชื่อหรือ key name ชื่อของปุ่มสามารถดูได้จาก getKeyName() ของคลาส Canvas

เมธ็อดในการจัดการอีเวนต์ของ key มี 3 เมธ็อดด้วยกัน

protect void keyPressed(int keyCode)

protect void keyReleased(int keyCode)

protect void keyRepeated(int keyCode)

อีเวนต์ key-repeated จะเกิดขึ้นเมื่อผู้ใช้กดปุ่มติดๆกันสองครั้งในช่วงเวลาสั้นๆ อุปกรณ์บางชนิดสามารถจัดการอีเวนต์ key-repeated ได้ บางชนิดจัดการไม่ได้ วิธีตรวจสอบว่าอุปกรณ์รู้จักอีเวนต์ key-repeated หรือไม่ทำได้โดย

public boolean hasRepeatEvents()

การเปลี่ยนแปลงที่เกิดกับซับคลาสของ Screen จะแสดงบนจอโดยอัตโนมัติ แอพพลิเคชั่นไม่จำเป็นต้องปรับปรุงหน้าจอ แต่สำหรับคลาส Canvas หน้าจอไม่สามารถปรับปรุงโดยอัตโนมัติได้เอง แอพพลิเคชั่นจะต้องใช้เมธ็อด repaint() เพื่อปรับปรุงหน้าจอเมื่อมีการเปลี่ยนแปลง

4.3.3.2.2 แอกชั่นเกม แอพพลิเคชั่นหลายตัวร้องการอีเวนต์ของปุ่มลูกศรและอีเวนต์อื่นๆ สำหรับเล่นเกม MIDP กำหนดค่าของปุ่มเหล่านั้นดังนี้ UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C และ GAME_D แต่ละปุ่มจะจับคู่กับแอกชั่นของเกม 1 แอกชั่น อย่างไรก็ตามแอกชั่นเกมหนึ่งๆ อาจใช้ปุ่มรวมกันมากกว่า 1 ปุ่ม ค่า key code และแอกชั่นเกมสามารถแปลงกลับไปมาได้โดยใช้เมธ็อดต่อไปนี้

int getKeyCode(int gameAction)

int getGameAction(int keyCode)

วิธีการจับคู่แอกชั่นเกมกับ key code จะขึ้นอยู่กับระบบ เช่น ในอุปกรณ์บางตัวแอกชั่น UP, DOWN, LEFT, RIGHT อาจจับคู่กับปุ่มลูกศรทั้งสี่ แต่อุปกรณ์อีกตัวอาจจับคู่ไปเป็นปุ่มตัวเลข 2, 4, 6, 8 เป็นต้น แอพพลิเคชั่นควรใช้วิธีแปลงค่า key code เป็นแอกชั่นเกมด้วย getGameAction() เพื่อให้สามารถโยกย้ายไปใช้กับอุปกรณ์อื่นๆได้ง่าย

4.3.3.3 อีเวนต์ Pointer สำหรับอุปกรณ์ที่มีจอแบบสัมผัส พอยน์เตอร์ (Pointer) หรือตัวชี้อาจเป็นวัตถุใดๆ ที่ใช้กดบนจอภาพ เช่น อุปกรณ์ชี้ ปากกาหรือแม้แต่นิ้วมือ อีเวนต์ของ pointer มี 3 แบบได้แก่ pointer pressevent, pointerrelease event และ pointer drag event

เมื่อมีออบเจ็กต์ Screen อยู่บนหน้าจอ การเปลี่ยนแปลงใดๆ ที่เกิดกับออบเจ็กต์ Screen จะส่งผลต่อหน้าจอโดยอัตโนมัติ แอพพลิเคชั่นไม่จำเป็นต้องปรับปรุงหน้าจอเอง แต่ถ้าเป็นออบเจ็กต์ Canvas อยู่บนจอ แอพพลิเคชั่นต้องจัดการปรับปรุงหน้าจอ เมื่อมีการเปลี่ยนแปลงบน Canvas โดยมีเมธ็อดในการปรับปรุงหน้าจอ 2 เมธ็อดดังนี้

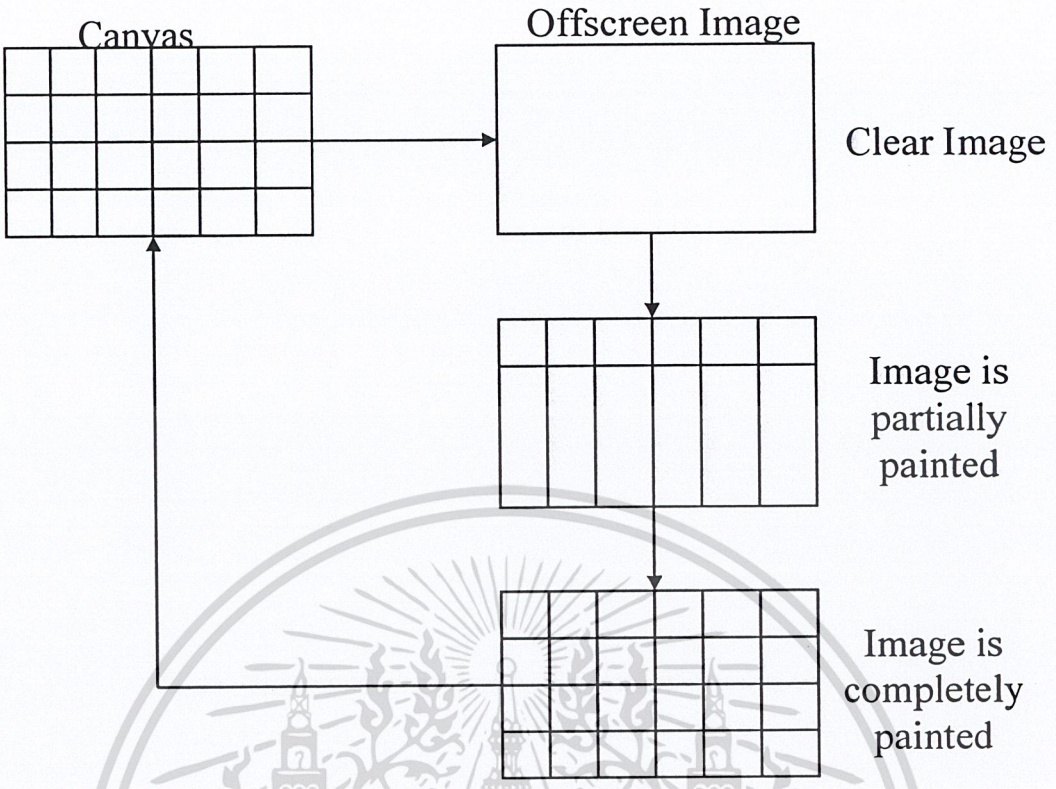
public final void repaint(int x, int y, int width, int height)

public final void repaint()

ถ้า Canvas อยู่ในสถานะที่มองไม่เห็น การเรียกเมธ็อดทั้งสองนี้ก็จะไม่เกิดผลใดๆ

เมื่อความเร็วในการทำงานของ KVM(K Virtual Machine) ช้าลง บางครั้งอาจสังเกตเป็นภาพกระตุกซึ่งมีสาเหตุมาจากการที่ MIDlet ต้องล้างพื้นที่วาดด้วยสี background ก่อนที่เนื้อหาใหม่จะถูกวาดลงไป วิธีแก้ทำได้โดยใช้เทคนิคบัฟเฟอร์ 2 ชั้น

เทคนิคบัฟเฟอร์ 2 ชั้น มีหลักการง่ายๆ คือ แทนที่จะวาดหรือล้างเนื้อหาบนพื้นที่ Canvas โดยตรง เราจะสร้างภาพที่อยู่นอก Screen ขึ้นมาแทนโดยเป็นเสมือนอีกสำเนาหนึ่งของ Canvas การวาดหรืออัพเดต Canvas จะเอาบัฟเฟอร์นี้เป็นพื้นที่ใช้งาน เมื่อเสร็จแล้วก็คัดลอกไปทับพื้นที่ Canvas จริงโดยไม่ต้องรอให้มีการเคลียร์ Screen ก่อน



รูปที่ 4.9 ขั้นตอนการอัพเดตหน้าจอด้วย offscreen image

MIDP บางตัวใช้เทคนิคแบบนี้สำหรับออบเจ็กต์ Graphic ที่อยู่บน Canvas คุณสามารถตรวจสอบว่าออบเจ็กต์ Graphics ใน Canvas ใช้เทคนิคนี้อยู่หรือไม่ด้วย

public boolean isDoubleBuffered()

สามารถเขียนโปรแกรมโดยใช้เทคนิคบัฟเฟอร์ 2 ชั้นในแอพพลิเคชั่นได้ ขั้นแรกสร้างรูปที่อยู่นอก Screen ที่มีขนาดเท่า Canvas โดยใช้

buffer_image = Image.createImage(width, height)

จากนั้นก็ดึงออบเจ็กต์ Graphics เพื่อมาวาด

bg = buffer_image.getGraphics()

สามารถดึงเอาออบเจ็กต์ของ Graphics ที่อยู่นอก Screen ได้โดยใช้ getGraphics() ของรูปนั้นและจะยังคงอยู่ตลอดเวลาที่แอพพลิเคชั่นทำงาน ถ้าออบเจ็กต์ Graphics ไม่ถูกทำลายหลังจากการใช้งานก็จะทำให้สิ้นเปลืองหน่วยความจำ ดังนั้นหลังจากใช้ออบเจ็กต์ Graphics แล้วต้องทำลายด้วย

bg = null

ด้วยการใช้เทคนิคบัฟเฟอร์ 2 ชั้น ทำให้แอพพลิเคชั่นสามารถขจัดปัญหาภาพกระตุกได้ อย่างไรก็ตามผลที่ตามมาคือแอพพลิเคชั่นจะตอบสนองต่ออีเวนต์ Key ช้าลงเพราะต้องคัดลอกรูปที่อยู่นอกจอมายัง Canvas

นอกจากนี้ เทคนิคบัฟเฟอร์ 2 ชั้น จะใช้หน่วยความจำมากและอุปกรณ์บางชนิดอาจไม่ได้เผื่อไว้ วิธีแก้ไขปัญหาภาพกระตุกอีกทางเลือกหนึ่งคือค่อยๆ ลบส่วนของหน้าจอที่เปลี่ยนแปลงอย่างระมัดระวัง แทนที่จะลบทั้งหน้าจอ

4.4 หน่วยเก็บข้อมูลแบบคงตัว

J2ME MIDP มีระบบฐานข้อมูลแบบเรกคอร์ดง่ายๆ ที่เรียกว่า record management system (RMS) ตารางในฐานข้อมูลจะเรียกว่าเป็น record store และแต่ละรายการข้อมูลใน recor stroe จะเรียกว่า 'เรกคอร์ด ชุดคำสั่ง API สำหรับสร้าง เข้าถึง และจัดการกับเรกคอร์ดต่างๆ นั้นจะอยู่ในแพ็กเกจ javax.microedition.rms

4.4.1 Record Stores rms ที่มีใน MIDP เป็นระบบฐานข้อมูลแบบง่ายๆ record store ซึ่งเทียบเท่ากับตารางในฐานข้อมูล คือไฟล์ที่ประกอบด้วยชุดเรกคอร์ด รายละเอียดของ record store จะกำหนดในคลาส RecordStore class

4.4.2 เรกคอร์ด กลุ่มของข้อมูลใน record store เรียกว่าเรกคอร์ด โครงสร้างของเรกคอร์ดเกิดจากอาร์เรย์ของไบท์และจะกำหนดค่าสำหรับแต่ละเรกคอร์ดด้วย RecordId เรกคอร์ดจะสามารถสร้างขึ้นมาใหม่ ลบ เรียกดู หรือแก้ไขได้ก็ต่อเมื่อได้เปิด record store ด้วยเมธ็อดใน API ดังนี้

```
public int addRecord(byte[] data, int offset, int numBytes)
public void deleteRecord(int recordId)
public int getRecord(int recorded, byte[] buffer, int offset)
public byte[] getRecord(int recordId)
public void setRecord(int recorded, byte[] newData, int offset, int numBytes)
```

การเพิ่มเรกคอร์ด เรกคอร์ดแรกที่สร้างขึ้นใน record store จะมีค่า RecordID = 1 เรกคอร์ดที่สร้างเพิ่มเตอมในภายหลังจะมีค่า recorded เพิ่มขึ้นครั้งละหนึ่งค่า

การลบเรกคอร์ด ทำได้โดย deleteRecord(int recordID) หากเรียกเมธ็อดนี้โดยไม่เปิด record store จะเกิด RecordStoreNotOpenException หรือหากหาเรกคอร์ดที่ต้องการลบไม่พบ ก็จะเกิด InvalidRecordIDException นอกจากนี้หากมีเอกเซ็พชั่น record store แบบอื่นๆ เกิดขึ้น ก็จะเกิด RecordStoreException ตามมา

ค่า RecordID ของเรกคอร์ดที่ลบออกไปแล้ว จะไม่นำกลับมาใช้อีก เพราะถือว่าเป็นค่าหลักสำหรับเร็กคอร์ดใหม่ที่อยู่ถัดไป โดยเรกคอร์ดถัดไปจะยึดค่าเดิมเป็นหลัก และเพิ่มค่าเข้าไปอีกหนึ่งค่าเสมอ

การลบ record ไม่ทำให้ขนาดของ record store ลดลง เพราะมาทำเครื่องหมายพื้นที่ว่างหน้าบล็อกข้อมูลของเรกคอร์ดที่ลบทิ้งไป พร้อมนำไปใช้งานอื่นๆต่อ

4.4.3 RecordEnumeration หลังจากที่เรกคอร์ดถูกลบออกไปแล้ว RecordId ใน record store จะไม่เป็นเลขที่ต่อเนื่องกันอีก และแม้จะสามารถอ่านค่าเรกคอร์ดทั้งหมดโดยใช้ MIDlet

การอ่านข้อมูลทั้งหมดใน record store ด้วยวิธีนี้เป็นการอ่านค่าแบบลองผิดลองถูก หากเรกคอร์ดที่มีค่ากำกับอยู่จริง ก็จะอ่านค่าเรกคอร์ดโดยใช้ getRecord() แต่หากไม่พบเรกคอร์ดนั้นๆ ก็จะเกิด InvalidRecordIDException การใช้วิธีนี้อ่านค่าจะได้ผลน้อยมาก หากมีการลบเรกคอร์ดใน record store ออกไปแล้วเป็นจำนวนมาก

API ของ RMS ได้เตรียม RecordEnumeration ซึ่งเข้าถึงข้อมูลใน record store ได้ดีกว่าไว้ เนื่องจากเป็นคลาสที่เข้าถึงเรกคอร์ดใน record store ได้ถึงสองทิศทางด้วยกัน ทั้งนี้ MIDP จะกำหนด RecordEnumeration เป็นอินเตอร์เฟส และปล่อยให้บริษัทผู้ผลิตกำหนดรายละเอียด ดังนั้น ผู้ที่สนับสนุน J2ME MIDP จะต้องนำ RecordEnumeration มาใช้งานในอุปกรณ์ของตนเสมอ นักพัฒนาถือว่า RecordEnumeration เป็นคลาสหลักคลาสหนึ่ง

โครงสร้างของ RecordEnumeration เป็นลักษณะของรายการที่มีปลายเชื่อมต่อ 2 ด้าน โดยแต่ละจุดเชื่อมแทนหนึ่ง recordId มีหน้าที่ในการจัดลำดับ recordId ของเรกคอร์ดใน record store สามารถสร้าง RecordEnumeration ในคลาส RecordStore โดย

public RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)

คลาส RecordEnumeration มี API สำหรับการเข้าถึงข้อมูลในเรกคอร์ดของ record store ดังนี้

```
void destroy()
boolean hasNextElement()
boolean hasPreviousElement()
byte[] nextRecord()
int nextRecord()
int numRecord()
byte[] previousRecord()
int previousRecord()
void rebuild()
void reset()
```

เมื่อต้องใช้ตัวแจกแจงหมายเลขหลายครั้ง ต้องใช้ reset() เพื่อย้ายจากจุดเชื่อมต่อปัจจุบัน (พอยน์เตอร์) กลับไปตั้งต้นที่ recordId แรกในรายการข้อมูลก่อน

4.5 หลักการเขียนโปรแกรมเครือข่ายใน MIDP ของ J2ME

การเขียนโปรแกรมเครือข่าย นำเอาจุดเด่นเรื่องความสามารถในการเชื่อมต่อของอุปกรณ์เหล่านี้มาใช้ มีบทบาทสำคัญในการพัฒนาแอพพลิเคชั่นสำหรับอุปกรณ์ไร้สาย

4.5.1 การเขียนโปรแกรมเครือข่ายด้วย J2SE และ J2ME

การเขียนโปรแกรมสำหรับเครือข่ายด้วย J2SE จะง่ายและตรงไปตรงมา ซึ่งใน J2SE ได้ให้ไลบรารีของเครือข่ายซึ่งมีฟังก์ชั่นให้ใช้มากมาย ในแพ็กเกจ java.io มีคลาสที่สนับสนุนการนำเข้า –

ส่งออกข้อมูลประมาณ 60 คลาส ขณะที่อีกประมาณ 20 คลาสในแพ็กเกจ java.net สนับสนุนการทำงานด้วนเครือข่าย

อย่างไรก็ดี คลาสส่วนใหญ่ในแพ็กเกจทั้งสองออกแบบมาสำหรับระบบคอมพิวเตอร์ทั่วไปที่มีหน่วยประมวลผล หน่วยความจำ และพื้นที่เก็บข้อมูลที่เพียงพอ ไฟล์คลาสเหล่านี้มีขนาดคงที่ทั้งสิ้นประมาณ 200 กิโลไบต์ ซึ่งใหญ่เกินกว่าที่จะใส่ลงไปในอุปกรณ์ไร้สาย ซึ่งมีทรัพยากรต่างๆ จำกัดและมีพื้นที่เก็บข้อมูลเพียงไม่กี่ร้อยกิโลไบต์

ไม่เฉพาะขนาดของไฟล์เท่านั้นที่เป็นอุปสรรคในการใช้งานบนอุปกรณ์ไร้สาย Java 2 Micro Edition จะต้องรองรับอุปกรณ์เคลื่อนที่ได้หลากหลายประเภท การติดต่อเครือข่ายและไลบรารีของไฟล์ I/O ในแต่ละอุปกรณ์แตกต่างกันมาก เนื่องจากความต้องการขั้นต่ำในการเชื่อมต่อและความสามารถของ I/O ที่แตกต่างกัน ความแตกต่างระว่างเครือข่ายทำให้ต้องกำหนดสาระสำคัญของการสื่อสาร (abstraction) ในจาวาไลบรารี 2 รูปแบบคือ การเชื่อมต่อแบบดาต้าแกรมสำหรับเครือข่ายสลับกลุ่มข้อมูล และคือ การเชื่อมต่อแบบซ็อกเก็ตสำหรับเครือข่ายแบบสลับวงจร ทั้งนี้ ผู้ผลิตซึ่งสนับสนุนดาต้าแกรมอาจไม่สนใจจะสนับสนุนซ็อกเก็ตก็ได้ และเช่นเดียวกันในกรณีของผู้ผลิตซึ่งสนับสนุนซ็อกเก็ต

การเข้าถึงไฟล์ I/O ในอุปกรณ์ไร้สายก็เป็นเรื่องเฉพาะอุปกรณ์และมีวิธีต่างๆ ในการนำไปใช้เช่นกัน ผู้ผลิตที่สนับสนุนกลไก I/O ประเภทหนึ่งแล้วมักจะไม่สนับสนุนกลไกประเภทอื่นๆ อีกเนื่องจากข้อจำกัดด้านหน่วยความจำ

เครือข่ายใน J2ME จะต้องมีความยืดหยุ่นสูงเพื่อรองรับอุปกรณ์หลากหลายประเภทขณะเดียวกันก็จะต้องสนับสนุนเฉพาะอุปกรณ์ด้วย ดังนั้น จึงได้ริเริ่มนำเอากรอบการติดต่อสื่อสารทั่วไปเข้ามาใช้ใน CLDC แนวคิดของกรอบการสื่อสารทั่วไปคือ การกำหนดสาระสำคัญ (abstraction) ของเครือข่ายและไฟล์ I/O เพื่อสนับสนุนอุปกรณ์พกพาประเภทต่างๆ ให้มากที่สุดเท่าที่จะทำได้ และปล่อยให้การนำสาระสำคัญของการสื่อสารมาใช้งานจริงเป็นหน้าที่ของผู้ผลิตอุปกรณ์แต่ละราย ทั้งนี้ได้กำหนดสาระสำคัญของการสื่อสารจะไว้ในอินเตอร์เฟสของจาวา ผู้ผลิตจะเป็นผู้เลือกว่าจะนำ MIDP หรือ PDAP มาใช้งานโดยพิจารณาตามความสามารถของอุปกรณ์ของตน

4.5.2 กรอบการติดต่อสื่อสารทั่วไป (The Generic Connection Framework)

นำกรอบการติดต่อสื่อสารทั่วไปเข้ามาใช้ใน CLDC ของ J2ME ก็เพื่อที่จะสะท้อนให้เห็นถึงความต้องการเครือข่ายขนาดเล็กและไฟล์ I/O สำหรับอุปกรณ์เคลื่อนที่ประเภทต่างๆ

กรอบการติดต่อสื่อสารทั่วไปสร้างฟังก์ชั่นเครือข่ายและคลาสไฟล์ I/O จากแพ็กเกจ java.io และ java.net ของ J2SE หรือเป็นซับเซ็ตของคลาสใน J2SE แต่มีขนาดเล็กกว่า คลาสและอินเตอร์เฟสทั้งหมดของ J2ME รวมอยู่ในแพ็กเกจเดียว คือ javax.microedition.io

เพื่อให้กรอบการติดต่อสื่อสารทั่วไปมีความยืดหยุ่นและสนับสนุนอุปกรณ์ได้หลากหลายตามที่ต้องการ จะนำสาระสำคัญในการติดต่อสื่อสารรูปแบบต่างๆมาใช้ ได้แก่ อินเตอร์เฟสการติดต่อ 7 ลักษณะ คือ Connection, ContentConnection, DatagramConnection, InputConnection, OutputConnection, StreamConnection และ StreamConnectionNotifier

4.5.3 การเขียนโปรแกรมเครือข่ายบนอุปกรณ์ไร้สายโดยการเชื่อมต่อแบบซ็อกเก็ต

ซ็อกเก็ตเป็นกลไกการติดต่อสื่อสารระหว่างโปรแกรมที่ทำงานบนเครือข่ายปลายทาง 2 แห่ง อาทิ อุปกรณ์ไร้สายกับเซิร์ฟเวอร์ระยะไกล หรือระหว่างอุปกรณ์ไร้สาย 2 อุปกรณ์ มีประสิทธิภาพน่าเชื่อถือ ความสามารถในการเชื่อมต่อแบบซ็อกเก็ตในอุปกรณ์เคลื่อนที่บางประเภททำให้เกิดแอพพลิเคชั่นที่เขียนด้วย J2ME บนไคลเอนต์และเซิร์ฟเวอร์ต่างๆ การเชื่อมต่อแบบซ็อกเก็ตกับเซิร์ฟเวอร์ SMTP และ POP3 ทำให้ไคลเอนต์อีเมล์บนอุปกรณ์ไร้สายสามารถรับและส่งอีเมล์ข้อความผ่านระบบอินเตอร์เน็ตได้ตามปกติ

ซ็อกเก็ตช่วยนักพัฒนา J2ME สามารถพัฒนาแอพพลิเคชั่นทุกชนิดเพื่อใช้งานบนอุปกรณ์ไร้สายได้ อย่างไรก็ดีผู้ผลิตอุปกรณ์บางรายไม่สามารถสนับสนุนการเชื่อมต่อแบบซ็อกเก็ตในอุปกรณ์ MIDP ซึ่งหมายถึงว่าแอพพลิเคชั่นที่พัฒนาโดยใช้ซ็อกเก็ตอาจจะจำกัดเฉพาะในอุปกรณ์ไร้สายบางชนิด และเคลื่อนย้ายข้ามเครือข่ายอุปกรณ์ไร้สายต่างๆ ได้น้อยลง

ในการนำซ็อกเก็ตมาใช้ ก่อนอื่นผู้รับผู้ส่งจะต้องสร้างการเชื่อมต่อระหว่างซ็อกเก็ตเสียก่อน โดยฝ่ายหนึ่งจะคอยฟังคำร้องขอเชื่อมต่อ ขณะที่อีกฝ่ายหนึ่งขอเชื่อมตอ่เครือข่าย เมื่อซ็อกเก็ตทั้ง 2 ด้านเชื่อมต่อเข้าด้วยกัน ก็จะส่งผ่านข้อมูลไปยังอีกด้านหนึ่ง

การรับข้อมูลจากเครื่องเซิร์ฟเวอร์ระยะไกลจะต้องสร้าง InputConnection ขึ้นมาเพื่อรับ InputStream จากการเชื่อมต่อนั้น การส่งข้อมูลไปยังเครื่องเซิร์ฟเวอร์ระยะไกลจะต้องสร้าง OutputConnection ขึ้นมาเพื่อรับ OutputStream จากการเชื่อมต่อนั้น ใน J2ME การเชื่อมต่อทั้ง 3 ประเภทที่กำหนดไว้เพื่อจัดการกับ Stream ข้อมูลขาเข้า / ขาออก ได้แก่ InputStream OutputStream และ StreamConnection โดยมีความหมายตามชื่อเรียก คือ InputConnection กำหนดให้ input stream สามารถรับส่งข้อมูลได้ ขณะที่ OutputConnection ก็กำหนดให้ output stream สามารถส่งข้อมูลได้ การจะนำการเชื่อมต่อประเภทใดมาใช้เมื่อไหร่นั้นขึ้นอยู่กับว่าต้องการรับ – ส่ง หรือทั้งรับและส่งข้อมูล

การเขียนโปรแกรมแบบเครือข่ายโดยใช้ซ็อกเก็ตใน J2ME นั้นเขียนอย่างตรงไปตรงมา โดยมีขั้นตอนการทำงานดังนี้

1. เปิดการเชื่อมต่อแบบซ็อกเก็ตกับเซิร์ฟเวอร์ระยะไกลหรืออุปกรณ์ไร้สายอื่นโดยใช้ Connector.open()
2. สร้าง InputStream และ OutputStream จากเปิดการเชื่อมต่อแบบซ็อกเก็ตเพื่อใช้ในการรับ – ส่งกลุ่มข้อมูล
3. รับ – ส่งข้อมูลจากเซิร์ฟเวอร์ระยะไกลผ่านการเชื่อมต่อแบบซ็อกเก็ต ด้วยโอเปอเรชั่นการอ่านและเขียนในออบเจ็กต์ InputStream หรือ OutputStream
4. ปิดการเชื่อมต่อแบบซ็อกเก็ตและ input / output stream ก่อนออกจากโปรแกรม

บทที่ 5
การออกแบบตัวแอพพลิเคชั่นฝั่งเซิร์ฟเวอร์

5.1 รูปแบบโดยทั่วไปและวิธีการเล่น

รูปแบบของแอพพลิเคชั่นจะเป็นเกมส์ออนไลน์บนโทรศัพท์มือถือซึ่งเขียนโดยภาษาจาวา ในรูปแบบของเครือข่ายแบบซ็อกเก็ท โดยตัวของแอพพลิเคชั่นนี้ได้ถูกแบ่งออกเป็น 2 ส่วนคือ ส่วนของตัวเซิร์ฟเวอร์ และ ส่วนของตัวไคลเอนต์

การเริ่มของแอพพลิเคชั่นนี้จะเริ่มจากตัวเซิร์ฟเวอร์เปิดให้บริการไว้ก่อน แล้วมีตัวไคลเอนต์เป็นตัวเล่น โดยเริ่มแรกตัวผู้จะเล่นจะต้องมีแอพพลิเคนชั่นของเกมส์นี้ก่อน (โดยเป็นแอพพลิเคชั่นฝั่งไคลเอนต์) เมื่อต้องการจะร่วมเล่นเกมส์ก็จะทำการเปิดแอพพลิเคชั่นขึ้นมาแล้วต้องทำการลงทะเบียนก่อน เมื่อลงทะเบียนแล้วก็จะสามารถร่วมเล่นเกมส์นี้ผ่านทางตัวเซิร์ฟเวอร์ได้

ลักษณะการเล่นของเกมส์เมื่อตัวไคลเอนต์ที่ได้ทำการลงทะเบียนไว้แล้ว เข้ามาร่วมเล่น ตัวเซิร์ฟเวอร์ซึ่งทำการสร้างศัตรูขึ้นมา (โดยได้ออกแบบให้มีศัตรูในฉากมีจำนวนคงที่คือ 20 ตัว) ผู้เล่นก็จะสามารถทำการฆ่าศัตรูที่มีอยู่ในฉากได้ (ศัตรูไม่สามารถทำอะไรตัวผู้เล่นได้) เมื่อทำการฆ่าศัตรู ตัวผู้เล่นก็จะมีคะแนนเพิ่มขึ้น 1 คะแนน แล้วตัวศัตรูก็จะตาย เมื่อศัตรูตายตัวเซิร์ฟเวอร์จะทำการสร้างศัตรูขึ้นมาใหม่ และทำการสร้างไอเท็มขึ้นมา ณ ตำแหน่งที่ศัตรูถูกฆ่า เมื่อผู้เล่นทำการเก็บไอเท็มนั้น ก็จะทำให้ผู้เล่นได้รับค่าต่างๆตามลักษณะของไอเท็มนั้นๆ

เมื่อตัวไคลเอนต์ตั้งแต่สองตัวขึ้นไปออนไลน์อยู่ในระบบ ตัวไคลเอนต์แต่ละตัวก็จะสามารถมองเห็นตัวไคลเอนต์ตัวอื่นๆ ซึ่งตัวไคลเอนต์แต่ละตัวจะสามารถทำการสู้กันได้ โดยเมื่อทำการฟันแล้วตัวที่ทำการฟันจะได้คะแนนเพิ่มขึ้นมา 1 คะแนน ส่วนตัวที่ถูกฟันจะมีพลังชีวิตลดลงตามแต่ระดับความสามารถของตัวผู้เล่นนั้นๆ

เมื่อตัวไคลเอนต์ได้ทำคะแนนถึงจุดแต่ละจุดก็จะเป็นการเพิ่มระดับความสามารถของตัวผู้เล่นขึ้น คือเมื่อมีคะแนนถึง 0 - 4 คะแนนก็จะทำให้มีในระดับความสามารถเท่ากับ 1 คะแนน 5 – 9 คะแนนจะมีในระดับความสามารถเท่ากับ 2 คะแนน 10 – 14 คะแนนจะมีในระดับความสามารถเท่ากับ 3 เป็นเช่นนี้ไปเรื่อยๆ โดยความแตกต่างระหว่างระดับความสามารถจะแตกต่างกัน คือเมื่อตัวผู้เล่นโดนฟันพลังชีวิตที่ลดลงจะแตกต่างกัน คือเมื่อผู้เล่นที่มีระดับอยู่ในระดับความสามารถ 3 ขึ้นไป เมื่อโดนฟัน พลังชีวิตจะลดลง 1 หน่วย ถ้าอยู่ในระดับความสามารถ 2 เมื่อโดนฟัน พลังชีวิตจะลดลง 3 หน่วยและถ้าอยู่ในระดับ 1 เมื่อโดนฟัน พลังชีวิตจะลดลง 5 หน่วย และความแตกต่างระหว่างระดับความสามารถอีกอย่างคือเมื่อมีความสามารถเพิ่มขึ้นที่เก็บพลังชีวิตจะมีเพิ่มขึ้นกล่าวคือเมื่อแรกเริ่มที่เก็บพลังชีวิตจะอยู่ที่ 20 หน่วยเมื่อเพิ่ม 1 ระดับความสามารถก็จะทำให้เพิ่มที่เก็บพลังชีวิตเพิ่มขึ้นระดับละ 2 หน่วย

เมื่อตัวผู้เล่นตาย ก็จะทำการเก็บคะแนนของผู้เล่นคนนั้นไว้ แล้วทำการตัวผู้เล่นออกจากระบบ เพื่อให้ไคลเอนต์ตัวอื่นๆ ไม่สามารถมองเห็นตัวที่ตายไปแล้ว แล้วเมื่อผู้เล่นนั้นเข้ามาใหม่ก็จะทำการเอาข้อมูลที่เก็บเอาไว้มาใช้ คือเอาคะแนนเดิมก่อนตายมาใช้เมื่อเริ่มใหม่ โดยที่การเริ่มใหม่ตัวผู้เล่นจะมีพลังชีวิตเต็มที่เก็บพลังชีวิตของตัวผู้เล่นเอง

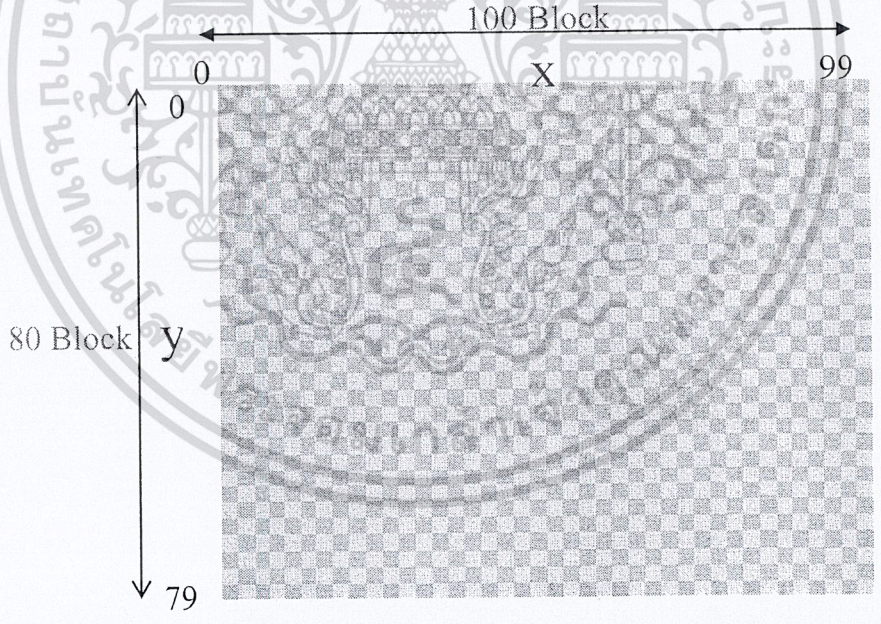
เมื่อผู้เล่นออกจากระบบก็จะทำการเก็บคะแนนของผู้เล่นคนนั้นไว้เช่นกัน แล้วจะทำการตัดผู้เล่นคนนั้นออกจากระบบเพื่อให้ไคลเอนต์ตัวอื่นๆ ไม่สามารถมองเห็นตัวที่ออกจากระบบไปแล้ว

5.2 ข้อมูลโดยทั่วไปของแอพพลิเคชั่น

ในตัว เซิร์ฟเวอร์ จะมี Record Store อยู่ 2 ตัวด้วยกัน คือ Record Store ตัวแรกจะทำการเก็บค่าการลงทำเบียนของ ไคลเอนท์ (เก็บ IP Port ID และคะแนน) ส่วนตัวที่สองจะเก็บค่าตัวที่กำลังออนไลน์อยู่ (เก็บ IP Port และ ID)

ในตัว ไคลเอนท์ ก็จะไม่มี Record Store แต่จะเก็บค่า IP Port และ ID ที่ได้จาก เซิร์ฟเวอร์ ในตอนลงทะเบียนไว้ที่ตัวแปรในส่วนของตัว ไคลเอนท์

การเก็บค่าตำแหน่งต่างๆของแผนที่จะถูกเก็บไว้ในค่า Map ซึ่งเป็นอาร์เรย์ 2 มิติ ของ Character โดยเป็นอาร์เรย์ที่มีขนาด 100 x 80 ซึ่งในแต่ละ Map[x][y] จะถูกเรียกว่าเป็น Block ดังนั้นใน Map ทั้งหมดจะมีทั้งหมด 8000 Block



รูปที่ 5.1 การจำลองแผนที่

ในแต่ละ Block จะเก็บค่าที่เป็น Character โดย Character แต่ละตัวจะมีความหมายที่แตกต่างกันดังตาราง

Char	ความหมาย	Char	ความหมาย
a	ผู้เล่นหันหน้า	j	ผู้เล่นหันซ้าย
b	ผู้เล่นหันหน้ากำลังเดิน	k	ผู้เล่นหันซ้ายกำลังเดิน
c	ผู้เล่นหันหน้ากำลังฟัน	l	ผู้เล่นหันซ้ายกำลังฟัน
d	ผู้เล่นหันหลัง	m	กำแพง
e	ผู้เล่นหันหลังกำลังเดิน	n	ก้อนหิน
f	ผู้เล่นหันหลังกำลังฟัน	o	หญ้า
g	ผู้เล่นหันขวา	t,u	Item
h	ผู้เล่นหันขวากำลังเดิน	z	ศัตรู
i	ผู้เล่นหันขวากำลังฟัน		

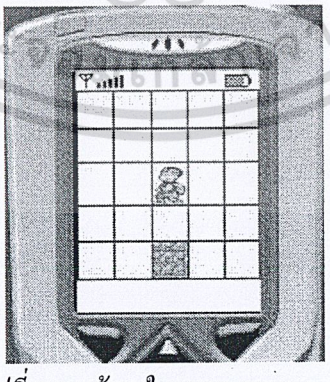
ตารางที่ 5.1 แสดงความหมายของ character แต่ละตัว

เช่น Map[x][y] = 'a' หมายความว่าที่ตำแหน่งแกนนอน x และแกนตั้ง y มีผู้เล่นที่หันหน้าอยู่ในตำแหน่งนี้ หรือ Map[x][y] = 'o' หมายความว่าที่ตำแหน่งแกนนอน x และแกนตั้ง y เป็นตำแหน่งหญ้า

การแสดงผลที่หน้าจอจะเป็นการเอาค่าที่อยู่ใน Map ที่เราต้องการ ซึ่งเป็น Character มาเปรียบเทียบว่าเป็นค่าอะไร ถ้าเป็นค่าไหนก็ให้แสดงรูปนั้น เช่น ถ้าเป็น 'a' ก็จะแสดงรูปผู้เล่นหันหน้า ถ้าเป็น 'm' ก็จะแสดงรูปกำแพง

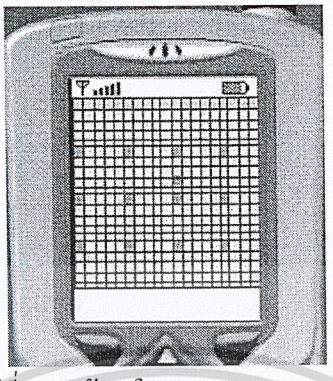
โดยการแสดงที่หน้าจอได้ทำให้มีการแสดงเป็น 2 มุมมอง

1. ค่า zoom มีค่าเป็น 0 จะได้มุมมองที่แสดงเป็น 5 x 5 Block โดยการแสดงรูปจะใช้รูปที่มีขนาด 19 x 20 pixels



รูปที่ 5.2 หน้าจอในมุมมองแบบ 5 x 5

2. ค่า zoom มีค่าเป็น 1 จะได้มุมมองที่แสดงเป็น 19 x 20 Block โดยการแสดงรูปจะใช้รูปที่มีขนาด 5 x 5 pixels



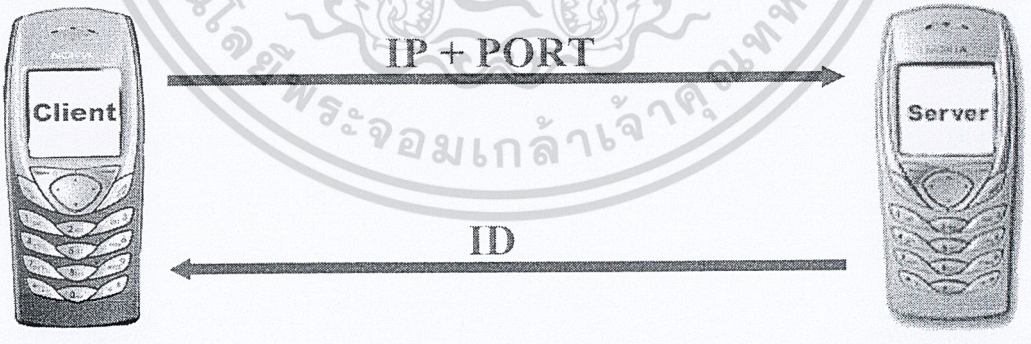
รูปที่ 5.3 หน้าจอในมุมมองแบบ 19 x 20

5.3 การทำงานฝั่งเซิร์ฟเวอร์และฟังก์ชั่นต่างๆ

การทำงานในฝั่งเซิร์ฟเวอร์ได้มีการทำงานหลายอย่างซึ่งเป็นการทำงานที่ต้องรับข้อมูลมาจากไคลเอนต์มาทำตามฟังก์ชั่นต่างๆตามที่ได้รับข้อมูลมา แล้วทำการส่งต่อไปยังไคลเอนต์ที่ต้องส่งตามฟังก์ชั่น โดยในตัวเซิร์ฟเวอร์เองจะมีหลายการทำงานแบ่งออกเป็น

5.3.1 การลงทะเบียน

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจเข้ามาก็จะมาทำการตรวจสอบว่าเป็นแพ็กเกจชนิดไหน โดยจะตรวจสอบจากความยาวของแพ็กเกจ เมื่อทราบแล้วว่าเป็นแพ็กเกจของการลงทะเบียน เซิร์ฟเวอร์ก็จะทำการสร้างค่า ID ขึ้นมาแล้วก็จะทำการบันทึกค่า IP Port (ไคลเอนต์เป็นตัวส่งมา) ID และบันทึกค่าคะแนนเริ่มต้นให้กับ ID นั้นๆ แล้วเซิร์ฟเวอร์ก็จะทำการส่งค่า ID ไปให้กับ ไคลเอนท์



รูปที่ 5.4 การลงทะเบียน

5.3.2 การสร้างศัตรูขึ้นมา

เมื่อเซิร์ฟเวอร์เปิดขึ้นก็จะมีการสร้างศัตรูขึ้นมาโดยการออกแบบของแอพพลิเคชั่นนี้ได้ออกแบบไว้ว่าต้องมีศัตรูคงที่ในแผนที่ คือ 20 ตัว โดยเริ่มต้นจะสร้างศัตรูด้วย gen_perm() โดยในฟังก์ชั่นนี้จะมีการเรียก generateRandoms() keep_mon() และ table() โดยใน table() จะมีการเรียกอีก 1 ฟังก์ชั่น คือ generateRandoms_die() โดยเมื่อสร้างศัตรูได้แล้วซึ่งเก็บค่าไว้แล้วในตัวแปร pos_x[ ] และ pos_y[ ] โดยการสร้างศัตรูขึ้นมานี้จะมีการตรวจสอบว่าในการสร้างนี้ตัวศัตรูที่สร้างขึ้นจะไม่ชนกับตำแหน่งของกำแพง ก้อนหิน

gen_perm() เป็นฟังก์ชั่นที่ใช้ในการสร้างศัตรูขึ้นมาโดยฟังก์ชั่นนี้จะถูกเรียกเมื่อตัวเซิร์ฟเวอร์ได้ถูกเปิดขึ้นมา โดยที่ตัว gen_perm() นี้จะมีการเรียกฟังก์ชั่นต่างๆเพื่อใช้ในการเก็บตำแหน่งของศัตรูและตรวจสอบการทับกันของตำแหน่ง

generateRandoms() เป็นฟังก์ชั่นที่ทำการ random ค่าตำแหน่งของศัตรูขึ้นมาทั้ง 20 ตำแหน่งโดยเมื่อ random ขึ้นมาแล้วจะถูกเก็บไว้เป็นสตริงที่มีความยาว 80 (ศัตรู 1 ตัวใช้ 4 ตำแหน่งเพื่อบอกตำแหน่งในตาราง)

keep_mon() เป็นฟังก์ชั่นที่มีไว้เพื่อทำการเก็บค่าตำแหน่งของศัตรูเก็บไว้ในตัวแปรอาร์เรย์ คือเก็บไว้ใน pos_x[ ] คือตำแหน่ง x ของศัตรู และ pos_y[ ] คือตำแหน่ง y ของศัตรู โดยในการเก็บค่าเหล่านี้ จะนำเอาค่าสตริงที่ได้จาก generateRandoms() มาทำการตัด คือ ใน 2 ตำแหน่งแรกของสตริงก็จะให้เก็บไว้ใน pos_x[0] และ 2 ตำแหน่งถัดไปก็จะให้เก็บไว้ใน pos_y[0] และ 2 ตำแหน่งถัดไปก็จะเก็บไว้ใน pos_x[1] และ 2 ตำแหน่งถัดไปก็จะให้เก็บไว้ใน pos_y[1] ทำเช่นนี้ไปเรื่อยๆจนสิ้นสุดของสตริงก็จะได้ตำแหน่งของศัตรูทั้งหมด 20 ตัว เก็บไว้ในอาร์เรย์ของ int

table() เป็นฟังก์ชั่นที่ใช้สำหรับตรวจสอบดูว่าตำแหน่งศัตรูที่สร้างขึ้นมาไปทับกับตำแหน่งอื่นหรือไม่ เช่น ทับกับตำแหน่งกำแพง ทับกับตำแหน่งก้อนหิน ถ้าทับกันก็จะให้เรียก generateRandoms_die() เพื่อทำการสร้างตำแหน่งขึ้นมาใหม่แทนที่ศัตรูตัวนั้น โดยการตรวจสอบจะใช้ while(Map[pos_x[k]][pos_y[k]] != 'o') โดย 'o' คือตำแหน่งหญ้า (สามารถให้เกิดศัตรูได้) ถ้าเข้ากรณีนี้จะทำการให้ สร้างตำแหน่งขึ้นมาใหม่แล้วทำการเก็บไว้ใน pos_x[k] และ pos_y[k] แล้วถ้าตำแหน่งที่สร้างใหม่ขึ้นมานั้นยังมีค่าเท่ากันอยู่ก็จะต้องทำการสร้างขึ้นมาใหม่

```
while(Map[pos_x[k]][pos_y[k]]!='o')
    {
      String new_mon = generateRandoms_die();
      String one = new_mon.substring(0,2);
      pos_x[k] = Integer.parseInt(one);
      String two = new_mon.substring(2,4);
      pos_y[k] = Integer.parseInt(two);
    }
```

generateRandoms_die() เป็นฟังก์ชั่นที่ใช้ในการ random ค่าศัตรูขึ้นมาใหม่โดยการ random ค่าศัตรูนั้นจะทำการ random เพียงค่า x กับ y เท่านั้น คือตำแหน่งของศัตรูเพียงตัวเดียว

5.3.3 การเริ่มเล่นของไคลเอนต์

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจจากไคลเอนต์แล้วได้รับการตรวจสอบแล้วว่าเป็นแพ็กเกจของการเริ่มต้น เซิร์ฟเวอร์ก็จะทำการสร้างตำแหน่งการเกิดให้กับตัวไคลเอนต์แล้วส่งไปพร้อมกับค่าของศัตรูที่ได้ทำการสร้างขึ้น ในการส่งครั้งนี้จะยังมีการเอาค่าของคะแนนที่ได้เก็บไว้ใน Record Store มารวมส่งไปให้กับตัวไคลเอนต์ด้วย เมื่อส่งไปให้แล้วตัวเซิร์ฟเวอร์ต้องทำการส่งค่าตำแหน่งของตัวที่เข้ามาใหม่ไปให้กับตัวไคลเอนต์อื่นๆที่ออนไลน์อยู่ให้ได้รู้ตำแหน่งของไคลเอนต์ตัวใหม่เพื่อที่จะได้ทำการเปลี่ยนค่าใน Map ให้มีค่าของไคลเอนต์ตัวใหม่ เซิร์ฟเวอร์ยังต้องส่งค่าตำแหน่งของไคลเอนต์ตัวอื่นๆ ไปให้กับตัวไคลเอนต์ที่เข้ามาใหม่เพื่อทำการแสดงตำแหน่งของไคลเอนต์ตัวอื่นๆที่ออนไลน์อยู่ด้วย

โดยการส่งตำแหน่งไคลเอนต์ตัวอื่นๆ ไปให้กับไคลเอนต์ตัวใหม่จะใช้ sendfirst() และการส่งค่าตำแหน่งของไคลเอนต์ตัวใหม่ไปให้กับไคลเอนต์ตัวอื่นๆจะใช้ sendboard()

sendfirst() เป็นฟังก์ชั่นที่จะส่งไปให้กับไคลเอนต์ตัวใหม่ที่เพิ่งเข้ามา โดยจะเริ่มจากการเปิด Record Store เพื่อดูว่ามีไคลเอนต์ตัวอื่นที่ออนไลน์อยู่หรือไม่ ถ้ามีก็จะทำการเอาค่า ID นั้นมาจากตารางเพื่อเอาค่าตำแหน่งของ ID นั้นจาก point[id] มาเพื่อส่งไปที่ตัวไคลเอนต์ตัวใหม่โดยในการส่งไปที่ไคลเอนต์ตัวใหม่นี้ก็จะไปดึงค่า IP และ Port จาก Record Store ของการลงทะเบียนเพื่อจะได้ส่งไปให้กับไคลเอนต์ตัวที่เข้ามาใหม่ได้อย่างถูกต้อง ถ้ามีไคลเอนต์ตัวอื่นๆมากกว่า 2 ตัวที่ต้องวนลูปเพื่อทำการส่ง การส่งก็จะทำการส่งเหมือนกับ ID แรกที่ส่ง

โดยการดึงค่าออกมาจาก Record จะใช้

```
len = rs.getRecord(i,recData,0);
rec = new String(recData,0,len);
```

เมื่อทำแล้วจะได้ Record ในตำแหน่งที่ i ออกมาเพื่อมาทำตรวจสอบและจะส่งด้วย

```
send(IP_new_client, position_another_client)
```

sendboard() เป็นฟังก์ชั่นที่ใช้สำหรับส่งตำแหน่งของไคลเอนต์ตัวใหม่ไปให้กับไคลเอนต์ตัวอื่นๆ ที่ออนไลน์อยู่ โดยเริ่มจากการเปิด Record Store เพื่อดูว่ามีไคลเอนต์ตัวอื่นๆที่ออนไลน์อยู่หรือไม่ ถ้ามีก็จะทำการเอาค่า ID เหล่านั้นมาใช้เพื่อไปดึงค่า IP และ Port ของแต่ละ ID เพื่อที่จะทำการส่ง ถ้ามีไคลเอนต์ตัวอื่นๆ มากกว่า 2 ตัวก็จะทำการวนลูปเพื่อทำการส่ง

ในการส่งนี้ใช้

```
send(IP_anther_client, position_new_client);
```

5.3.4 การเดินและการหันของไคลเอนต์

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจจากไคลเอนต์ แล้วได้รับการตรวจสอบแล้วว่าเป็นแพ็กเกจของการเดินและการหัน เซิร์ฟเวอร์ก็จะต้องทำการตัดค่าเหล่านี้เพื่อจะเอาค่า ID ที่ส่งมาไปทำการเก็บตำแหน่งของตัวไคลเอนต์ เมื่อทำการเก็บตำแหน่งแล้วก็จะต้องทำการส่งค่าตำแหน่งนี้ไปให้กับไคลเอนต์ตัวอื่นๆ

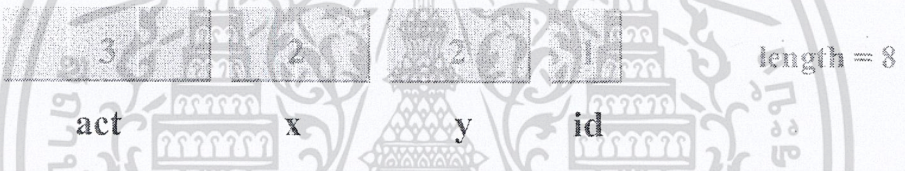
```
if (stringpoint.length()==8) {
        StringToInt();
        keeppoint();
        sendbroad();
      }
```

เมื่อ stringpoint คือ ค่าที่รับมาจากไคลเอนต์ซึ่งถ้ามีความยาวเท่ากับ 8 แสดงว่าเป็นแพ็กเกจของการเดินและการหันเมื่อเข้ามาแล้วจะไปเข้า StringToInt() เพื่อทำการแปลงค่าเป็น int เสร็จแล้วก็จะไปเข้า keeppoint() เพื่อทำการเก็บตำแหน่ง เสร็จแล้วก็จะไปเข้า sendboard() เพื่อทำการส่งค่าตำแหน่งไปให้กับตัวอื่นๆ

StringToInt() เป็นฟังก์ชั่นที่การเอาค่าที่ได้รับเข้ามา มาทำการดึงเอาค่า ID ของ ไคลเอนท์ เพื่อมาทำการเก็บไว้ในตัวแปร idstr

keeppoint() เป็นฟังก์ชั่นที่เก็บค่าที่รับเข้ามาไว้ใน point[idstr]

sendboard() เป็นฟังก์ชั่นเดียวกันกับในหัวข้อ 5.3.3 คือจะส่งค่าตำแหน่ง(แพ็กเกจที่มีความยาวเท่ากับ 8)ไปให้กับไคลเอนต์ตัวอื่นๆ ที่ออนไลน์อยู่



รูปที่ 5.5 แพ็กเกจในกรณีการเดินและการหันของเซิร์ฟเวอร์ที่จะส่ง

act หมายถึง ค่าที่แสดงการกระทำของตัวผู้เล่น
x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่น
y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่น
id หมายถึง ค่า id ของตัวผู้เล่น

5.3.5 การฆ่าศัตรู

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจจากไคลเอนต์ แล้วได้รับการตรวจสอบแล้วว่าเป็นแพ็กเกจของการฆ่าศัตรู เซิร์ฟเวอร์ก็จะเข้าฟังก์ชั่นของการฆ่าศัตรู

```
if(stringpoint.length()==9){
        StringToInt();
        if (stringpoint.substring(0,1).compareTo("1") == 0)
            ManageKill();
}
```

เมื่อ stringpoint คือ ค่าที่รับมาจากไคลเอนต์ซึ่งถ้ามีความยาวเท่ากับ 9 แล้วจะเข้าฟังก์ชั่น StringToInt() เพื่อทำการแปลงค่าเป็น int แล้วก็จะไปทำการตรวจสอบในตำแหน่งแรกว่ามีค่าเท่ากับ "1" หรือไม่ ถ้าเท่าแสดงว่าเป็นแพ็กเกจของการฆ่าศัตรู ก็จะไปเข้า ManageKill()

Stringpoint() เป็นฟังก์ชั่นเดียวกันกับในหัวข้อ 5.3.4 ซึ่งจะเก็บ ID ตัวที่ฆ่าศัตรูไว้ที่ตัวแปร idstr

ManageKill() เป็นฟังก์ชั่นที่จัดการกับการฆ่าศัตรูของไคลเอนต์ เมื่อเข้าฟังก์ชั่นนี้แล้วก็จะทำการตัดค่าต่างๆที่รับมาเก็บไว้ แล้วทำการหาตำแหน่งของศัตรูที่ต้องการฆ่า โดยจะดูจากตำแหน่งของไคลเอนต์จากที่รับมา แล้วดูว่าหันหน้าฆ่าไปทางใด จะได้รู้ตำแหน่งศัตรูที่ต้องการฆ่า เมื่อรู้ตำแหน่งแล้ว ก็จะทำการตรวจสอบดูว่าตำแหน่งที่ต้องการฆ่านั้นมีศัตรูอยู่จริงหรือไม่ ถ้ามีอยู่จริงแสดงว่าสามารถฆ่าได้ ก็จะทำการเปลี่ยนตำแหน่งนั้นให้เป็น Item แล้วก็ทำการสร้างศัตรูขึ้นมาใหม่ แล้วก็ทำการส่งแพ็กเกจที่มีความยาวเท่ากับ 12 ไปให้กับไคลเอนต์ทุกๆตัวเพื่อทำการแก้ไข Map

ทำการตรวจสอบว่าค่าตำแหน่งศัตรูที่ต้องการฟันมีอยู่จริงหรือไม่ด้วย

```
if((pos_x[j] == mon_kill_x)&&(pos_y[j] == mon_kill_y))
```

ทำการเปลี่ยนค่าให้เป็น Item ด้วย

```
Map[mon_kill_x][mon_kill_y] = 't';
```

ทำการส่งให้ไปให้ไคลเอนต์ทุกๆตัวด้วย

```
send(IP_all_client, string_managekill);
```

รูปที่ 5.6 แพ็กเกจในกรณีการฆ่าศัตรูของเซิร์ฟเวอร์ที่จะส่ง

new_monster_position	หมายถึง ค่าตำแหน่ง x และ y ของศัตรูตัวใหม่ที่ได้ random ขึ้นมา
id_mon	หมายถึง ค่า id ของศัตรูตัวที่ random ขึ้นมาใหม่
Item_position	หมายถึง ค่าตำแหน่ง x และ y ของ Item ที่ได้สร้างขึ้นมา
id_item	หมายถึง รหัสของ Item
id_killer	หมายถึง ค่า id ของตัวที่ฆ่าศัตรูได้

5.3.6 การเก็บ Item

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจจากไคลเอนต์ แล้วได้รับการตรวจสอบแล้วว่าเป็นแพ็กเกจของการฆ่าเก็บ Item เซิร์ฟเวอร์ก็จะเข้าฟังก์ชั่นของการเก็บ Item

```
if(stringpoint.length()==9){
    StringToInt();
```

```
if (stringpoint.substring(0,1).compareTo("2") == 0)
        ManageItem();
}
```

เมื่อ stringpoint คือ ค่าที่รับมาจากไคลเอนต์ซึ่งถ้ามีความยาวเท่ากับ 9 แล้วจะเข้าฟังก์ชั่น StringToInt() เพื่อทำการแปลงค่าเป็น int แล้วก็จะไปทำการตรวจสอบในตำแหน่งแรกว่ามีค่าเท่ากับ "2" หรือไม่ ถ้าเท่าแสดงว่าเป็นแพ็กเกจของการเก็บ Item ก็จะไปเข้า ManageItem();

Stringpoint() เป็นฟังก์ชั่นเดียวกันกับในหัวข้อ 5.3.4 ซึ่งจะเก็บ ID ไคลเอนต์ตัวที่ทำการเก็บ Item ไว้ที่ตัวแปร idstr

ManageItem() เป็นฟังก์ชั่นที่จัดการกับการเก็บ Item ของไคลเอนต์ เมื่อเข้าฟังก์ชั่นนี้แล้ว ก็จะทำการตัดค่าต่างๆที่รับมาเก็บไว้ แล้วทำการหาตำแหน่งของ Item ที่ต้องการเก็บ โดยจะดูจากตำแหน่งของไคลเอนต์จากที่รับมา แล้วดูว่าหันหน้าเดินไปทางใด จะได้รู้ตำแหน่ง Item ที่ต้องการเก็บ เมื่อรู้ตำแหน่งแล้ว ก็จะทำการตรวจสอบดูว่าตำแหน่งที่ต้องการเก็บนั้นมี Item อยู่จริงหรือไม่ ถ้ามีอยู่จริง แสดงว่าสามารถเก็บ Item นั้นได้ ก็จะทำการเปลี่ยนตำแหน่งนั้นให้ไปเป็นหญ้าเหมือนเดิม แล้วก็ทำการส่งแพ็กเกจที่มีความยาวเท่ากับ 7 ไปให้กับไคลเอนต์ทุกๆตัวเพื่อทำการแก้ไข Map

ทำการตรวจสอบว่าค่าตำแหน่ง Item ที่ต้องการเก็บว่ามีอยู่จริงหรือไม่ด้วย

```
if(item[j] == keep_item_xy)
```

ทำการเปลี่ยนค่าให้เป็น Item ด้วย

```
Map[keep_item_x][keep_item_y] = 'o';
```

ทำการส่งให้ไปให้ไคลเอนต์ทุกๆตัวด้วย

```
send(IP_all_client, string_manageitem);
```

รูปที่ 5.7 แพ็กเกจในกรณีการเก็บ Item ของเซิร์ฟเวอร์ที่จะส่ง

x_item	หมายถึง ค่าตำแหน่ง x ของ Item
y_item	หมายถึง ค่าตำแหน่ง y ของ Item
id_item	หมายถึง ค่า id ของ Item
id_keep_item	หมายถึง ค่า id ของผู้เล่นที่เก็บ Item

5.3.7 การสู้กับผู้เล่นคนอื่น

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจจากไคลเอนต์ แล้วได้รับการตรวจสอบแล้วว่าเป็นแพ็กเกจของการสู้กับผู้เล่นคนอื่น เซิร์ฟเวอร์ก็จะเข้าฟังก์ชั่นของการสู้กับผู้เล่นคนอื่น

```
if(stringpoint.length()==9){
```

```
StringToInt();
if (stringpoint.substring(0,1).compareTo("3") == 0)
        ManageKillActor();
}
```

เมื่อ stringpoint คือ ค่าที่รับมาจากไคลเอนต์ซึ่งถ้ามีความยาวเท่ากับ 9 แล้วจะเข้าฟังก์ชั่น StringToInt() เพื่อทำการแปลงค่าเป็น int แล้วก็จะไปทำการตรวจสอบในตำแหน่งแรกว่ามีค่าเท่ากับ "3" หรือไม่ ถ้าเท่าแสดงว่าเป็นแพ็กเกจของการสู้กับผู้เล่นคนอื่นก็จะไปเข้า ManageKillActor();

Stringpoint() เป็นฟังก์ชั่นเดียวกันกับในหัวข้อ 5.3.4 ซึ่งจะเก็บ ID ไคลเอนต์ตัวที่ทำการสู้กับผู้เล่นคนอื่นไว้ที่ตัวแปร idstr

ManageItem() เป็นฟังก์ชั่นที่จัดการกับการสู้กับผู้เล่น ของไคลเอนต์ เมื่อเข้าฟังก์ชั่นนี้แล้ว ก็จะทำการตัดค่าต่างๆที่รับมาเก็บไว้ แล้วทำการหาตำแหน่งของผู้เล่นคนอื่นที่ต้องการสู้ โดยจะดูจากตำแหน่งของไคลเอนต์จากที่รับมา แล้วดูว่าหันหน้าฟันไปทางใด จะได้รู้ตำแหน่งของผู้เล่นคนอื่นที่ต้องการสู้ เมื่อรู้ตำแหน่งแล้ว ก็จะทำการตรวจสอบดูว่าตำแหน่งที่ต้องการสู้นั้นมีผู้เล่นคนอื่น อยู่จริงหรือไม่ ถ้ามีอยู่จริงแสดงว่าสามารถฟันผู้เล่นผู้นั้นได้ แล้วก็ทำการส่งแพ็กเกจที่มีความยาวเท่ากับ 6 ไปให้กับไคลเอนต์ตัวที่ทำการสู้กับผู้เล่นตัวที่ถูกฟัน

ทำการตรวจสอบว่าค่าตำแหน่ง Item ที่ต้องการเก็บว่ามีอยู่จริงหรือไม่ด้วย

```
if(point[j].substring(3,7).compareTo(stringkill_actor) == 0)
```

ทำการส่งให้ไปให้ไคลเอนต์ตัวที่ทำการต่อสู้กับไคลเอนต์ตัวที่ถูกฟัน

```
send(IP_both_client, string_managekillactor);
```

รูปที่ 5.8 แพ็กเกจในกรณีการสู้กับผู้เล่นคนอื่นของเซิร์ฟเวอร์ที่จะส่ง

x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่นที่โดนฟัน

y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่นที่โดนฟัน

id หมายถึง ค่า id ของตัวที่โดนฟัน

id_killer หมายถึง ค่า id ของตัวที่ทำการฟัน

5.3.8 การออกจากการเล่น

เมื่อเซิร์ฟเวอร์ได้รับแพ็กเกจเข้ามา เมื่อตรวจสอบแล้วว่าเป็นแพ็กเกจของการออกจากการเล่น เซิร์ฟเวอร์ก็จะทำการตัดเอาค่า ID และคะแนนที่ไคลเอนต์ส่งมาแล้วทำการมาเก็บไว้ใน Record Store ของการลงทะเบียนโดยดึงค่าจาก Record Store ที่มี ID เดียวกันกับที่ไคลเอนต์ส่งมา แล้ว

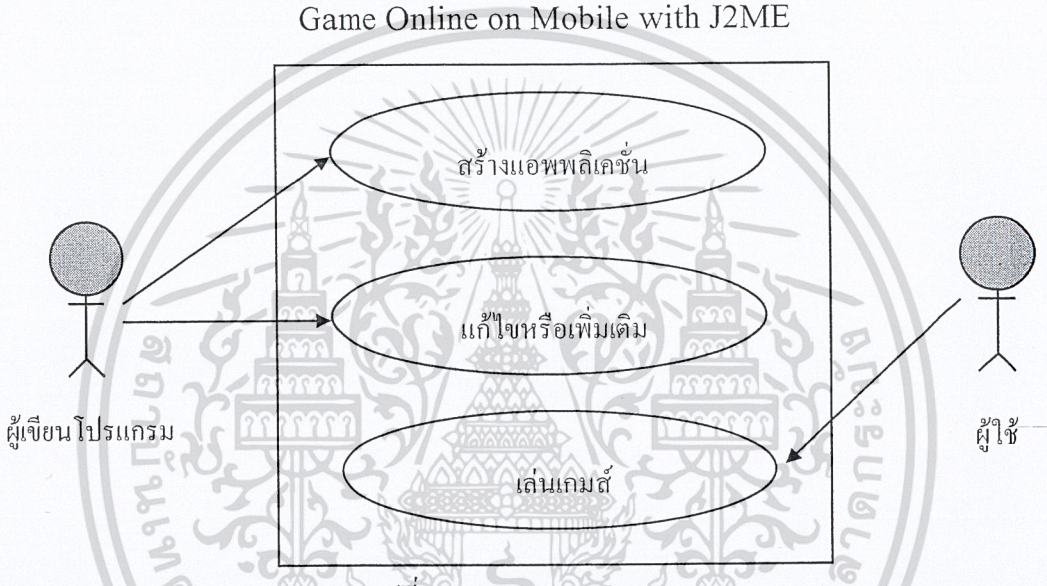
ทำการบันทึกใหม่ โดยมีค่าคะแนนที่เปลี่ยนไป เมื่อทำการบันทำแล้วตัวเซิร์ฟเวอร์ก็จะทำการส่งค่าตำแหน่งที่เก็บไว้ใน point[id] ไปให้กับไคลเอนต์ตัวอื่นๆที่ยังออนไลน์อยู่

ทำการส่งให้ไปให้ไคลเอนต์ตัวอื่นๆที่ยังออนไลน์อยู่ด้วย

send(IP_client_out, positon_client_out)

5.4 ไดอะแกรมต่างๆของตัวแอพพลิเคชั่น

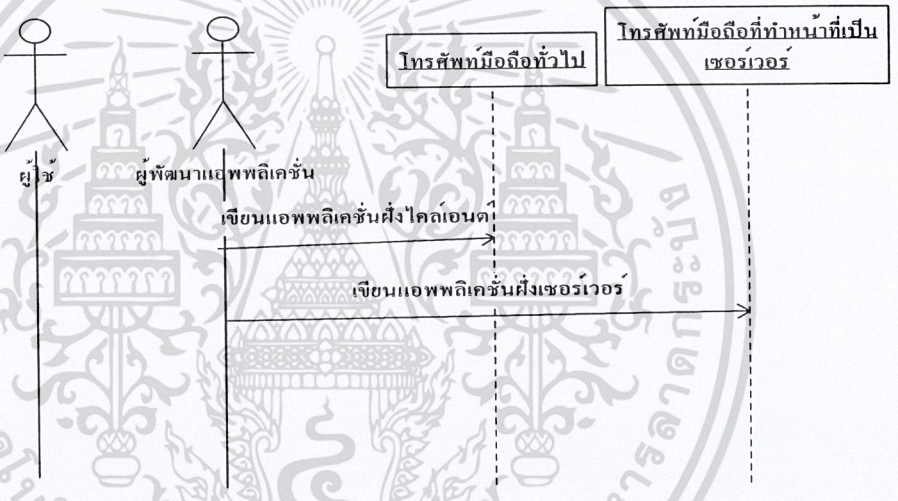
5.4.1 Use Case Diagram



รูปที่ 5.9 Use Case Diagram

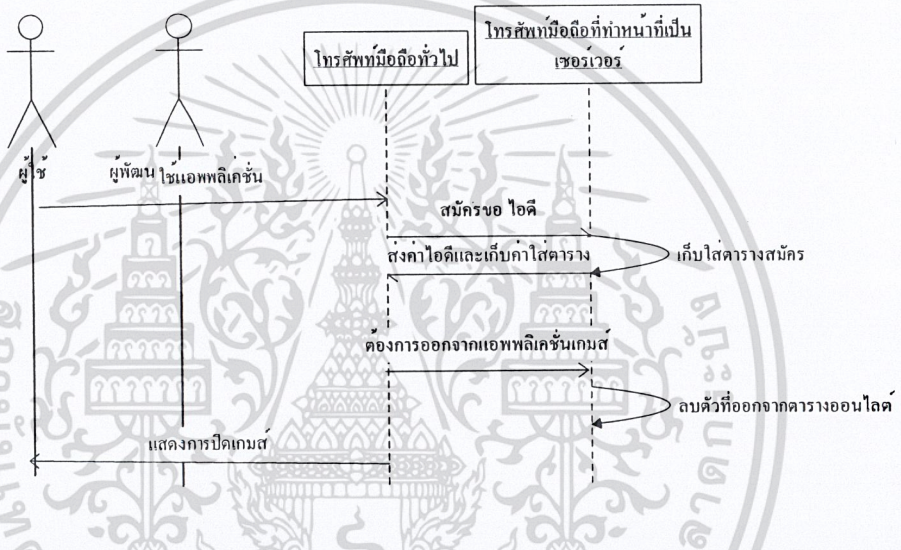
5.4.2 Sequence Diagram

5.4.2.1 การพัฒนาแอพลิเคชั่น



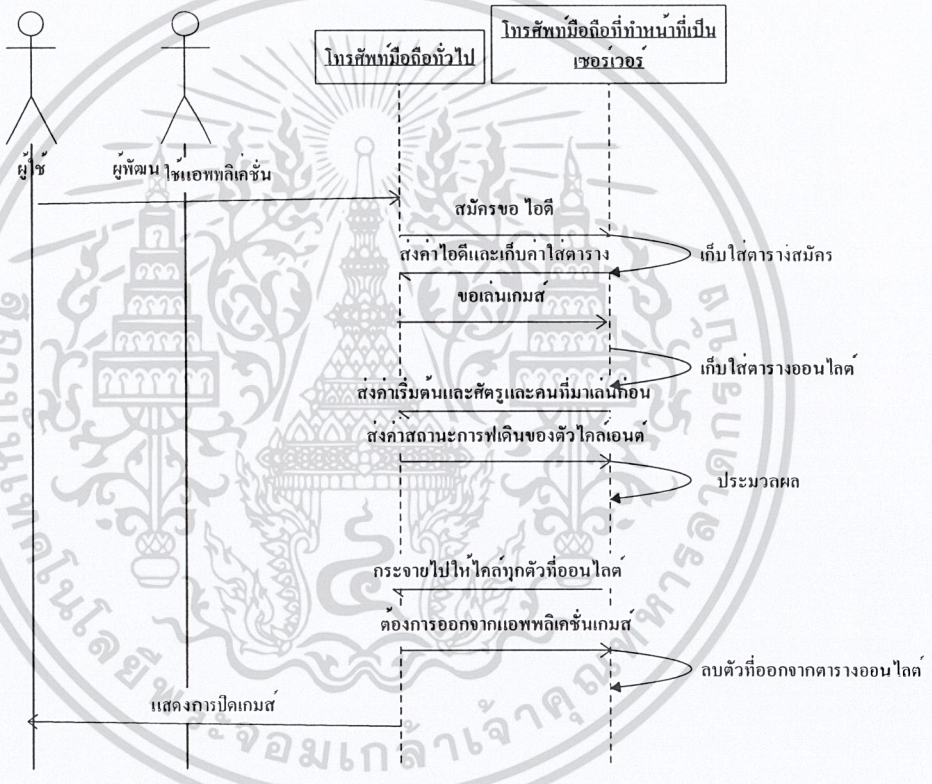
รูปที่ 5.10 Sequence Diagram ของการพัฒนาแอพพลิเคชั่น

5.4.2.2 การลงทะเบียน



รูปที่ 5.11 Sequence Diagram ของการลงทะเบียน

5.4.2.3 การฆ่าศัตรู

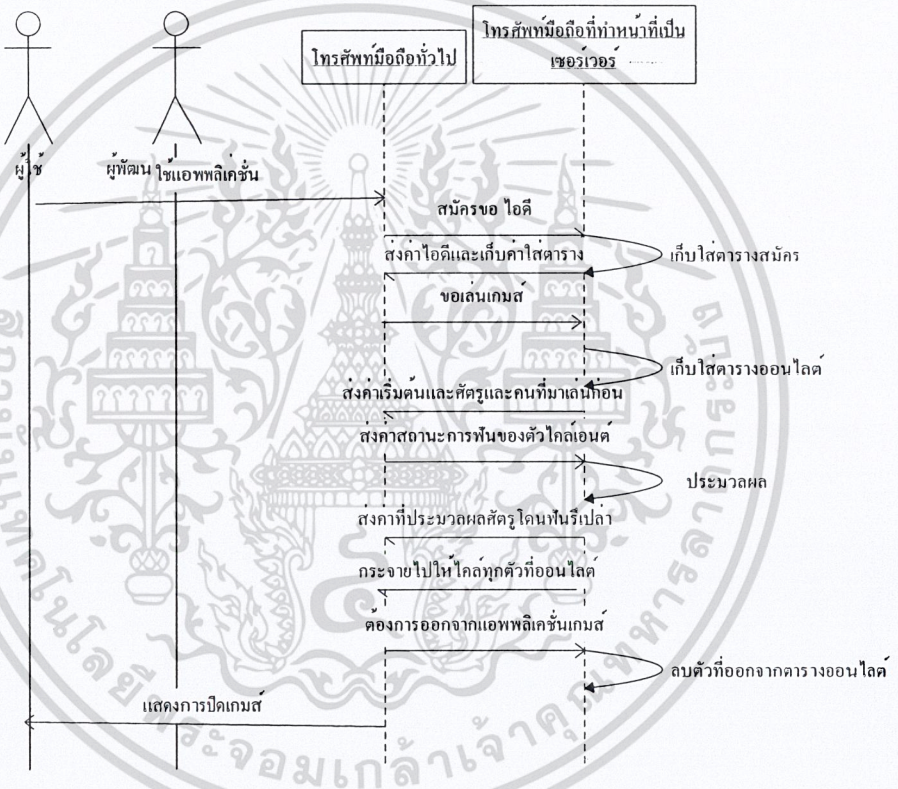


รูปที่ 5.12 Sequence Diagram ของการฆ่าศัตรู

5.4.2.4 การเก็บ Item

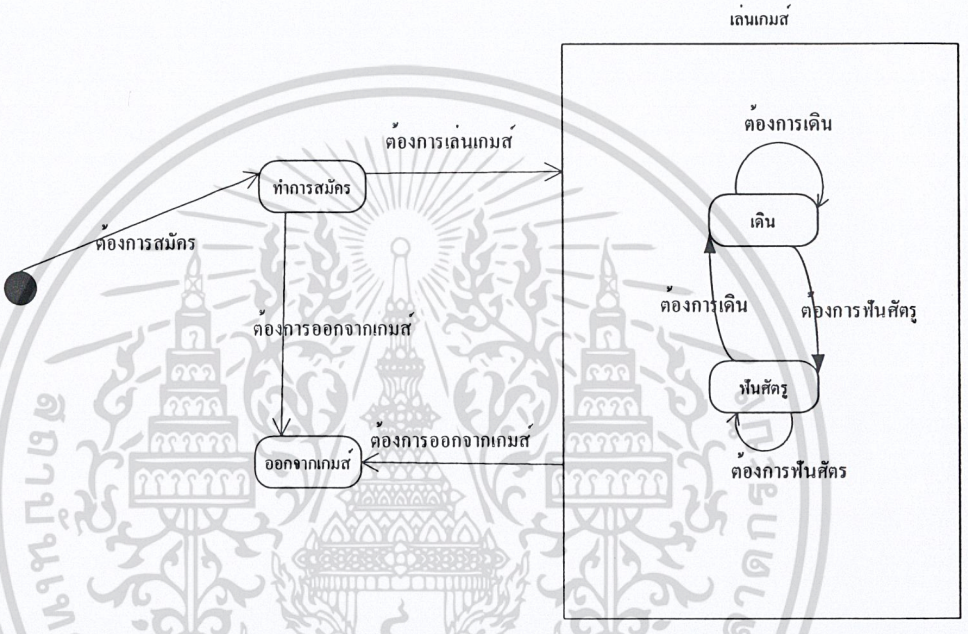
รูปที่ 5.13 Sequence Diagram ของการเก็บ Item

5.4.2.5 การสู้กับผู้เล่นคนอื่น



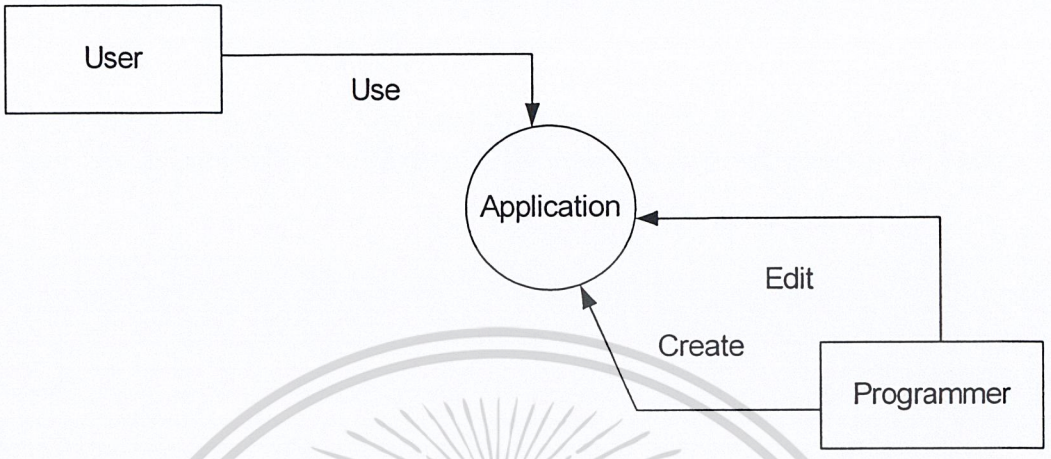
รูปที่ 5.14 Sequence Diagram ของการสู้กับผู้เล่นคนอื่น

5.4.3 State Chart Diagram



รูปที่ 5.15 State Chart Diagram

5.4.4 Context Diagram



รูปที่ 5.16 Context Diagram

บทที่ 6
การออกแบบตัวแอพพลิเคชั่นฝั่งไคลเอนต์

6.1 รูปแบบโดยทั่วไปของฝั่งไคลเอนต์

การออกแบบฝั่งไคลเอนต์นี้จะมีอะไรที่พิเศษกว่าทางเซิร์ฟเวอร์ คือจะมีฟังก์ชั่นที่เพิ่มขึ้นมาเพื่อใช้สำหรับการวาดซึ่งการวาดนี้จะเป็นการวาด Map ในตำแหน่งที่ต้องการซึ่งได้อธิบายไว้บ้างแล้วในบทที่ 5 ในหัวข้อ 5.2

6.2 การทำงานฝั่งไคลเอนต์และฟังก์ชั่นต่างๆ

การทำงานในฝั่งไคลเอนต์ได้มีการทำงานหลายอย่างซึ่งเป็นการทำงานที่ต้องส่งข้อมูลมาไปที่เซิร์ฟเวอร์เพื่อให้เซิร์ฟเวอร์ไปดำเนินการต่างๆ แล้วก็ต้องทำการรอรับข้อมูลจากเซิร์ฟเวอร์เพื่อมาทำการเปลี่ยนแปลงค่าต่างๆ โดยในตัวไคลเอนต์เองจะมีหลายการทำงานแบ่งออกเป็น

6.2.1 การลงทะเบียน

เมื่อเริ่มเล่นฝั่งไคลเอนต์จะยังไม่มีค่า ID ซึ่งทำให้ไม่สามารถเริ่มเล่นได้ ทำให้ทางไคลเอนต์ต้องทำการลงทะเบียนก่อน เมื่อเลือกการลงทะเบียน ก็จะเข้าสู่หน้าต่างให้ป้อน IP และ Port ลงไปที่ TextField แล้วกดส่ง เมื่อกดส่งแล้วตัวไคลเอนต์จะทำการส่งค่า IP + Port ไปที่เซิร์ฟเวอร์

เมื่อกดส่งแล้วไคลเอนต์จะทำการเปิดการรับข้อมูล โดยเปิดการรับข้อมูลตามที่ได้ทำการป้อนใน TextField เพื่อที่จะรอรับแพ็กเกจตอบกลับจากเซิร์ฟเวอร์ได้

เมื่อไคลเอนต์ได้รับแพ็กเกจมาจากเซิร์ฟเวอร์ซึ่งเซิร์ฟเวอร์จะส่งค่าของ ID มาให้ ตัวไคลเอนต์ต้องทำการเก็บค่าเหล่านี้ไว้ในตัวแปร โดยข้อมูลที่ต้องบันทึกคือค่า IP Port และ ID

6.2.2 การเริ่มต้นของไคลเอนต์

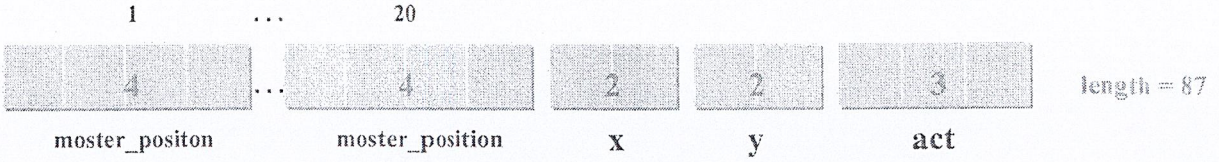
การเริ่มต้นของไคลเอนต์จะไม่สามารถเริ่มต้นได้ถ้ายังไม่มีค่า ID (ยังไม่ได้ทำการลงทะเบียน) ถ้าทำการลงทะเบียนแล้วจะสามารถเริ่มเล่นได้ โดยไคลเอนต์จะทำการดึงค่า IP และ Port มาทำการเปิดรับข้อมูล แล้วก็ดึงค่าของ ID มาเก็บไว้ที่ตัวแปร idreal

เมื่อเก็บค่าแล้วก็จะเป็นการส่งข้อมูลไปหาเซิร์ฟเวอร์เพื่อบอกให้เซิร์ฟเวอร์รู้ว่าไคลเอนต์ได้ทำการเข้ามาเริ่มเล่นแล้ว

เมื่อได้รับแพ็กเกจจากเซิร์ฟเวอร์ก็จะทำการเก็บค่าที่รับเข้ามาไว้ โดยแพ็กเกจที่รับเข้ามาจะมีความยาวเท่ากับ 87 แล้วทำการเก็บค่าเหล่านี้ไว้ใน pos_x[ ] pos_y[ ] และ actorX และ actorY

ในการเก็บค่าของ actorX actorY นั้นทำโดยการตัดเอาค่า 4 ตัวสุดท้ายของแพ็กเกจที่รับเข้ามาแล้วทำการแบ่งเป็นอย่างละสอง สองตัวแรกเก็บไว้ที่ actorX ส่วนสองตัวหลังเก็บไว้ที actorY เสร็จแล้วก็ทำการตัดค่าแพ็กเกจที่รับเข้ามาให้เหลือแค่ 80 ตัว

ในการเก็บค่าของตัวศัตรูนั้นจะใช้ keep_mon()



รูปที่ 6.1 แพ็กเกจในกรณีเริ่มต้นของไคลเอนต์ที่ได้รับ

monster_position หมายถึง ค่าตำแหน่งของศัตรู

x หมายถึง ค่าตำแหน่ง x ของผู้เล่นที่เซิร์ฟเวอร์เป็นผู้กำหนดให้

y หมายถึง ค่าตำแหน่ง y ของผู้เล่นที่เซิร์ฟเวอร์เป็นผู้กำหนดให้

act หมายถึง ค่าการกระทำของตัวผู้เล่นที่เซิร์ฟเวอร์เป็นผู้กำหนดให้

keep_mon() เป็นฟังก์ชั่นที่มีไว้เพื่อทำการเก็บค่าตำแหน่งของศัตรูเก็บไว้ในตัวแปรอาร์เรย์ คือเก็บไว้ใน pos_x[ ] คือตำแหน่ง x ของศัตรู และ pos_y[ ] คือตำแหน่ง y ของศัตรู โดยในการเก็บค่าเหล่านี้ จะนำเอาค่าสตริงที่ได้จากแพ็กเกจที่ถูกตัดมาเหลือ 80 ตำแหน่งมาทำการตัด คือ ใน 2 ตำแหน่งแรกของสตริงก็จะให้เก็บไว้ใน pos_x[0] และ 2 ตำแหน่งถัดไปก็จะให้เก็บไว้ใน pos_y[0] และ 2 ตำแหน่งถัดไปก็จะเก็บไว้ใน pos_x[1] และ 2 ตำแหน่งถัดไปก็จะให้เก็บไว้ใน pos_y[1] ทำเช่นนี้ไปเรื่อยๆ จนสิ้นสุดของสตริงก็จะได้ตำแหน่งของศัตรูทั้งหมด 20 ตัว เก็บไว้ในอาร์เรย์ของ int

เมื่อเก็บค่าต่างๆเสร็จสิ้นแล้วก็จะเป็นการเข้าสู่การวาดรูปที่หน้าจอเพื่อแสดงตำแหน่งของผู้เล่นด้วย

```
ColorTestCanvas a = new RagNo1.ColorTestCanvas();
a.repaint();
```

โดยที่ ColorTestCanvas เป็นคลาสที่ถูกสร้างขึ้นเพื่อใช้สำหรับวาดรูป ส่วน RagNo1 เป็นชื่อของคลาสหลักของแอพพลิเคชั่นของไคลเอนต์

6.2.3 การเดินและการหันของไคลเอนต์

เมื่อไคลเอนต์ทำการกดปุ่มทิศทางซึ่งหมายถึงการเดินหรือการหันหน้า แล้วจะเข้าสู่ส่วนของการตรวจสอบโดยแบ่งการตรวจสอบออกเป็น 4 แบบ

6.2.3.1 เมื่อกดลงก็จะทำการตรวจสอบว่าในตำแหน่งนั้นหันหน้าไปทางข้างหน้าหรือไม่ ถ้าไม่ได้หันหน้าไปทางข้างหน้าก็จะให้ ค่าการกระทำเป็นหันไปข้างหน้า แต่ถ้าหันไปทางข้างหน้าอยู่แล้วก็จะทำการตรวจสอบตำแหน่งที่อยู่ข้างล่างว่าเป็นหญ้าหรือไม่ ถ้าเป็นหญ้าแสดงว่าสามารถเดินได้ ก็จะให้เลื่อนตำแหน่งของผู้เล่นลงไปข้างล่างแล้วก็ทำการวาดรูปใหม่ แล้วส่งค่าไปให้กับเซิร์ฟเวอร์

ตรวจสอบว่าหันหน้าไปทางข้างหน้าหรือไม่ด้วย

```
if ((Map[actorX][actorY] == 'a')||(Map[actorX][actorY] == 'b'))
```

ให้ค่าการกระทำเป็นหันไปข้างหน้าด้วย

```
act = 'a'
```

ตรวจสอบตำแหน่งที่อยู่ข้างล่างว่าเป็นหญ้าหรือไม่ด้วย

if (Map[actorX][actorY+1]== 'o')

เลื่อนตำแหน่งของผู้เล่นลงไปข้างล่างด้วย

actorY = actorY + 1

6.2.3.2 เมื่อกดขึ้นก็จะทำการตรวจสอบว่าในตำแหน่งนั้นหันหลังหรือไม่ ถ้าไม่ได้หันหลังก็จะให้ ค่าการกระทำเป็นหันไปข้างหลัง แต่ถ้าหันไปทางข้างหลังอยู่แล้วก็จะทำการตรวจสอบตำแหน่งที่อยู่ข้างบนว่าเป็นหญ้าหรือไม่ ถ้าเป็นหญ้าแสดงว่าสามารถเดินได้ ก็จะให้เลื่อนตำแหน่งของผู้เล่นขึ้นไปข้างบนแล้วก็ทำการวาดรูปใหม่ แล้วส่งค่าไปให้กับเซิร์ฟเวอร์

ตรวจสอบว่าหันหน้าไปทางข้างหลังหรือไม่ด้วย

if ((Map[actorX][actorY] == 'd')||(Map[actorX][actorY] == 'e'))

ให้ค่าการกระทำเป็นหันไปข้างหลังด้วย

act = 'd'

ตรวจสอบตำแหน่งที่อยู่ข้างบนว่าเป็นหญ้าหรือไม่ด้วย

if (Map[actorX][actorY-1]== 'o')

เลื่อนตำแหน่งของผู้เล่นขึ้นไปข้างบนด้วย

actorY = actorY – 1

6.2.3.3 เมื่อกดซ้ายก็จะทำการตรวจสอบว่าในตำแหน่งนั้นหันซ้ายหรือไม่ ถ้าไม่ได้หันซ้ายก็จะให้ ค่าการกระทำเป็นหันไปทางซ้าย แต่ถ้าหันไปทางข้างซ้ายอยู่แล้วก็จะทำการตรวจสอบตำแหน่งที่อยู่ข้างซ้ายว่าเป็นหญ้าหรือไม่ ถ้าเป็นหญ้าแสดงว่าสามารถเดินได้ ก็จะให้เลื่อนตำแหน่งของผู้เล่นไปข้างซ้ายแล้วก็ทำการวาดรูปใหม่ แล้วส่งค่าไปให้กับเซิร์ฟเวอร์

ตรวจสอบว่าหันหน้าไปทางข้างซ้ายหรือไม่ด้วย

if ((Map[actorX][actorY] == 'j')||(Map[actorX][actorY] == 'k'))

ให้ค่าการกระทำเป็นหันไปข้างซ้ายด้วย

act = 'j'

ตรวจสอบตำแหน่งที่อยู่ข้างซ้ายว่าเป็นหญ้าหรือไม่ด้วย

if (Map[actorX-1][actorY]== 'o')

เลื่อนตำแหน่งของผู้เล่นไปข้างซ้ายด้วย

actorX = actorX - 1

6.2.3.4 เมื่อกดขวาก็จะทำการตรวจสอบว่าในตำแหน่งนั้นหันขวาหรือไม่ ถ้าไม่ได้หันขวาก็จะให้ ค่าการกระทำเป็นหันไปทางขวา แต่ถ้าหันไปทางข้างขวาอยู่แล้วก็จะทำการตรวจสอบตำแหน่งที่อยู่ข้างขวาว่าเป็นหญ้าหรือไม่ ถ้าเป็นหญ้าแสดงว่าสามารถเดินได้ ก็จะให้เลื่อนตำแหน่งของผู้เล่นไปข้างขวาแล้วก็ทำการวาดรูปใหม่ แล้วส่งค่าไปให้กับเซิร์ฟเวอร์

ตรวจสอบว่าหันหน้าไปทางข้างซ้ายหรือไม่ด้วย

if ((Map[actorX][actorY] == 'g')||(Map[actorX][actorY] == 'h'))

ให้ค่าการกระทำเป็นหันไปข้างซ้ายด้วย

act = 'g'

ตรวจสอบตำแหน่งที่อยู่ข้างซ้ายว่าเป็นหญ้าหรือไม่ด้วย

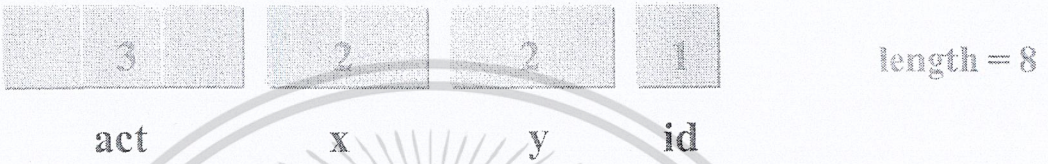
if (Map[actorX+1][actorY]== 'o')

เลื่อนตำแหน่งของผู้เล่นไปข้างซ้ายด้วย

actorX = actorX + 1

ในการวาดรูปใหม่จะใช้ repaint()

และในการส่งไปที่เซิร์ฟเวอร์จะใช้ send() โดยจะมีค่า sendoption = 0



รูปที่ 6.2 แพ็กเกจในกรณีการเดินและการหันของไคลเอนท์ที่จะส่ง

act หมายถึง ค่าที่แสดงการกระทำของตัวผู้เล่น

x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่น

y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่น

id หมายถึง ค่า id ของตัวผู้เล่น

เมื่อได้รับแพ็กเกจที่มีความยาวเท่ากับ 8 แสดงว่าได้รับแพ็กเกจเกี่ยวกับการเดินหรือการหันก็จะทำการเข้าฟังก์ชั่น StringToInt() และก็เข้าฟังก์ชั่น keeppoint()

StringToInt() เป็นฟังก์ชั่นที่เปลี่ยนค่าที่รับเข้ามาให้เป็น int โดยจะทำการเก็บค่า ID ที่ได้รับเข้ามาไว้ในตัวแปร id

kepoint() เป็นฟังก์ชั่นที่ใช้สำหรับเก็บตำแหน่งของศัตรูซึ่งจะเก็บไว้ใน point[id]

6.2.4 การฆ่าศัตรู

เมื่อทำการกดปุ่มเพื่อฆ่าศัตรู (ปุ่มฟันซึ่งเป็นปุ่มเดียวกันกับปุ่มสู้กับผู้เล่นคนอื่น) แล้วก็จะดูว่าผู้เล่นหันหน้าไปทางไหน ถ้าหันไปทางไหนก็จะให้เปลี่ยนเป็นการกระทำหันหน้าไปทางนั้นแล้วกำลังฟัน เมื่อเปลี่ยนค่าแล้วก็จะทำการวาดรูป เมื่อวาดแล้วก็ตรวจสอบดูว่าในตำแหน่งที่มันต้องการฟันนั้นมีตัวศัตรูอยู่หรือไม่ ถ้ามีก็จะทำการให้ค่า sendoption เป็น 1 เมื่อเสร็จแล้วก็จะทำการเปลี่ยนการกระทำเป็นอย่างเดิมก่อนทำการฟัน เสร็จแล้วก็จะทำการส่งค่าไปให้กับเซิร์ฟเวอร์ โดยจะแบ่งได้เป็น 4 แบบ

6.2.4.1 หันหน้าไปข้างหน้า

ตรวจสอบและให้ค่าการกระทำเป็นฟันด้วย

if ((Map[actorX][actorY] == 'a')||(Map[actorX][actorY] == 'b'))

act = 'c'

ตรวจสอบว่ามีศัตรูอยู่หรือไม่ด้วย

if(Map[actorX][actorY+1] == 'z')

6.2.4.2 หันหน้าไปข้างหลัง

ตรวจสอบและให้ค่าการกระทำเป็นฟันด้วย

if ((Map[actorX][actorY] == 'd')||(Map[actorX][actorY] == 'e'))

act = 'f'

ตรวจสอบว่ามีศัตรูอยู่หรือไม่ด้วย

f(Map[actorX][actorY-1] == 'z')

6.2.4.3 หันหน้าไปข้างซ้าย

ตรวจสอบและให้ค่าการกระทำเป็นฟันด้วย

if ((Map[actorX][actorY] == 'j')||(Map[actorX][actorY] == 'k'))

act = 'l'

ตรวจสอบว่ามีศัตรูอยู่หรือไม่ด้วย

if(Map[actorX-1][actorY] == 'z')

6.2.4.4 หันหน้าไปข้างขวา

ตรวจสอบและให้ค่าการกระทำเป็นฟันด้วย

if ((Map[actorX][actorY] == 'g')||(Map[actorX][actorY] == 'h'))

act = 'i'

ตรวจสอบว่ามีศัตรูอยู่หรือไม่ด้วย

if(Map[actorX+1][actorY] == 'z')

เมื่อเสร็จแล้วก็จะทำการส่งด้วย send() โดยจะมีค่า sendoption = 1

รูปที่ 6.3 แพ็กเกจในกรณีการฆ่าศัตรูของไคลเอนท์ที่จะส่ง

"1" หมายถึง ใส่เพื่อแบ่งให้รู้ว่าเป็นแพ็กเกจของการฆ่าศัตรู

act หมายถึง ค่าที่แสดงการกระทำของตัวผู้เล่น

x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่น

y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่น

id หมายถึง ค่า id ของตัวผู้เล่น

เมื่อได้รับแพ็กเกจที่มีความยาวเท่ากับ 12 ถ้าใช่แสดงว่าเป็นแพ็กเกจเกี่ยวกับการฆ่าศัตรู ก็จะทำการเข้าฟังก์ชั่น Killed_mon()

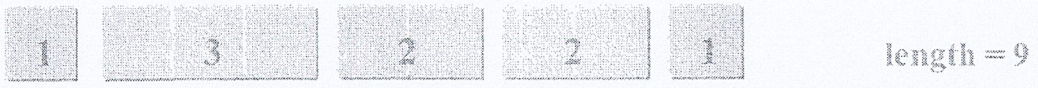
Killed_mon() จะเป็นฟังก์ชั่นที่ทำการตรวจสอบดูว่าแพ็กเกจที่รับเข้ามามีค่า ID ตรงกับค่า id ของตนเองหรือไม่ ถ้าเท่ากันแสดงว่าเราได้ฆ่าศัตรูแล้ว ก็จะทำการเพิ่มคะแนนให้กับตัวเราเอง แล้วทำการเอาค่าที่รับมานั้นซึ่งมีค่าตำแหน่งของศัตรูตัวใหม่อยู่ด้วยก็จะทำการเอาค่าตำแหน่งของศัตรูตัวใหม่มาทำการแทนที่ของศัตรูตัวเก่าด้วย แล้วยังทำการเก็บค่า Item ที่เกิดขึ้นด้วย โดยเมื่อเข้าฟังก์ชั่นนี้แล้วจะทำการแปลงค่าที่รับเข้ามาเป็นค่าต่างๆ

```
ทำการเปรียบเทียบ ID ที่รับมากับ ID ของตนเองด้วย
if (id_kill.compareTo(idreal) == 0)
ทำการเก็บค่าตำแหน่งของศัตรูตัวใหม่ด้วย
pos_x[id_mon_kill] = new_mon_x
pos_y[id_mon_kill] = new_mon_y
เก็บค่า Item ด้วย
item[it] = item_xy
```

6.2.5 การเก็บ Item

ในการเก็บ Item นั้นจะเป็นเหมือนการเดินแต่เพียงจะทำการตรวจสอบว่าในตำแหน่งที่จะเดินไปนั้นเป็นตำแหน่งของ Item หรือไม่ ถ้าเป็นก็จะทำการกำหนด sendoption ให้เท่ากับ 2 แล้วทำการส่ง

```
ตรวจสอบว่าในตำแหน่งที่จะเดินไปเป็น Item ด้วย
if(Map[actorX][actorY+1]== 't') หรือ
if(Map[actorX][actorY-1]== 't') หรือ
if(Map[actorX-1][actorY]== 't') หรือ
if(Map[actorX+1][actorY]== 't')
```



รูปที่ 6.4 แพ็กเกจในกรณีการเก็บ Item ของไคลเอนท์ที่จะส่ง

"2" หมายถึง ใส่เพื่อแบ่งให้รู้ว่าเป็นแพ็กเกจของการเก็บ Item

act หมายถึง ค่าที่แสดงการกระทำของตัวผู้เล่น

x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่น

y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่น

id หมายถึง ค่า id ของตัวผู้เล่น

เมื่อได้รับแพ็กเกจที่มีความยาวเท่ากับ 7 ถ้าใช่แสดงว่าเป็นแพ็กเกจเกี่ยวกับการเก็บ Item ก็จะทำการเข้าฟังก์ชั่น Get_item()

Get_item() จะเป็นฟังก์ชั่นที่ทำการตรวจสอบดูว่าแพ็กเกจที่รับเข้ามามีค่า ID ตรงกับค่า id ของตนเองหรือไม่ ถ้าเท่ากันแสดงว่าเราได้เก็บ Item แล้ว ก็จะทำการเพิ่มความสามารถให้เราตามแต่ Item ที่เราเก็บได้ เสร็จแล้วก็จะทำการเปลี่ยนค่าใน item[ ] ให้มีค่าเป็น 0

ทำการเปรียบเทียบ ID ที่รับมากับ ID ของตนเองด้วย

```
if (id_keep.compareTo(idreal) == 0)
```

ทำการเปลี่ยนค่าใน item[ ] ให้เท่ากับ 0 ด้วย

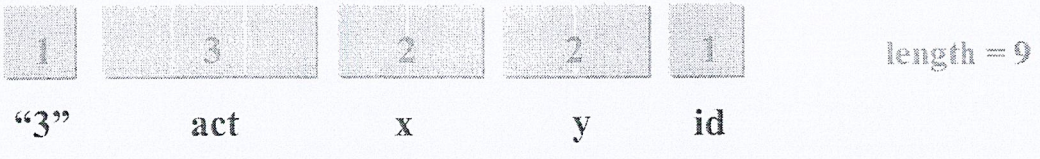
```
item[id_item] = 0
```

6.2.6 การสู้กับผู้เล่นคนอื่น

เมื่อทำการกดปุ่มเพื่อสู้กับผู้เล่นคนอื่น (ปุ่มฟันซึ่งเป็นปุ่มเดียวกันกับปุ่มฆ่าศัตรู) แล้วก็จะดูว่าผู้เล่นหันหน้าไปทางไหน ถ้าหันไปทางไหนก็จะให้เปลี่ยนเป็นการกระทำหันหน้าไปทางนั้นแล้วกำลังฟัน เมื่อเปลี่ยนค่าแล้วก็จะทำการวาดรูป เมื่อวาดแล้วก็ตรวจสอบดูว่าในตำแหน่งที่มันต้องการฟันนั้นมีผู้เล่นคนอื่นอยู่หรือไม่ ถ้ามีก็จะทำการให้ค่า sendoption เป็น 3 เมื่อเสร็จแล้วก็จะทำการเปลี่ยนการกระทำเป็นอย่างเดิมก่อนทำการฟัน เสร็จแล้วก็จะทำการส่งค่าไปให้กับเซิร์ฟเวอร์

ตรวจสอบว่าในตำแหน่งที่ต้องการฟันเป็นผู้เล่นคนอื่นหรือไม่ด้วย

```
if ((Map[actorX][actorY+1] == 'a')||(Map[actorX][actorY+1] == 'b')
||(Map[actorX][actorY+1] == 'd')||(Map[actorX][actorY+1] == 'e')
||(Map[actorX][actorY+1] == 'g')||(Map[actorX][actorY+1] == 'h')
||(Map[actorX][actorY+1] == 'j')||(Map[actorX][actorY+1] == 'k'))
```



รูปที่ 6.5 แพ็กเกจในกรณีการสู้กับผู้เล่นคนอื่นของไคลเอนท์ที่จะส่ง

"3" หมายถึง ใส่เพื่อแบ่งให้รู้ว่าเป็นแพ็กเกจของการสู้กับผู้เล่นคนอื่น

act หมายถึง ค่าที่แสดงการกระทำของตัวผู้เล่น

x หมายถึง ค่าตำแหน่ง x ของตัวผู้เล่น

y หมายถึง ค่าตำแหน่ง y ของตัวผู้เล่น

id หมายถึง ค่า id ของตัวผู้เล่น

เมื่อได้รับแพ็กเกจที่มีความยาวเท่ากับ 6 ถ้าใช่แสดงว่าเป็นแพ็กเกจเกี่ยวกับการฆ่าศัตรูก็จะทำการเข้าฟังก์ชั่น Killed_actor()

Killed_actor() จะเป็นฟังก์ชั่นที่ทำการตรวจสอบดูว่าแพ็กเกจที่รับเข้ามาซึ่งจะมีค่า ID อยู่ 2 ค่าโดยจะเป็น ID ของผู้ฟันกับ ID ของผู้ถูกฟัน โดย ID ของตนเองตรงกับ ID ของผู้ฟันก็จะไปเพิ่มคะแนนให้กับตัวเอง แต่ถ้า ID ของตนเองตรงกับ ID ของผู้ถูกฟันก็จะไปลดพลังชีวิตของตนเอง

ตรวจสอบว่า ID ของตนเองตรงกับ ID ของผู้ฟัน แล้วทำการเพิ่มคะแนนด้วย

```
if (id_kill.compareTo(idreal) == 0)
        score = score + 1
```

ตรวจสอบว่า ID ของตนเองตรงกับ ID ของผู้ถูกฟัน แล้วทำการลดพลังชีวิตด้วย

```
if (id_killed.compareTo(idreal) == 0)
        blood = blood - 2
```

6.2.7 การออกจากโปรแกรม

เมื่อไคลเอนต์ต้องการออกจากโปรแกรม ก็จะกดปุ่ม * เพื่อให้เลือกออบชั่น เมื่อกดแล้วตัวไคลเอนต์ก็จะแสดงหน้าต่างเพื่อให้เลือกออบชั่น เมื่อเลือก Exit Game แล้วก็ไคลเอนต์ก็จะทำการเก็บค่าคะแนนปัจจุบันที่อยู่ในตัวแปร score แล้วก็ค่า ID ของตัวเองเพื่อส่งไปให้กับเซิร์ฟเวอร์

6.2.8 การวาดรูป

การวาดรูปจะเริ่มด้วยจากการกำหนดค่าต่างให้ว่าให้เป็นรูปอะไรเช่นถ้าเป็น 'm' ก็ให้แทนด้วยรูปกำแพง ถ้าเป็น 'a' ก็ให้แทนด้วยรูปผู้เล่นหันหน้าไปทางข้างหน้า แล้วก็จะเข้า createmap()

createmap() เป็นฟังก์ชั่นที่ใช้สำหรับการกำหนดค่า Map ว่าให้แต่ละตำแหน่งเป็นอะไร โดยมีการกำหนดค่าของตัวผู้เล่นอยู่ในนี้ด้วย Map[actorX][actorY] = act

ต่อไปก็จะเป็นการตรวจสอบว่ามีศัตรูตัวอื่นหรือไม่ ถ้ามีก็จะทำการเปลี่ยนค่าใน Map

ตรวจสอบด้วย

```
if(point[j] != null)
```

ทำการเปลี่ยนค่าใน Map ด้วย

Map[receiveactorX][receiveactorY] = ค่าที่เก็บไว้ใน point[ ]

ต่อไปก็จะเป็นการตรวจสอบค่าของศัตรูที่เก็บตำแหน่งไว้ใน pos_x[ ] และ pos_y[ ]

ทำโดย Map[pos_x[k]][pos_y[k]] = 'z';

ต่อไปก็จะเป็นการตรวจสอบค่าของ Item ที่เก็บไว้ใน item[ ]

ทำโดย if(item[k] != 0)

Map[item[k]/100][item[k]%100] = 't';

เป็นการวาดรูปต่างๆ ที่อยู่ใน Map โดยจะทำการวาดเพียงตำแหน่งใน Map ที่ต้องการ โดยตำแหน่งที่ต้องการนั้นจะดูได้จากการทำให้ตัวผู้เล่นต้องอยู่ตรงกลางของจอ แล้วก็จะทำการดูว่าในตำแหน่งที่ต้องการวาดเป็นรูปอะไรก็ให้วาดรูปนั้นออกไป

บทที่ 7
การทดสอบการทำงาน

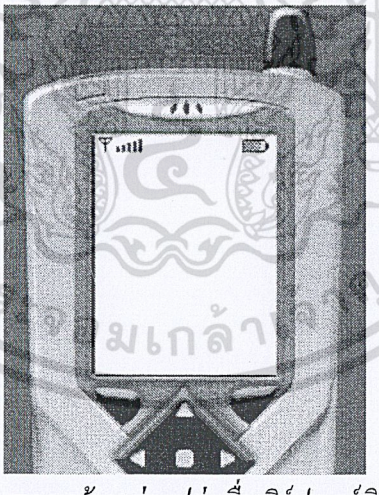
7.1 วิธีการทดสอบการทำงาน

ทำการเปิดตัวแอพพลิเคชั่นของฝั่งเซิร์ฟเวอร์ไว้ที่เครื่องคอมพิวเตอร์เครื่องหนึ่ง แล้วทำการเปิดตัวไคลเอนต์ในเครื่องคอมพิวเตอร์ที่ทำการเปิดเซิร์ฟเวอร์ไว้ และยังทำการเปิดไคลเอนต์ไว้ที่เครื่องคอมพิวเตอร์อีกเครื่องหนึ่ง

7.2 ผลการทดสอบ

ในการทดสอบได้เปิดตัวแอพพลิเคชั่นฝั่งเซิร์ฟเวอร์ไว้ที่เครื่องคอมพิวเตอร์ IP 161.246.6.6 แล้วทำการเปิดตัวไคลเอนต์ไว้ที่เครื่องนี้อีกเป็นจำนวน 1 เครื่อง และเปิดตัวไคลเอนต์ไว้ที่เครื่องคอมพิวเตอร์ IP 161.246.6.7

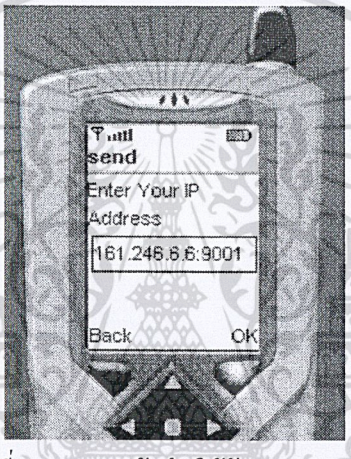
เมื่อเริ่มเปิดโปรแกรมฝั่งเซิร์ฟเวอร์ก็จะเป็นการเริ่มให้บริการซึ่งยังไม่มีการทำงานอะไร แล้วเมื่อเปิดไคลเอนต์ตัวที่ 1 ตามมา ไคลเอนต์ตัวที่ 1 ก็จะแสดงหน้าจอว่างเปล่าสักระยะ ก็จะแสดงหน้าจอที่มีภาพขึ้นมา แล้วหน้าจอนั้นก็วาดรูปใหม่ที่มีตัวศัตรูขึ้นมา แล้วเมื่อทำการเปิดตัวไคลเอนต์ตัวที่ 2 ที่อีกเครื่องหนึ่งก็จะเป็นในลักษณะเดียวกันจนเมื่อศัตรูขึ้นมา ก็จะมีตำแหน่งของตัวไคลเอนต์ตัวที่ 1 มาปรากฏอยู่ที่หน้าจอของไคลเอนต์ตัวที่ 2 แล้วในไคลเอนต์ตัวที่ 1 ก็มีไคลเอนต์ตัวที่ 2 มาปรากฏที่หน้าจอ โดยหน้าจอที่เริ่มแสดงเป็นแบบ 5 x 5 block เริ่มการทดสอบการกระทำต่างๆ



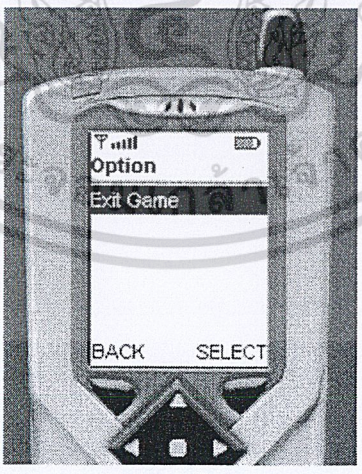
รูปที่ 7.1 แสดงหน้าจอว่างเปล่าเมื่อเซิร์ฟเวอร์เปิดเครื่อง

การเดินและการหัน เมื่อตัวที่ 1 ทำการเดิน โดยการกดปุ่มลง ไคลเอนต์ตัวที่ 1 ก็จะทำการเดินซึ่งจะช้ากว่าการกดเล็กน้อย ส่วนในไคลเอนต์ตัวที่สองก็แสดงหน้าจอตัวที่ 1 กำลังเดินลงโดยการแสดงนี้จะช้ากว่าการกดอย่างมาก

รูปที่ 7.2 แสดงหน้าต่างให้เลือก Register กับ Play

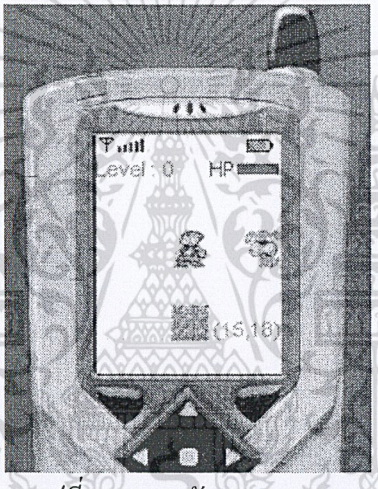


รูปที่ 7.3 แสดงหน้าต่างให้ป้อน IP Address

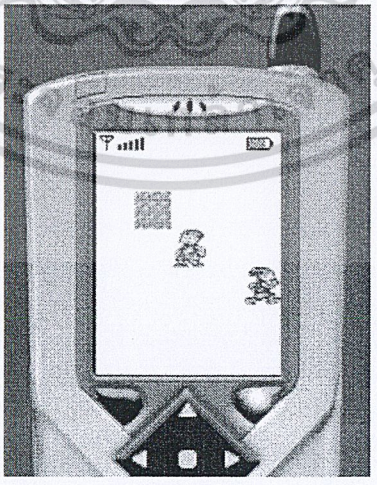


รูปที่ 7.4 แสดงหน้าต่างเมื่อให้เลือก Exit Game

รูปที่ 7.5 แสดงการเปิดเครื่องของไคลเอนต์ ขนาด 5x5

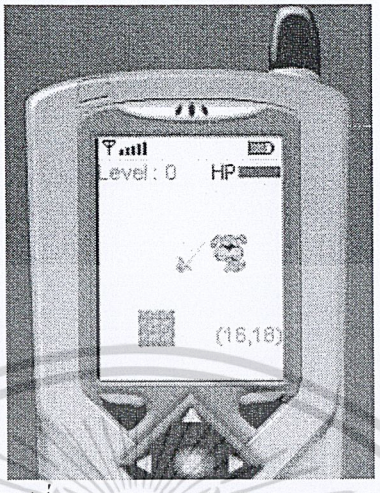


รูปที่ 7.6 แสดงศัตรู ขนาด 5x5



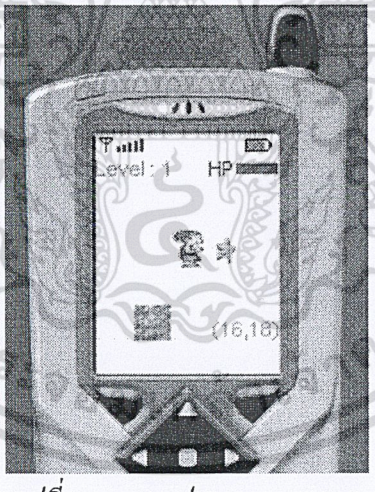
รูปที่7.7 แสดงการเข้ามาของไคลเอนต์อีกตัว ขนาด 5x5

การฟันศัตรู เมื่อตัวที่ 1 ทำการฟันศัตรูรูปก็จะเปลี่ยนไปเป็นการฟันแล้วก็เปลี่ยนไปเป็นรูปอย่างเดิมก่อนฟันพร้อมกับศัตรูได้หายไปแต่เปลี่ยนเป็น มี Item ขึ้นมาแทน แล้วคะแนนของตัวที่ 1 ก็เพิ่มขึ้น โดยในการแสดงที่หน้าจอนั้นจะช้ามากกว่าจะเปลี่ยนไปแต่ละรูป



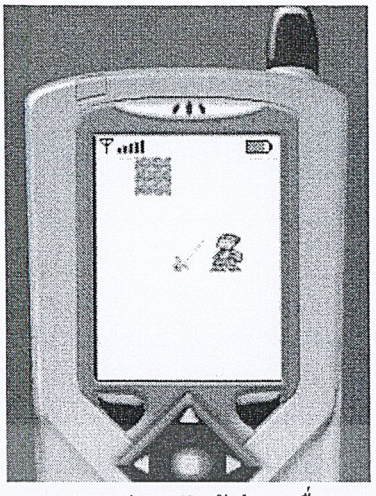
รูปที่ 7.8 แสดงการฟันศัตรู ขนาด 5x5

การเก็บ Item เมื่อตัวที่ 1 ทำการเก็บ Item ก็จะทำการเดินไปทับที่ตำแหน่ง Item แล้วคะแนนของตัวที่ 1 เพิ่มขึ้นพร้อมกับที่เก็บพลังชีวิตเพิ่มขึ้น โดยในการเพิ่มค่าคะแนนและที่เก็บพลังชีวิตที่แสดงที่หน้าจอก็จะช้ามาก



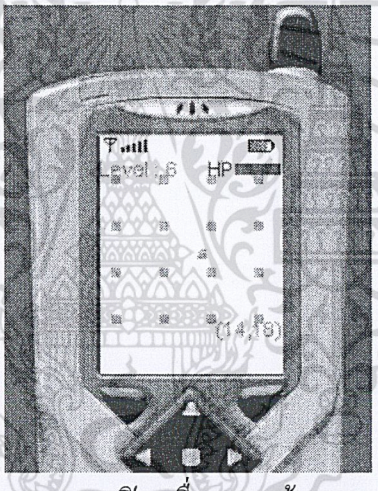
รูปที่ 7.9 แสดงรูป Item ขนาด 5x5

การสู้กับผู้เล่นคนอื่น เมื่อตัวที่ 1 ทำการฟันผู้เล่นตัวที่ 2 แล้ว ไคลเอนต์ตัวที่ 1 ก็จะเปลี่ยนไปเป็นรูปการฟันแล้วก็เปลี่ยนไปเป็นรูปอย่างเดิม แล้วสักพักคะแนนของตัวที่ 1 ก็เพิ่มขึ้น แล้วตัวที่ 2 พลังชีวิตก็จะลดลง โดยในการกระทำแต่ละการกระทำจะช้ามาก

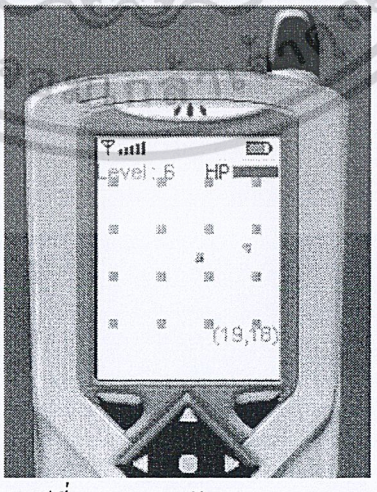


รูปที่ 7.10 แสดงรูปการฟันผู้เล่นคนอื่น ขนาด 5x5

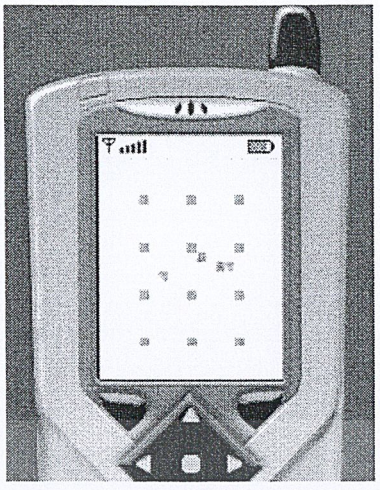
ไคลเอนต์ทั้งสองตัวได้กดเปลี่ยนไปแสดงหน้าจอที่มีขนาด 19 x 20 แล้วก็เริ่มทำการทดสอบการกระทำต่างๆ



รูปที่ 7.11 แสดงการเปิดเครื่องของหน้าจอ ขนาด 19x20



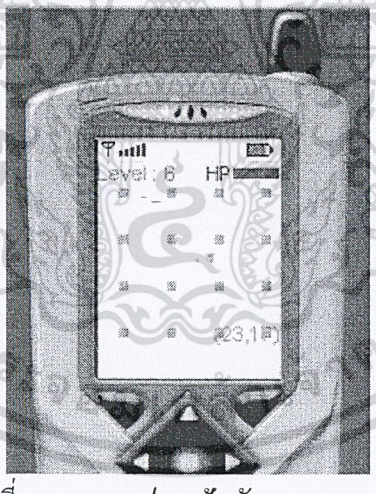
รูปที่ 7.12 แสดงศัตรู ขนาด19x20



รูปที่ 7.13 แสดงการเข้ามาของไคลเอนต์อีกตัว ขนาด 19x20

การเดินและการหัน เมื่อตัวที่ 1 ทำการเดิน โดยการกดปุ่มลง ไคลเอนต์ตัวที่ 1 ก็จะทำการเดินซึ่งจะช้ากว่าการกดเล็กน้อย ส่วนในไคลเอนต์ตัวที่สองก็แสดงหน้าจอตัวที่ 1 กำลังเดินลงโดยการแสดงนี้จะช้ากว่าการกดซึ่งเมื่อเปรียบเทียบแล้วเร็วกว่าในการแสดงหน้าจอแบบ 5 x 5 เป็นอย่างมาก

การฟันศัตรู เมื่อตัวที่ 1 ทำการฟันศัตรูรูปก็จะเปลี่ยนไปเป็นการฟันแล้วก็เปลี่ยนไปเป็นรูปอย่างเดิมก่อนฟันพร้อมกับศัตรูได้หายไปแต่เปลี่ยนเป็น มี Item ขึ้นมาแทน แล้วคะแนนของตัวที่ 1 ก็เพิ่มขึ้น โดยในการแสดงที่หน้าจอนั้นจะช้าเล็กน้อยกว่าจะเปลี่ยนไปแต่ละรูป

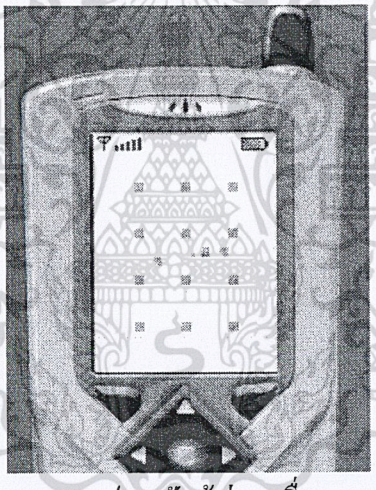


รูปที่ 7.14 แสดงรูปการฟันศัตรู ขนาด 19x20

การเก็บ Item เมื่อตัวที่ 1 ทำการเก็บ Item ก็จะทำการเดินไปทับที่ตำแหน่ง Item แล้วคะแนนของตัวที่ 1 เพิ่มขึ้นพร้อมกับที่เก็บพลังชีวิตเพิ่มขึ้น โดยในการเพิ่มค่าคะแนนและที่เก็บพลังชีวิตที่แสดงที่หน้าจอก็จะช้าเล็กน้อย

รูปที่ 7.15 แสดงรูป Item ขนาด 19x20

การสู้กับผู้เล่นคนอื่น เมื่อตัวที่ 1 ทำการฟันผู้เล่นตัวที่ 2 แล้ว ไคลเอนต์ตัวที่ 1 ก็จะเปลี่ยนไปเป็นรูปการฟันแล้วก็เปลี่ยนไปเป็นรูปอย่างเดิม แล้วสักพักคะแนนของตัวที่ 1 ก็เพิ่มขึ้น แล้วตัวที่ 2 พลังชีวิตก็จะลดลง โดยในการกระทำแต่ละการกระทำจะช้าเล็กน้อย



รูปที่ 7.16 แสดงรูปการฟันผู้เล่นคนอื่น ขนาด 19x20

เมื่อตัวที่ทำการกดปุ่มเดินเร็วๆจะทำให้หน้าจอของตัวที่ 1 แสดงภาพการเลื่อนตำแหน่งแบบเร็วๆ ไปที่ตำแหน่งปลายทาง ที่หน้าจอตัวที่ 2 แสดงภาพการเคลื่อนที่ของตัวที่ 1 ไม่ทันโดยไม่ได้ไปอยู่ในตำแหน่งที่ตัวที่ 1 ต้องอยู่จริงๆ

บทที่ 8
บทวิจารณ์และสรุป

8.1 สรุปผลการทดสอบ

เมื่อทำการทดสอบแล้วจากหลายๆการกระทำได้พบว่าในการ แสดงผลที่หน้าจอนั้นจะเห็นได้ว่าในการแสดงผลขนาด 5x5 จะช้ากว่าการแสดงผลที่การแสดงผลขนาด 19x20 กล่าวได้ว่าในการแสดงผลที่ต้องใช้รูปใหญ่จะมีผลทำให้การแสดงผลที่หน้าจอช้า

ในการรับข้อมูลผ่านทางเครือข่ายเมื่อมีการกดปุ่มหลายๆครั้งติดต่อกันอย่างรวดเร็วจะทำให้ในตัวอีกตัวหนึ่งแสดงตำแหน่งของตัวที่ทำการกดได้อย่างไม่ถูกต้อง ซึ่งหมายความว่าการรับข้อมูลของเซิร์ฟเวอร์มีปัญหา เนื่องจากไม่สามารถรับแพ็กเกจบางแพ็กเกจได้ทัน จึงทำให้ส่งไปที่ไคลเอนต์อีกตัวหนึ่งได้ไม่ครบตามที่ไคลเอนต์ตัวแรกได้ส่งมา เพราะฉะนั้นแสดงว่าในการรับข้อมูลนั้นไม่สามารถรับข้อมูลได้หลายๆข้อมูลในเวลาติดๆกัน

8.2 แนวทางการพัฒนาต่อ

โดยแนวทางการพัฒนาต่อก็คือ ต้องพัฒนาการแสดงผลให้มีประสิทธิภาพมากขึ้น โดยให้สามารถแสดงการรีเฟรชของหน้าจอให้มีความเร็วที่มากขึ้น เพื่อให้ทันต่อการเปลี่ยนแปลงค่าต่างๆที่ต้องการมาแสดงที่หน้าจอ

ในเรื่องของการรับข้อมูลนั้นต้องพัฒนาให้สามารถรับแพ็กเกจที่เข้ามาให้ได้เร็วหรือควรจะทำให้รับได้ทุกๆ แพ็กเกจที่เข้ามาเพื่อที่ว่าจะได้เปลี่ยนแปลงค่าต่างๆ ที่แพ็กเกจส่งมาให้ได้อย่างถูกต้อง

ในเรื่องของกราฟิกควรจะต้องพัฒนาให้มีความสวยงามให้มากขึ้น เพื่อที่จะให้ภาพดูสวยงามมากขึ้นและดูสมจริงสมจังในการทำแต่ละการะกระทำ

ในเรื่องของเสียง ควรจะพัฒนาให้มีการจัดการเกี่ยวกับเสียงเข้ามาเกี่ยวข้องในขณะทำการเล่นเกมส์เพื่อทำให้เกมส์ดูมีความสมจริงสมจังในการกระทำแต่ละการกระทำ

บรรณานุกรม

ภาษาไทย

[1] จันทรมาส สาณะเสน พันธุ์คำ , "สู่ก้าวใหม่แห่งการพัฒนาโปรแกรมบนโทรศัพท์เคลื่อนที่ด้วย J2ME" , สำนักพิมพ์ซายน์ซอฟต์ คอร์ปอเรชั่น

ภาษาอังกฤษ

[1] Vartan Piroumian , " Wireless J2ME Platform Programming" , A Prentice Hall Title

[2] John w. Muchow , "Core J2ME Technology & MIDP" , Prentice Hall PTR , Upper Saddle River , NJ 07458

[3] Paul Tremblett , "Instant Wireless Java with J2ME" , McGraw-Hill/Osborne

เว็บไซต์

[1] http://www.samspublishing.com

[2] http://www.webyu.com

[3] http://java.sun.com/aboutJava/communityprocess/

[4] http://java.sun.com/j2me/

[5] http://java.sun.com/product/cldc/

[6] http://java.sun.com/aboutJava/communityprocess/final/jsr030/index.html

[7] http://java.sun.com/aboutJava/communityprocess/final/jsr037/index.html

[8] http://java.sun.com/products/javaphone/

[9] http://www.sun.com/forte/ffj/

[10] http://sun.java.com/products/j2mewtoolkit/

[11] http://java.sun.com/product/midp/

[12] http://archives.java.com/archives/kvm-interest.html

[13] http://www.wirelessdevnet.com

[14] http://www.kvmworld.com

[15] www.phptr.com

ภาคผนวก ก
แสดงคลาสไลบรารีของ CLDC

Java.lang

class java.lang.Object

class java.lang.Boolean

class java.lang.Byte

class java.lang.Character

class java.lang..Class

class java.lang.Integer

class java.lang.Long

class java.lang.Math

class java.lang.Runtime

class java.lang.Short

class java.lang.String

class java.lang.StringBuffer

class java.lang.System

class java.lang.Thread (implements java.lang.Runnable)

class Class java.lang.Throwable

class java.lang.Error

class java.lang.VirtualMachineError

class java.lang.OutOfMemoryError

class java.lang.Exception

class java.lang.ClassNotFoundException

class java.lang.IllegalAccessException

class java.lang.InstantiationException

class java.lang.InteruptedException

class java.lang.RuntimeException

class java.lang.ArithmeticException

class java.lang.ArrayStoreException

class java.lang.ClassCastException

class java.lang.IllegalArgumentException

class java.lang.IllegalThreadStateException

class java.lang.NumberFormatException

class java.lang.IllegalMonitorStateException

class java.lang.IllegalStateException

class java.lang.IndexOutOfBoundsException

class java.lang.ArrayIndexOutOfBoundException

class java.lang.StringIndexOutBoundException

class java.lang.NegativeArraySizeException

class java.lang.NullPointerException

class java.lang.SecurityException

java.io

class java.lang.Object

class java.io.InputStream

class java.io.ByteArrayInputStream (implements java.io.DataInput)

class java.io.OutputStream

class java.io.ByteArrayOutputStream

class java.io.DataOutputStream (implements java.io.DataOutput

class java.io.PrintStream

class java.io.Reader

class java.io.InputStreamReader

class java.io.Throwable

class java.io.Exception

class java.io.IOException

class java.io.EOFException

class java.io.InterruptedIOException

class java.io.UnsupportedEncodingException

class java.io.UTFDataFoormatException

class java.io.Write

class java.io.OutputStreamException

Interface java.io.DataInput

Interface java.io.DataOutput

Java.util

class java.lang.Object

class java.util.Calendar

class java.util.Date

class java.util.Hashtable

class java.util.Random

class java.lang.Throwable

class java.lang.Exception

class java.lang.RuntimeException

class java.util.EmptyStackException

class java.util.NoSuchElementException

class java.util.Timer

class java.util.TimerTask(implements java.lang.Runnable)

class java.util.TimeZone

class java.util.Vector

class java.util.Stack

javax.microedition.io

class java.lang.Object

class javax.microedtion.io.Connector

class java.lang.Throwable

class java.io.IOException

class javax.microedtion.io.ConnectionNotFoundException

interface javax.microedtion.io.Connection

interface javax.microedtion.io.DatagramConnection

interface javax.microedtion.io.InputConnection

interfacejavax.microedtion.io.StreamConnection(alsextendsjavax.microedtion.io.OutputConne
ction

interface javax.microedtion.io.ContentConnection

interface javax.microedtion.io.OutputConnection

interface

javax.microedtion.io.StreamConnection(alsextendsjavax.microedtion.io.InputConnection)

interface javax.microedtion.io.ContentConnection

interface javax.microedtion.io.StreamConnectionNotifier

interface javax.io.DataInput

interface javax.microedtion.io.Datagram(alsextends java.io.DataOutput)

interface javax.io.DataOutput

interface javax.microedtion.io.Datagram(alsextends java.io.DataInput)

ภาคผนวก ข

คลาสไลบรารีของ MIDP

Javax.microedtion.lcdui

class java.lang.Object

class javax.microedtion.lcdui.AlertType

class javax.microedtion.lcdui.Command

class javax.microedtion.lcdui.Display

class javax.microedtion.lcdui.Displayable

class javax.microedtion.lcdui.Canvas

class javax.microedtion.lcdui.Screen

class javax.microedtion.lcdui.Alert

class javax.microedtion.lcdui.Form

class javax.microedtion.lcdui.List(implements class javax.microedtion.lcdui.Choice)

class javax.microedtion.lcdui.TextBox

class javax.microedtion.lcdui.Font

class javax.microedtion.lcdui.Graphics

class javax.microedtion.lcdui.Image

class javax.microedtion.lcdui.Item

class javax.microedtion.lcdui.ChoiceGroup(implements class javax.microedtion.lcdui.Choice)

class javax.microedtion.lcdui.DateField

class javax.microedtion.lcdui.Gauge

class javax.microedtion.lcdui.ImageItem

class javax.microedtion.lcdui.StringItem

class javax.microedtion.lcdui.TextField

class javax.microedtion.lcdui.Ticker

Interface javax.microedtion.lcdui.Choice

Interface javax. microedtion.lcdui.CommandListernet

Interface javax. microedtion.lcdui.ItemStateListener

Javax.microedtion.rms

class java.lang.Object

class javax.microedtion.rms.RecordStore

class java.lang.Throwable

class java.lang.Exception

class java.microedtion.rms.RecordStroreException

class java.microedtion.rms.InvalidRecordIDException

class java.microedtion.rms.RecordStoreFullException

class java.microedtion.rms.RecordStoreNotFoundException

class java.microedtion.rms.RecordStoreNotOpenException

Interface javax.microedtion.rms.RecordComparator

Interface javax.microedtion.rms.RecordEnumeration

Interface javax.microedtion.rms.RecordFilter

Interface javax.microedtion.rms.RecordListener

ภาคผนวก ค

แสดงเมธ็อดของคลาสต่างๆ

Method	Description
Boolean readBoolean()	อ่าน 1 ไบต์ และจะส่งค่ากลับเป็น true เมื่อไม่ใช่ 0 แต่จะส่งค่าเป็น false เมื่อเป็น 0
Byte readByte()	อ่านอินพุท 1 ไบต์
Char readChar()	อ่านหนึ่ง char
Void readFully(byte[] b)	อ่าน b ไบต์ กันจนกระทั่ง b ไบต์หาได้ หรือจบไฟล์
Void close()	ปิดอินพุทสตรีม
Int read()	อ่านไบต์ถัดไปและส่งค่ากลับเป็น int ในช่วง 0-255
int read(byte[] b,int off,int len)	อ่านความยาว len ไบต์ของอาเรย์ b ส่งค่ากลับเป็นจำนวนจริงของไบต์ที่อ่าน สามารถอ่านค่าที่น้อยกว่าหรือเท่ากับ len ได้ จะส่งค่ากลับเป็น -1 ถ้าจบไฟล์
String toString()	จะแปลงทีละไบต์ของ char และจะได้ค่าเป็น String ออกมา
Void writeChar(char v)	เขียนค่า char 2 ไบต์ ไบต์สูงก่อน
Void flush()	flush เป็นสตรีม
Static int parseInt(String s)	เปลี่ยน s เป็นเลขฐานสิบ
static int abs(int a)	ให้ค่า a ที่เป็น absolute
int compareTo(String s)	ให้ค่า 0 เมื่อเท่ากัน -1 เมื่อน้อยกว่า 1 เมื่อมาก
String concat(String s)	ให้ค่าที่ต่อของ String นี้และ s
String substring(int b)	ให้ค่า String ตั้งแต่ b ถึงสุดท้าย
String substring(int b,int e)	ให้ค่า String ตั้งแต่ b ถึง e
int nextInt()	ให้ค่า random ถัดไปที่เป็น int
InputStream openInputStream()	เปิดและส่งค่า input สตรีมสำหรับการติดต่อ
OutputStream openOutputStream()	เปิดและส่งค่า output สตรีมสำหรับการติดต่อ
Static Connection open(String name)	สร้างและเปิดการเชื่อมต่อ
Static InputStream openInputStream(String name)	สร้างและเปิดการเชื่อมต่อ input สตรีม
Static OutputStream openOutputStream(String name)	สร้างและเปิดการเชื่อมต่อ output สตรีม
Int getKeyCode(int gameAction)	ให้ค่า key code ที่ตรงกันกับแอ๊กชั่นของเกมส์

Protected void keyPressed(int keycode)	เรียกเมื่อ key ถูกกด
Protected void keyReleased(int keycode)	เรียกเมื่อ key มาถึง
Protected abstract void paint(Graphic g)	ทำ Canvas
Void repaint()	ขอการวาดซ้ำสำหรับสกีน
Static Display getDisplay()	รับการแสดงของ MIDlet
Void setCurrent(Displayable next)	ร้องขอ Displayable สร้างภาพขึ้น
Void addCommand(Command cmd)	เพิ่ม Command
Int append(String s)	ต่อสตริงให้กับ form
Static Image createImage(String name)	สร้างรูปจากแหล่งภาพ
Int addRecord(byte[] data,int off,off nb)	เขียน nb ไบต์ของข้อมูลเริ่มที่ off ถึงตำแหน่ง ถัดไปในที่เก็บและส่งค่า recordID กลับ
Void closeRecordStore()	ปิด RecordStore
Void deleteRecordStore()	ลบ record
Int getNumRecords()	ส่งค่าจำนวน record ในปัจจุบันมา