Mục lục

LỜI MỞ ĐẦU

Ngày nay kỹ thuật đồ họa ngày càng phát triển và ứng dụng trong nhiều lĩnh vực của đời sống .Mọi người có thể dễ dàng nhận ra điều đó trong các bộ phim, các bức ảnh kỹ thuật số, trò chơi giải trí và trong cả các lĩnh vực khoa học , xã hội như y học, thiên văn học, khí tượng học, kiến trúc, quy hoạch đô thị .

Với mong muốn đi sâu vào nghiên cứu, tìm hiểu kỹ thuật đồ họa ứng dụng trong thực tế ,chúng em đã chọn đề tài " Đồ họa trên J2ME và viết ứng dụng minh họa " để làm đề tài Bài tập lớn của nhóm . Bên cạnh đó không chỉ bởi tính hấp dẫn của J2ME vốn là một nền tảng kỹ thuật khá mới mẻ, mà còn bởi vì ứng dụng J2ME phần lớn được viết cho các thiết bị điện thoại di động – một thiết bị đã quá quen thuộc trong cuộc sống của mỗi người , đặc biệt là tầng lớp học sinh, sinh viên.

Chúng em xin cảm ơn thầy giáo Lê Tấn Hùng - Giảng viên bộ môn Công nghệ phần mềm, viện Công nghệ thông tin, trường Đại học Bách Khoa Hà Nội đã tận tình hướng dẫn chúng em trong quá trình học tập và nghiên cứu .

Chúng tớ xin cảm ơn các thành viên lớp Công nghệ phần mềm đã có những góp ý xây dựng và giúp đỡ chúng tớ hoàn thành đề tài Bài tập lớn .

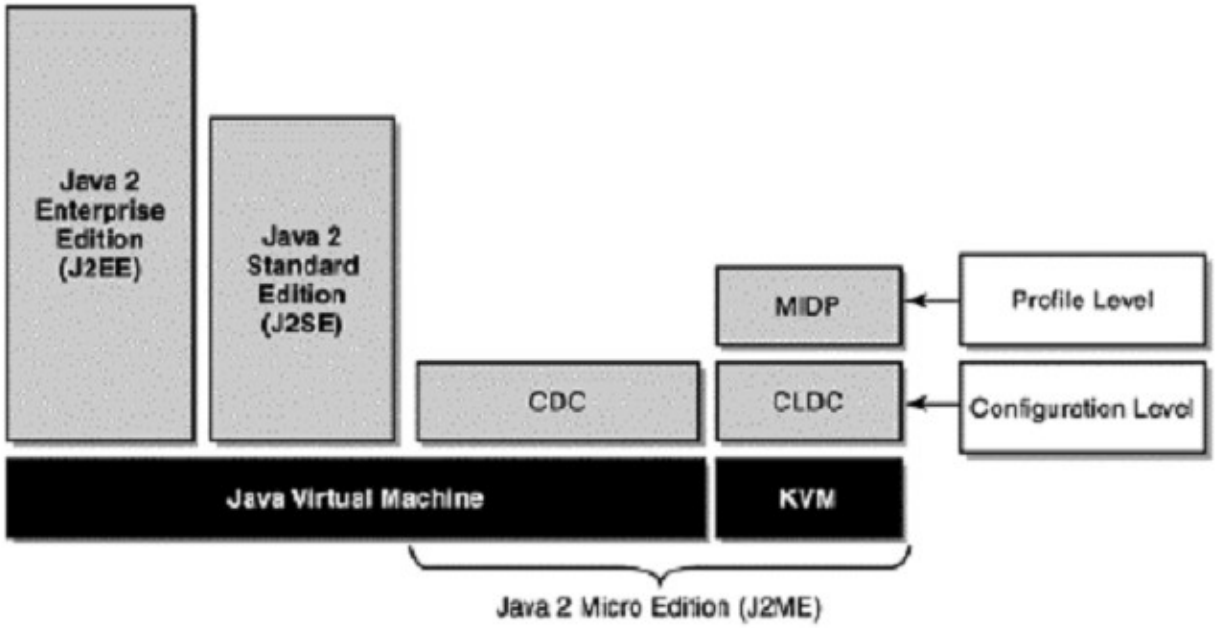
A .Các mức đồ họa trong J2ME

I. Tổng quan về J2ME

1. Giới thiệu J2ME :

J2ME được phát triển từ kiến trúc Java Card, Embeded Java và Personal Java của phiên bản Java 1.1. Đến sự ra đời của Java 2 thì Sun quyết định thay thế Personal Java và đươc gọi với tên mới là Java 2 Micro Edition, hay viết tắt là J2ME. Đúng với tên gọi, J2ME là nền tảng cho các thiết bị có tính chất nhỏ, gọn. Hiện tại phần lớn các hang điện thoại di động trên thế giới đều tích hợp máy ảo Java trong sản phẩm của mình, chính vì vậy cho tới ngày any J2ME vẫn là ngôn ngữ phổ biến nhất để lập trình ứng dụng trên di động nói chung và game trên di động nói riêng.

2 Kiến trúc của J2ME :



<u>Giới thiệu các thành phần trong nền tảng J2ME:</u>

<u>* Định nghĩa về Configuration (Cấu hình):</u> Là đặc tả định nghĩa một môi trường phần mềm cho một dòng các thiết bị được phân loại bởi tập hợp các đặc tính, ví dụ như:

• Kiểu và số lượng bộ nhớ
• Kiểu và tốc độ bộ vi xử lý
• Kiểu mạng kết nối

Do đây là đặc tả nên các nhà sản xuất thiết bị như Samsung, Nokia …bắt buộc phải thực thi đầy đủ các đặc tả do Sun qui định để các lập trình viên có thể dựa vào môi trường lập trình nhất quán và thông qua sự nhất quán này, các ứng dụng được tạo ra có thể mang tính độc lập thiết bị cao nhất có thể. Ví dụ như một lập trình viên viết chương trình game cho điện thoại Samsung thì có thể sửa đổi chương trình của mình một cách tối thiểu nhất để có thể chạy trên điện thọai Nokia..

Hiện nay Sun đã đưa ra 2 dạng Configuration:

• CLDC (Connected Limited Device Configuration-Cấu hình thiết bị kết nối giới hạn): được thiết kế để nhắm vào thị trường các thiết bị cấp thấp (low-end), các thiết bị này thông thường là máy điện thọai di động và PDA với khoảng 512 KB

bộ nhớ. Vì tài nguyên bộ nhớ hạn chế nên CLDC được gắn với Java không dây (Java Wireless), dạng như cho phép người sử dụng mua và tải về các ứng dụng Java, ví dụ như là Midlet.

- CDC- Connected Device Configuration (Cấu hình thiết bị kết nối): CDC được đưa ra nhắm đến các thiết bị có tính năng mạnh hơn dòng thiết bị thuộc CLDC nhưng vẫn yếu hơn các hệ thống máy để bàn sử dụng J2SE. Những thiết bị này có nhiều bộ nhớ hơn (thông thường là trên 2Mb) và có bộ xử lý mạnh hơn. Các sản phẩm này có thể kể đến như các máy PDA cấp cao, điện thoại web,…

Cả 2 dạng Cấu hình kể trên đều chứa máy ảo Java (Java Virtual Machine) và tập hợp các lớp (class) Java cơ bản để cung cấp một môi trường cho các ứng dụng J2ME. Tuy nhiên, bạn chú ý rằng đối với các thiết bị cấp thấp, do hạn chế về tài nguyên như bộ nhớ và bộ xử lý nên không thể yêu cầu máy ảo hổ trợ tất cả các tính năng như với máy ảo của J2SE.

<u>* Định nghĩa về Profile :</u>

Profile mở rộng Configuration bằng cách thêm vào các class để bổ trợ các tính năng cho từng thiết bị chuyên biệt. Cả 2 Configuration đều có những profile liên quan và từ những profile này có thể dùng các class lẫn nhau. Đến đây ta có thể nhận thấy do mỗi profile định nghĩa một tập hợp các class khác nhau, nên thường ta không thể chuyển một ứng dụng Java viết cho một profile này và chạy trên một máy hỗ trợ một profile khác. Cũng với lý do đó, bạn không thể lấy một ứng dụng viết trên J2SE hay J2EE và chạy trên các máy hỗ trợ J2ME.

Sau đây là các profile tiêu biểu:

- Mobile Information Device Profile (MIDP): profile này sẽ bổ sung các tính năng như hỗ trợ kết nối, các thành phần hỗ trợ giao diện người dùng … vào CLDC. Profile này được thiết kế chủ yếu để nhắm vào điện thọai di động với đặc tính là màn hình hiển thị hạn chế, dung lượng chứa có hạn. Do đó MIDP sẽ cung cấp một giao diện người dùng đơn giản và các tính năng mạng đơn giản dựa trên HTTP. Có thể nói MIDP là profile nổi tiếng nhất bởi vì nó là kiến thức cơ bản cho lập trình Java trên các máy di động (Wireless Java)
- PDA Profile: tương tự MIDP, nhưng với thị trường là các máy PDA với màn hình và bộ nhớ lớn hơn
- Foundation Profile: cho phép mở rộng các tính năng của CDC với phần lớn các thư viện của bộ Core Java2 1.3

Ngoài ra còn có Personal Basis Profile, Personal Profile, RMI Profile, Game Profile.

3 Căn bản về lập trình J2ME :

3.1 MIDlet :

MIDlet là một ứng dụng Java được thiết kết để chạy trên các thiết bị di động. Một MIDlet chứa các lớp Java dùng bởi CLDC và MIDP. Một bộ MIDlet gồm một hoặc nhiều MIDlet được đóng gói cùng nhau và nén trong file JAR. File JAR

sẵn sàng cho việc cài đặt vào điện thoại. Vấn đề của lập trình viên là tạo ra các MIDlet. Sau đây là cấu trúc lập trình của một MIDlet.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*
public class MidletExample extends MIDlet {
   public void MIDletExample() { }
   public void startApp() { }

   public void pauseApp() {}

   public void destroyApp(boolean unconditional) { }
}
```

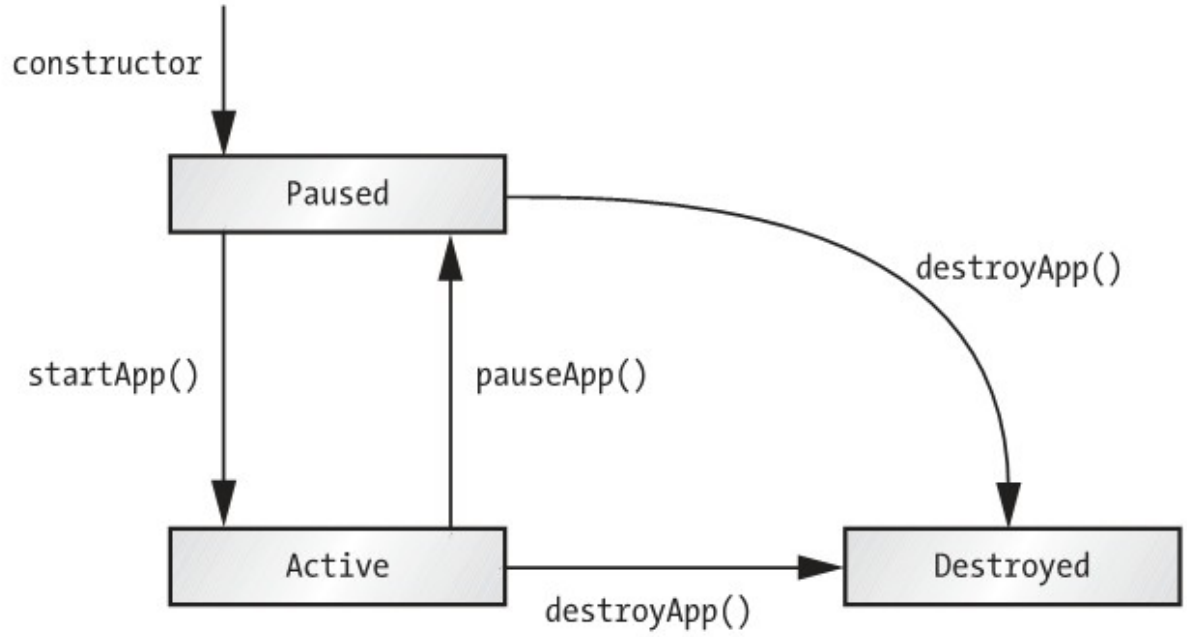
Một Midlet luôn luôn kế thừa lớp MIDlet có sẵn trong MIDP. Lớp MIDlet nằm trong gói: javax.microedition.midlet, các phương thức cơ bản mà một Midlet luôn phải có :

startApp( ) :Phương thức này được gọi bởi bộ quản lý ứng dụng khi khởi tạo MIDlet và mỗi khi MIDlet trở về từ trạng thái paused.

pauseApp( ): Phương thức này gọi bộ quản lý ứng dụng mỗi khi ứng dụng cần tạm dừng ứng dụng đang thực thi. Khi sử dụng pauseApp ta giải phóng một phần tài nguyên của MIDlet để dành bộ nhớ cho các ứng dụng khác.

destroyApp( ): Phương thức này được dùng khi thoát khỏi MIDlet. Trước đó phải giải phóng hoàn toàn bộ nhớ được lấy bởi MIDlet.

<u>* Vòng đời của một Midlet :</u>

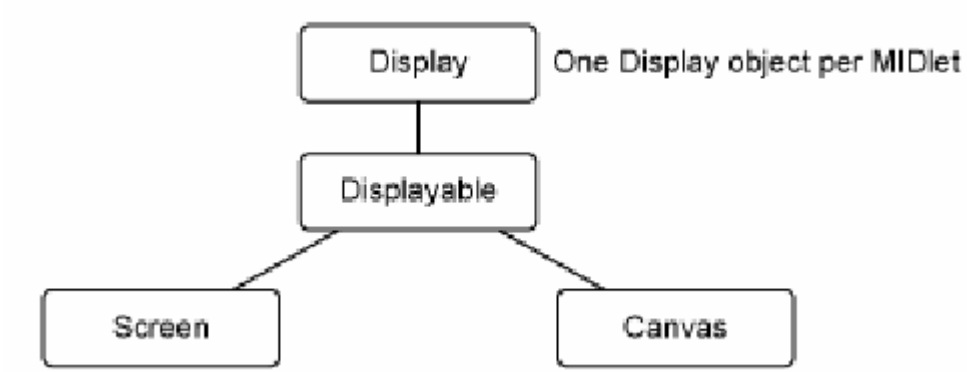


3.2 API của J2ME :

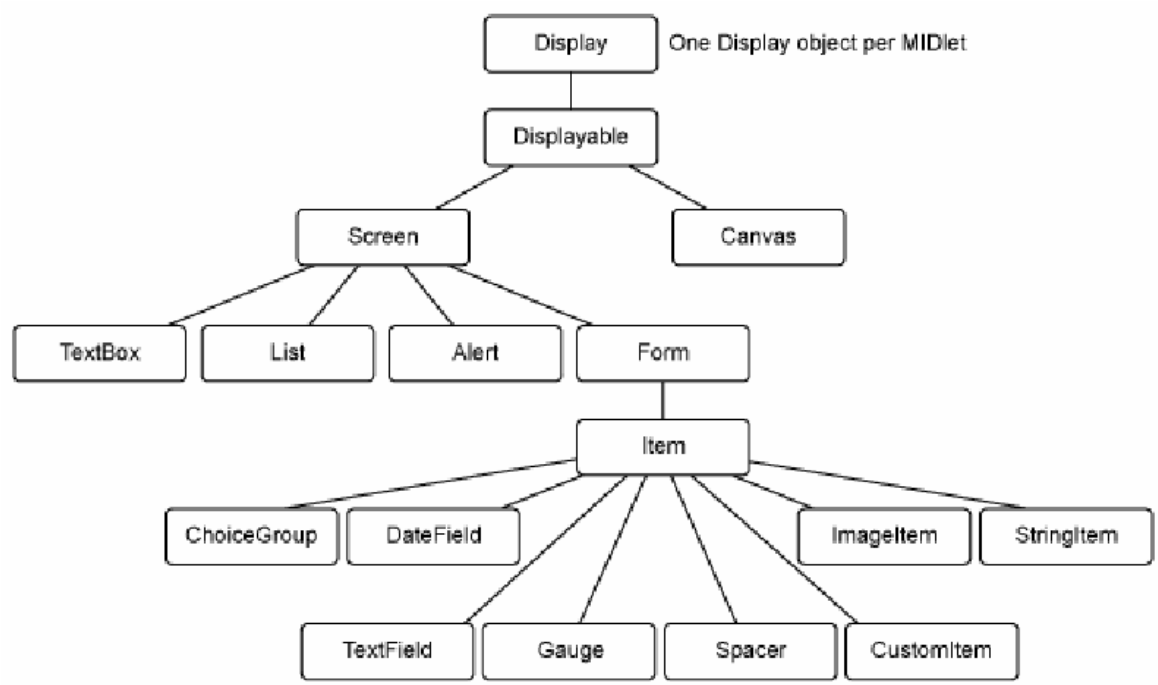
Một ứng dụng MIDlet chỉ có 1 đối tượng thể hiện Display. Đối tượng này dùng để lấy thông tin về đối tượng trình bày, ví dụ màu được hỗ trợ, và bao gồm các phương thức để yêu cầu các đối tượng được trình bày. Đối tượng Display cần thiết cho bộ quản lý việc trình bày trên thiết bị điều khiển thành phần nào sẽ được hiển thị lên trên thiết bị

Mặc dù chỉ có một đối tượng Display ứng với mỗi MIDlet, nhưng nhiều đối tượng trong một MIDlet có thể được hiển thị ra trên thiết bị như Forms, TextBoxes, ChoiceGroups, ..

Một đối tượng Displayable là một thành phần được hiển thị trên một thiết bị. MIDP chứa 2 lớp con của lớp Displayable là Screen và Canvas. Hình dưới đây mô tả mối quan hệ trên



Một đối tượng Screen không phải là một cái gì đó hiện ra trên thiết bị, mà lớp Screen này sẽ được thừa kế bởi các thành phần hiển thị ở mức cao, chính các thành phần này sẽ được hiển thị ra trên màn hình. Hình dưới đây sẽ mô tả mối quan hệ của lớp Screen và các thành phần thể hiện ở mức cao.



Lớp Canvas được sử dụng trong các chương trình game hay các ứng dụng cần vẽ những đối tượng riêng biệt, đặc điểm của nó là ít tính khả chuyển, ta thao tác với lớp này để vẽ các đối tượng lên màn hình. Lớp Canvas cũng có thể bắt các sự kiện bằng nhấn phím bấm. Vì là mức thấp cho phép người lập trình thỏa mái thao tác với màn hình. Cả hai lớp trên đều nằm trong gói javax.microedition.lcdui.*

Trong lập trình game trên di động các thành phần giao diện ở mức cao được sử dụng để tạo các Menu, các lựa chọn cài đặt cho Game.

II.Các thành phần đồ họa ở mức cao của ứng dụng MIDP

1. Form và Items :

Trong phần này sẽ giới thiệu các thành phần được hiển thị ra trên một Form. Một Form chỉ đơn giản là một khung chứa các thành phần, mà mỗi thành phần được thừa kế từ lớp Item. Chúng ta sẽ xem qua các thành phần hiển thị trên thiết bị trên:

- DateField
- Gauge
- StringItem
- TextField
- ChoiceGroup
- Spacer
- CustomItem
- Image and ImageItem

a) DateField

Thành phần DateField cung cấp một phương tiện trực quan để thao tác đối tượng Date được định nghĩa trong java.util.Date. Khi tạo một đối tượng DateField, bạn cần chỉ rõ là người dùng chỉ có thể chỉnh sửa ngày, chỉnh sửa giờ hay đồng thời cả hai. Các phương thức khởi tạo của lớp DateField gồm:

DateField(String label, int mode)

DateField(String label, int mode, TimeZone timeZone)

Các mode tương ứng của lớp DateField gồm:

DateField.DATE_TIME: cho phép thay đổi ngày giờ

DateField.TIME: chỉ cho phép thay đổi giờ

DateField.DATE: chỉ cho phép thay đổi ngày

Ví dụ:

private DateField dfAlarm;

// Tạo một đổi tượng DateField với nhãn, cho phép thay đổi cả ngày và giờ

dfAlarm = new DateField("Set Alarm Time", DateField.DATE_TIME);

dfAlarm.setDate(new Date());

b) Gauge

Một thành phần Gauge là một kiểu giao diện thường được dùng để mô tả mức độ hoàn thành một công việc. Có 2 loại Gauge là loại tương tác và loại không tương tác. Loại đầu cho phép người dùng có thể thay đổi Gauge, loại 2 thì đòi hỏi người phát triển phải cập nhật Gauge

Dưới đây là hàm khởi tạo của lớp Gauge:

Gauge(String label, boolean interactive, int maxValue, int initialValue)

Ví dụ:

private Gauge gaVolume; // Điều chỉnh âm lượng

gaVolume = new Gauge("Sound Level", true, 100, 4);

c) StringItem

Một thành phần StringItem được dùng để hiển thị một nhãn hay chuỗi văn bản. Người dùng không thể thay đổi nhãn hay chuỗi văn bản khi chương trình đang chạy.

StringItem không nhận ra sự kiện

Phương thức khởi tạo của lớp StringItem

StringItem(String label, String text)

d) TextField

Một thành phần TextField thì tương tự như bất kỳ các đối tượng nhập văn bản tiêu biểu nào. Bạn có thể chỉ định một nhãn, số ký tự tối đa được phép nhập, và loại dữ liệu được phép nhập. Ngoài ra TextField còn cho phép bạn nhập vào mật khẩu với các ký tự nhập vào sẽ được che bởi các ký tự mặt nạ

Phương thức khởi tạo của lớp TextField

TextField(String label, String text, int maxSize, int constraints)

Thành phần thứ 3 *constraints* là thành phần mà chúng ta quan tâm, vì nó là phương tiện để xác định loại dữ liệu nào được phép nhập vào TextField MIDP định nghĩa các tham số ràng buộc sau cho thành phần TextField:

- ANY: cho phép nhập bất kỳ ký tự nào
- EMAILADDR: chỉ cho phép nhâp vào các địa chỉ email hợp lệ
- NUMERIC: chỉ cho phép nhập số
- PHONENUMBER: Chỉ cho phép nhập số điện thoại
- URL: Chỉ cho phép nhập các ký tự hợp lệ bên trong URL
- PASSWORD: che tất cả các ký tự nhập vào

e) ChoiceGroup

Thành phần ChoiceGroup cho phép người dùng chọn từ một danh sách đầu vào đã được định nghĩa trước. ChoiceGroup có 2 loại:

- multi-selection(cho phép chọn nhiều mục): nhóm này có liên quan đến các checkbox
- exclusive-selection(chỉ được chọn một mục): nhóm này liên quan đến nhóm các radio button

f) Image and ImageItem

Hai lớp được dùng để hiển thị hình ảnh là: Image và ImageItem. Image được dùng để tạo ra một đối tượng hình ảnh và giữ thông tin như là chiều cao và chiều rộng, và dù ảnh có biến đổi hay không. Lớp ImageItem mô tả một tấm ảnh sẽ được hiển thị như thế nào, ví dụ tấm ảnh sẽ được đặt ở trung tâm, hay đặt về phía bên trái, hay bên trên của màn hình MIDP đưa ra 2 loại hình ảnh là loại ảnh không biến đổi và ảnh biến đổi. Một tấm ảnh không biến đổi thì không thể bị thay đổi kể từ lúc nó được tạo ra. Đặc trưng của loại ảnh này là được đọc từ một tập tin. Một tấm ảnh biến đổi về cơ bản là một vùng nhớ. Điều này tùy thuộc vào việc bạn tạo nội dung của tấm ảnh bằng cách ghi nó lên vùng nhớ. Chúng ta sẽ làm việc với những tấm ảnh không biến đổi trong bảng sau.

Các phương thức dựng cho lớp Image và ImageItem

- *Image createImage(String name)*
- *Image createImage(Image source)*
- *Image createImage(byte[] imageDate, int imageOffset, int imageLength)*
- *Image createImage(int width, int height)*
- *Image createImage(Image image, int x, int y, int width, int height, int transform)*
- *Image createImage(InputStream stream)*
- *Image createRGBImage(int[] rgb, int width, int height, boolean processAlpha)*
- *ImageItem(String label, Image img, int layout, String altText)*

Đoạn mã dưới đây mô tả làm thế nào tạo một tấm ảnh từ một tập tin, và gắn nó với một đối tượng ImageItem và thêm một bức ảnh vào một Form

```
Form fmMain = new Form("Images");
...
// Create an image
Image img = Image.createImage("/house.png");
// Append to a form
fmMain.append(new ImageItem(null, img, ImageItem.LAYOUT_CENTER, null));
```

Chú ý: PNG là loại ảnh duy nhất được hỗ trợ bởi bất kỳ thiết bị MIDP nào

2. List, Textbox, Alert, và Ticker :

a) List

Một List chứa một dãy các lựa chọn được thể hiện một trong ba dạng. Chúng ta đã thấy loại cho phép nhiều lựa chọn và loại chỉ được phép chọn một khi làm việc với ChoiceGroup. Dạng thứ 3 là là dạng không tường minh. Các List không tường minh được dùng để thể hiện một thực đơn các chọn lựa

b) TextBox

TextBox được dùng để cho phép nhập nhiều dòng. Thành phần TextBox và TextField có những ràng buộc giống nhau trong việc chỉ định loại nội dung được phép nhâp vào. Ví dụ ANY, EMAIL, URI… Dưới đây là phương thức dựng của một TextBox:

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

c) Alert và AlertType

Một Alert đơn giản là một hộp thoại rất nhỏ. Có 2 loại Alert:

- Modal: là loại hộp thoại thông báo được trình bày cho đến khi người dùng ấn nút đồng ý
- Non-modal: là loại hộp thoại thông báo chỉ được trình bày trong một số giây nhất định

Các phương thức khởi tạo của Alert:

Alert(String title)

Alert(String title, String alertText, Image alertImage, AlertType alertType)

Thành phần AlertType sử dụng âm thanh để thông báo cho người dùng biết có một sự kiện xảy ra. Ví dụ bạn có thể sử dụng AlertType để mở một đoạn âm thanh nào đó báo hiệu cho người dùng biết khi có lỗi xảy ra

Thành phần AlertType bao gồm 5 loại âm thanh định sẵn là: thông báo, xác nhận, báo lỗi, thông báo và cảnh báo

Ta thấy các phương thức dựng của Alert cho biết là Alert có thể bao gồm 1 tham chiếu đến một đối tượng AlertType.

d) Ticker

Thành phần Ticker được dùng để thể hiện một đoạn chuỗi chạy theo chiều ngang. Tham số duy nhất của thành phần Ticker là đoạn văn bản được trình bày. Tốc độ và chiều cuốn được xác định bởi việc cài đặt trên thiết bị nào.
Phương thức khởi tạo của Ticker
Ticker(String str)
Từ cây phân cấp các thành phần thể hiện trên thiết bị, ta thấy là thành phần Ticker không là lớp con của lớp Screen mà Ticker là một biến của lớp Screen. Điều này có nghĩa là một Ticker có thể được gắn vào bất cứ lớp con của lớp Screen bao gồm cả Alert

3. Xử lý sự kiện :

a. Đối tượng Command :

Khi một đối tượng xảy ra trên thiết bị di động, một đối tượng Command giữ thông tin về sự kiện đó. Thông tin này bao gồm loại hành động thực thi, nhãn của mệnh lệnh và độ ưu tiên của chính nó. Trong J2ME, các hành động nói chung được thể hiện dưới dạng các nút trên thiết bị.

Nếu có quá nhiều hành động được hiển thị trên thiết bị, thiết bị sẽ tạo ra một thực đơn để chứa các hành động .

Chỉ có các thành phần MIDP sau đây mới có thể chứa các đối tượng Command là: Form, TextBox, List và Canvas.
Các bước cơ bản để xử lý các sự kiện với một đối tượng Command:
• Tạo một đối tượng Command
• Đặt đối tượng Command lên trên một đối tượng Form, TextBox, List, hay Canvas
• Tạo một bộ lắng nghe

Khi có một sự kiện xảy ra, bộ lắng nghe sẽ phát sinh một lời gọi đến phương thức commandAction(). Trong thân phương thức này bạn có thể xác định đối tượng nào phát sinh ra sự kiện và tạo ra các xử lý tương ứng
Dưới đây là đoạn mã minh họa việc tạo ra sự kiện Command và xử lý sự kiện:

```
private Form fmMain;            // Form
private Command cmExit;         // Command to exit the MIDlet
 ...
fmMain = new Form("Core J2ME");   // Create Form and give it a title
// Create Command object, with label, type and priority
cmExit = new Command("Exit", Command.EXIT, 1);
fmMain.addCommand(cmExit);        // Add Command to Form
fmMain.setCommandListener(this);  // Listen for Form events
...
public void commandAction(Command c, Displayable s)
{
  if (c == cmExit)
```

```
  {
    destroyApp(true);
    notifyDestroyed();
  }
}
```

b. Đối tượng Item

Ngoài việc xử lý sự kiện bằng đối tượng Command ta có thể xử lý sự kiện bằng đối tượng Item. Nhiều đối tượng trong MIDP đã được định nghĩa trước cho việc xử lý các sự kiện cụ thể nào đó. Ví dụ đối tượng DateField cho phép người sử dụng chọn lựa ngày và giờ trên màn hình, đối tượng TextField cho phép người dùng nhập vào một chuỗi các ký tự, số và các ký tự đặc biệt Tương tự như việc xử lý sự kiện bằng đối tượng Command, thì khi có một thay đổi xảy ra đối với bất kỳ thành phần Item nào thì phương thức *itemStateChanged()* sẽ được gọi. Và chúng ta sẽ thực hiện các xử lý bên trong phương thức này

Dưới đây là đoạn mã minh họa việc tạo ra sự kiện Command và xử lý sự kiện:

```
private Form fmMain;                // Form
private DateField dfToday;          // DateField item
...
fmMain = new Form("Core J2ME");                  // Create Form object
dfToday = new DateField("Today:", DateField.DATE);  // Create DateField
...
fmMain.append(dfToday);             // Add DateField to Form
fmMain.setItemStateListener(this);  // Listen for Form events

...

public void itemStateChanged(Item item)
{
  // If the datefield initiated this event
  if (item == dfToday)
    ...
}
```

c. Ví dụ

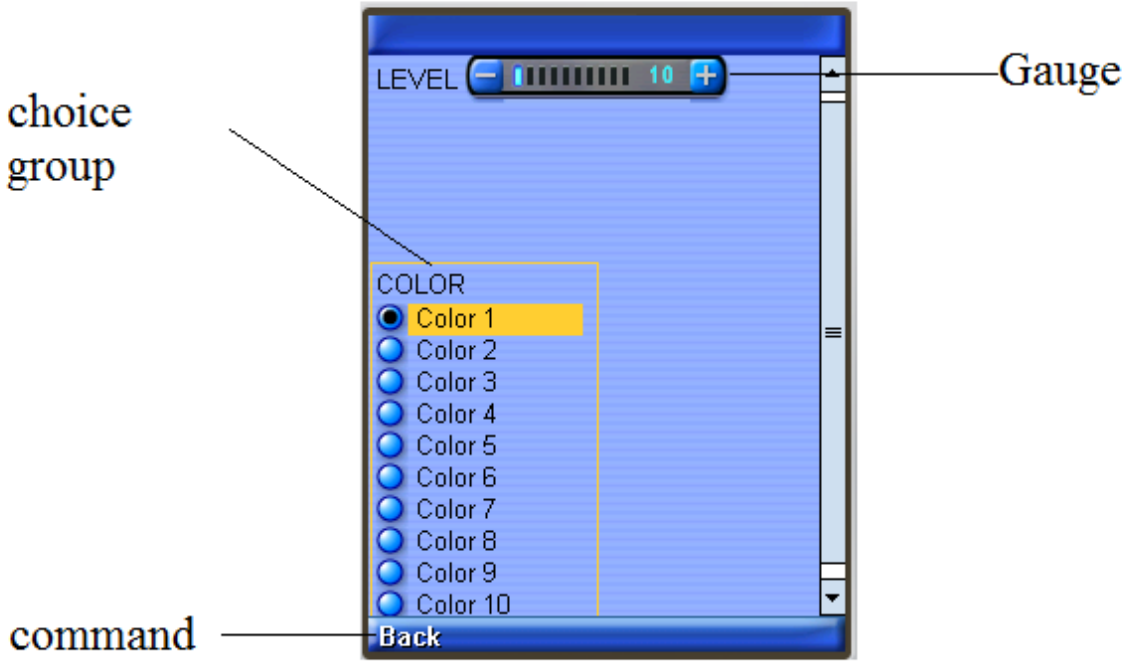
Dưới đây là đoạn mã minh họa việc tạo ra và xử lý các sự kiện Command, Item

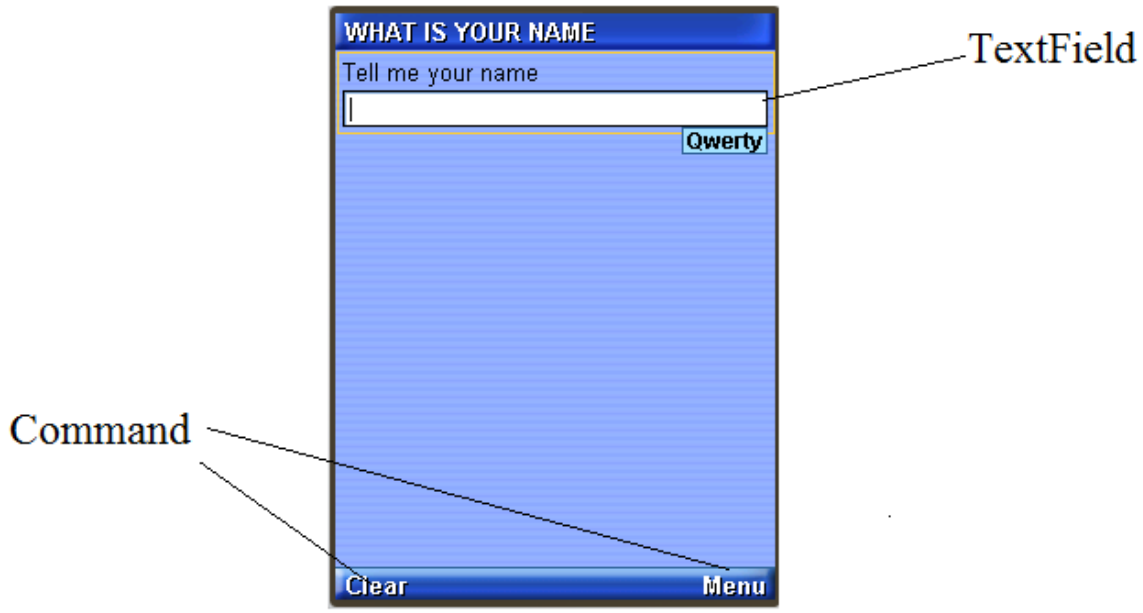
```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
public class EventProcessing extends MIDlet implements
ItemStateListener, CommandListener
{
    private Display display;    // Reference to Display object for this MIDlet
    private Form fmMain;        // Main Form
    private Command cmExit;     // Command to exit the MIDlet
    private TextField tfText;   // TextField component (item)
    public EventProcessing()
    {
      display = Display.getDisplay(this);
      // Create the date and populate with current date
      tfText = new TextField("First Name:", "", 10, TextField.ANY);

      cmExit = new Command("Exit", Command.EXIT, 1);

      // Create the Form, add Command and DateField
      // listen for events from Command and DateField
      fmMain = new Form("Event Processing Example");
      fmMain.addCommand(cmExit);
      fmMain.append(tfText);
      fmMain.setCommandListener(this);     // Capture Command events (cmExit)
      fmMain.setItemStateListener(this);   // Capture Item events (dfDate)
    }

    // Called by application manager to start the MIDlet.
    public void startApp()
    {
      display.setCurrent(fmMain);
    }

    public void pauseApp()
    { }

    public void destroyApp(boolean unconditional)
    { }

    public void commandAction(Command c, Displayable s)
    {
      System.out.println("Inside commandAction()");
```

```
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }

  public void itemStateChanged(Item item)
  {
    System.out.println("Inside itemStateChanged()");
  }
}
```

Một số hình ảnh minh họa





III. Các thành phần đồ họa ở mức thấp của ứng dụng MIDP

Khi lập trình cho máy tính, thường chúng ta chỉ sử dụng các thành phần giao diện có sẵn để tạo ra giao diện người dùng. Tuy nhiên, khi lập trình cho điện thoại di động sử dụng MIDP, thành phần giao diện có sẵn khá nghèo nàn. MIDP chỉ cung cấp các thành phần giao diện thông dụng nhất. Đối với máy tính, nếu một thành phần giao diện chúng ta cần không sẵn có trong bộ công cụ phát triển (SDK), chúng ta có thể dễ dàng sử dụng thư viện của một nhà cung cấp thứ ba. Điều này rất khó cho các ứng dụng trên ĐTDĐ luôn yêu cầu dung lượng nhỏ. Chúng ta không thể tích hợp cả một thư viện vào ứng dụng để sử dụng một hoặc một vài thành phần giao diện. Để giải quyết vấn đề này, chúng ta cần phải sử dụng thư viện lập trình giao diện người dùng cấp thấp (Low-Level User Interface Libraries) để tạo các thành phần giao diện mong muốn.

Canvas và **Graphics** là 2 lớp trái tim của các hàm API cấp thấp. Bạn sẽ làm tất cả các công việc bằng tay. Canvas là một khung vẽ cho phép người phát triển có khả năng vẽ lên thiết bị trình bày cũng như là việc xử lý sự kiện. Còn lớp Graphics cung cấp các công cụ thật sự để vẽ như **drawRoundRect()** và **drawString()**

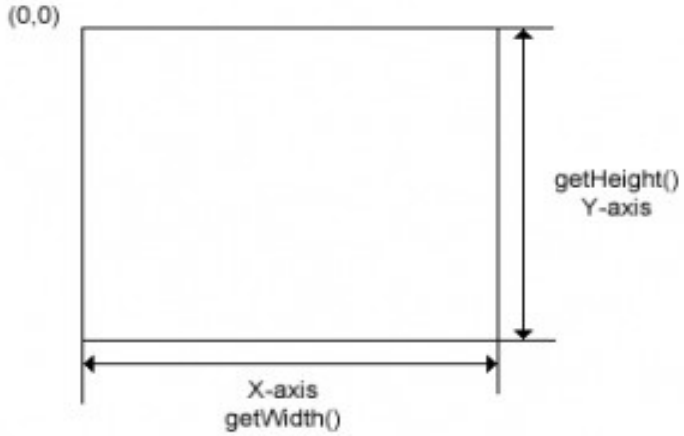
1. Lớp Canvas

Canvas là lớp cơ sở trong J2ME đượ sử dụng để xử lý các sự kiện cấp thấp và các vấn đề liên quan đến đồ họa để hiển thị lên màn hình.

Do Canvas là một lớp ảo(abstract), nên bản thân lớp Canvas sẽ không thể tạo ra đối tượng trực tiếp mà phải thông qua các lớp con của nó. Các lớp con của Canvas, sau khi extends Canvas, phải cài đặt phương thức ảo của nó, trong Canvas chỉ có phương thức paint() là phương thức ảo, nên ta có thể cài đặt con của lớp Canvas như sau:

```
public class MainCanvas extends Canvas{
   public void paint(Graphics g){
   g.setColor(255,255,255);
   g.drawString("SMI Java",0,0,Graphics.TOP|Graphics.LEFT);
}
}
```

1. Hệ tọa độ:



Mục tiêu đầu tiên của chúng ta là làm quen với hệ thống trục tọa độ để làm việc
với thiết bị thể hiện. Hệ thống tọa độ cho lớp Canvas có tâm tọa độ là điểm trái trên của thiết bị trình bày. Giá trị x tăng dần về phía phải, giá trị y tăng dần khi đi xuống phía dưới. Khi vẽ độ dày bút vẽ là một điểm ảnh.
Các phương thức sau đây sẽ giúp xác định chiều rộng và chiều cao của canvas:
+ int getWidth(): xác định chiều rộng của canvas

+ int getHeight (): xác định chiều cao của canvas
Chiều rộng và chiều cao của Canvas cũng đại diện cho toàn bộ diện tích khung vẽ có thể trên thiết bị trình bày. Nói cách khác, chúng không thể chỉ định kích thước cho canvas, mà phần mềm trên một thiết bị MIDP sẽ trả về diện tích lớn nhất có thể có đối với một thiết bị cho trước.

2. Vẽ trên đối tượng Canvas:

a. Thiết lập chế độ vẽ Canvas lên toàn bộ màn hình:
Có 2 chế độ để Canvas được hiển thị:

- Normal: trong chế độ này, chiều cao vùng hiển thị canvas bằng tổng chiều cao màn hình trừ đi vùng tiêu đề và vùng vẽ các command.
- Full: trong chế độ này, chiều cao vùng hiển thị canvas bằng tổng chiều cao màn hình.

Các đối tượng Canvas thường được tạo ra theo chế độ mặc định. Để thiết lập chế độ full màn hình, bạn chỉ cần gọi phương thức setFullScreenMode(boolean mode).

Việc gọi phương thức setFullScreenMode (boolean mode) thực chất là gọi phương thức sizeChanged(int w, int h), bình thường thì phương thức này không làm gì. Ứng dụng có thể lợi dụng phương thức này để thay đổi chiều cao vào chiều rộng vùng vẽ đối các thiết bị có chức nay xoay màn hình.

b. Phương thức paint(Graphics g)

- Phương thức paint(Graphics g) được sử dụng để vẽ các đối tượng graphics lên màn hình. Để gọi phương thức paint(Graphics g), đơn giãn bạn chỉ cần tạo ra một instance của lớp Canvas và sau đó gọi phương thức paint(Graphics g).
- Có thể gọi phương thức paint(Graphics g) một cách bất đồng bộ thông qua gọi phương thức repaint().
- Việc gọi phương thức paint() thông qua gọi phương thức repaint() có thể không được thực hiện ngay lập tức. Nó có thể bị trì hoãn cho đến khi luồng(flow) điều khiển trả về từ phương thức xử lý sự kiện hiện tại. Sự chậm trễ ở đây không phải là vấn đề lớn, nhưng trong khi xử lý các chuyển động, cách an toàn nhất để kích hoạt việc vẽ lại là gọi phương thức Display.callSerially() hay yêu cầu vẽ lại từ một thread riêng biệt hoặc TimerTask. Ngoài ra, ứng dụng có thể ép buộc việc vẽ lại ngay lập tức bằng cách gọi phương thức serviceRepaints().Ứng dụng có thể sử dụng phương thức phương thức Display.callSerially() hay serviceRepaints() để đồng bộ với phương thức paint().

Vd: xóa màn hình

```
protected void paint(Graphics g)
{
// Set background color to white
g.setColor(255, 255, 255);
// Fill the entire canvas
g.fillRect(0, 0, getWidth(), getHeight());
}
```

2.Lớp Graphics:

- Đối tượng Graphics được sử dụng bởi Canvas, nó có thể được biểu diễn trực tiếp lên màn hình hay thông qua một bộ đệm off-screen trước khi vẽ lên màn hình.
- Đối tượng Graphics hỗ trợ vẽ các chuỗi string, image, đường thẳng, đường tròn…
- Cung cấp một kiểu màu sắc 24bit, trong đó 8bit cho mỗi thành phần màu red, green và blue. Không phải tất cả các thiết bị đều hỗ trợ đầy đủ 24bit màu, vì vậy chúng sẽ được ánh xạ các màu sắc yêu cầu của ứng dụng vào bảng màu hiện tại được hỗ trợ trên thiết bị.
- Graphics có một phương thức rất quan trọng translate(int x, int y) để dịch chuyển tọa độ gốc, phương thức này thường dùng trong các kỹ thuật cuộn màn hình.
- Ngoài phương thức translate(int x, int y), lớp Graphics còn có một phương thức nữa quan trọng khác là setClip(int x, int y, int width, int height), các vùng nằm ngoài vùng clip được thiết lập sẽ không có hiệu lực, chính điều này làm tăng tốc quá trình vẽ.

1. Thiết lập màu:

Màu sắc trong đối tượng Graphics được tạo từ 3 màu cơ bản là đỏ, xanh lục và xanh dương(RGB) được biểu diễn bởi 24 bit màu, mỗi màu chiếm 8 bit. Nếu thiết bị không hỗ trợ hiển thị màu hay chỉ hỗ trợ với số màu ít hơn thì thiết bị sẽ tự chuyển đổi để hiển thị ra màu gần nhất so với yêu cầu. Grayscale dùng để hiển thị các sắc độ màu khác nhau trên màn hình trắng đen.

Khi gán màu cho đối tượng Graphics, có thể dùng một trong 2 cách:

- Kết hợp 3 màu (đỏ, xanh dương và xanh lục) trong một số nguyên kiểu int hay biểu diễn từng thành phần trong từng số int. Khi kết hợp 3 màu với nhau, mỗi

màu được biểu diễn bởi 8 bit và xếp theo thứ tự từ cao xuống thấp là đỏ, xanh lục và xanh dương.

R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
RED								GREEN								BLUE							

Do đó, để kết hợp 3 giá trị màu này vào một số nguyên kiểu int, ta phải dịch bit để có kết quả chính xác. Giá trị màu đỏ sẽ dịch trái 16bits, giá trị màu xanh lục dịch trái 8 bits và giá trị màu xanh dương chiếm 8 bit thấp nhất.

VD:

int red=0; *0 0 0 0 0 0 0 0*

int reen=127; *0 1 1 1 1 1 1 1*

int blue=255; *1 1 1 1 1 1 1 1*

int color=(red<<16) | (green<<8) | blue;

g.setColor(color); //g là đối tượng kiểu Graphics

//Gán giá trị cho từng màu cơ bản riêng biệt:

int red=0;

int green=127;

int blue=255;

g.setcolor(red, green, blue);

Tương tự, khi lấy giá trị màu, ta cũng có thể lấy ra dưới dạng kết hợp hay từng màu riêng biệt. Nếu lấy giá trị màu dưới dạng kết hợp, để có được giá trị màu cho từng màu cơ bản cần phải bỏ đi các bit không thích hợp.

Vd:

int color, red, green, blue;
color= g.getColor(); //g là đối tượng kiểu Graphics
red=(color & 0xFF0000)>>16;
green=(color & 0xFF00)>>8;
blue=color & 0xFF;

Có thể kiểm tra thiết bị di động có màn hình màu không bằng phương thức: *javax.microedition.lcdui.Display.isColor().*

Ngoài ra có thể lấy số màu thiết bị có hỗ trợ bằng phương thức *javax.microedition.lcdui.Display.numColors().*

, nếu thiết bị chỉ có màn hình trắng đen thì giá trị trả về là giá trị grayscale mặc định.

2. Nét vẽ

Kiểu nét vẽ (stroke style): Khi vẽ đường thẳng, đường tròn, hay hình chữ nhật, ta có thể lựa chọn kiểu nét vẽ là nét liền (Graphics.SOLID) hay cách khoảng (Graphics.DOTTED). Nét liền là giá trị mặc định của các kiểu vẽ.

Các phương thức vẽ:

- *void* ***drawLine*** *(int x1, int y1, int x2, int y2)*

Vẽ đường thẳng từ điểm (*x1, y1*) đến điểm (*x2, y2*)

- *void* ***drawArc*** *(int x, int y, int width, int height, int startAngle, int acrAngle)*

Vẽ đường cong góc *arcAngle* bắt đầu từ góc *startAngle* nội tiếp hình chữ nhật xác định bởi góc trái trên (*x,y*) và chiều dài hai cạnh *width, height*

- *void* ***fillArc*** *(int x, int y, int width, int height, int startAngle, int acrAngle)*

Vẽ đường cong góc *arcAngle* bắt đầu từ góc *startAngle* nội tiếp hình chữ nhật xác định bởi góc trái trên (*x,y*) và chiều dài hai cạnh *width, height* và tô màu bên trong đường cong vừa vẽ.

- *void* ***drawRect*** *(int x, int y, int width, int height)*

Vẽ hình chữ nhật với góc trái trên (*x,y*) và chiều dài hai cạnh *width, height*

- *void* ***drawRoundRect*** *(int x, int y, int width, int height, int arcW, int arcH)*

Vẽ hình chữ nhật với các góc tròn. *arcW* và *arcH* là kích thước của hình chữ nhật ngoại tiếp góc cong

- *void* ***fillRect*** *(int x, int y, int width, int height)*

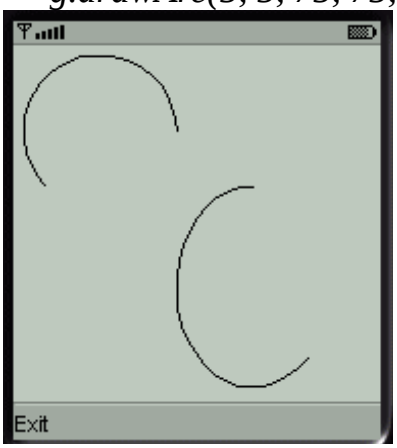
Tô hình chữ nhật

- *void* ***fillRoundRect*** *(int x, int y, int width, int height, int arcW, int arcH)*

tô hình chữ nhật góc tròn

VD:

g.drawArc(5, 5, 75, 75, 0, 225);



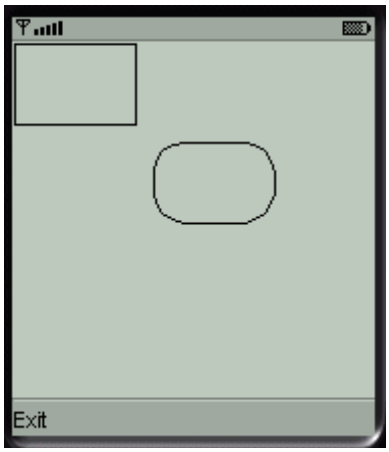
g.drawArc(80, 70, 75, 100, 90, 225);

Hình chữ nhật gồm 4 loại:

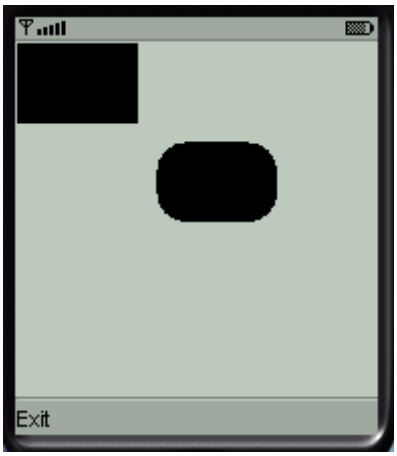
- Hình chữ nhật thường, không tô
- Hình chữ nhật góc cong, không tô
- Hình chữ nhật thường, tô đặc
- Hình chữ nhật góc tròn, tô đặc

Hình chữ nhật góc tròn là hình chữ nhật trong đó các góc được vẽ theo dạng đường cong chứ không phải là góc vuông như hình chữ nhật thông thường. Trong đó các góc cong là ¼ đường cong nội tiếp hình chữ nhật kích thước rộng dài tương ứng là: arcW và arcH.

VD: *g.drawRect(1, 1, 60, 40);*
g.drawRoundRect(70, 50, 60, 40, 30, 40);



g.fillRect(1, 1, 60, 40);
g.fillRoundRect(70, 50, 60, 40, 30, 40);



Các phương thức hỗ trợ vẽ text:

- *void* ***drawChar*** *(char character, int x, int y, int anchor)*
vẽ ký tự *character* lên màn hình

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

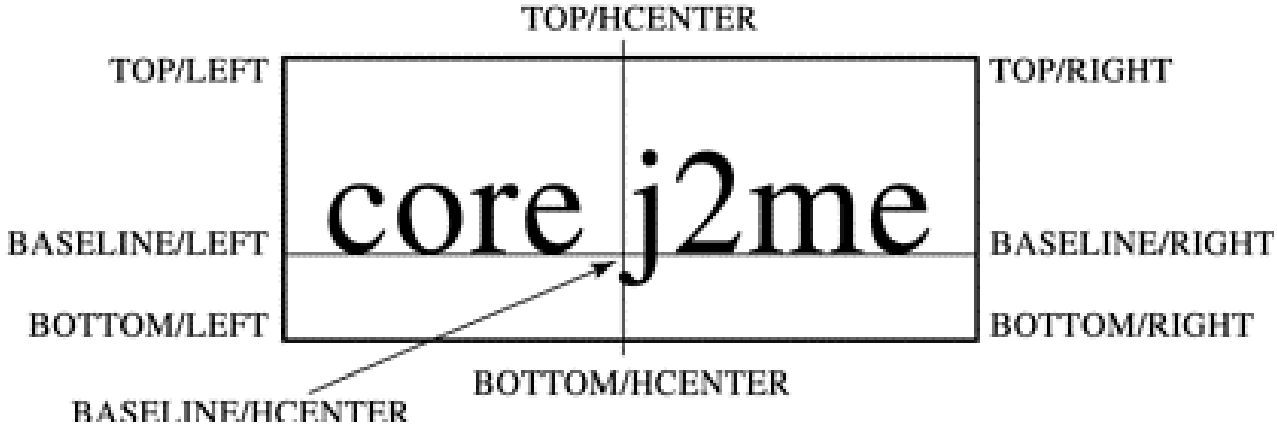
vẽ chuỗi ký tự *data* với chiều dài *length* từ vị trí *offset* lên màn hình

- *void* ***drawString*** *(String str, int x, int y, int anchor)*
vẽ chuỗi *str* lên màn hình

- *void* ***drawSubstring*** *(String str, int offset, int len, int x, int y, int anchor)*
vẽ chuỗi con có chiều dài *len* , từ vị trí *offset* của chuỗi *str* lên màn hình

- *Font* ***getFont()*** : Lấy đối tượng Font hiện tại
- *void* ***setFont ()*** : Gán Font cho đối tượng Graphics

Khi vẽ text lên màn hình, bất kể đó là một ký tự hay một chuỗi ký tự, ta phải cung cấp đủ ba tham số: *x*, *y* và *anchor*. Trong đó, (x, y) là toạ độ điểm gốc, và anchor là vị trí tương đối của điểm gốc so với đoạn text sẽ được hiển thị. Các vị trí tương đối của anchor đã được định nghĩa sẵn trong lớp Graphics.



3. Font chữ:

Phần sau đây cũng quan trọng không kém là cách sử dụng font chữ được hỗ trợ bởi giao diện cấp thấp của ứng dụng MIDP. Sau đây là một số các phương thức dựng của lớp Font

- *Font getFont(int face, int style, int size)*
- *Font getFont(int fontSpecifier)*
- *Font getDefaultFont()*

Một số thuộc tính của lớp Font

FACE_SYSTEM
FACE_MONOSPACE
FACE_PROPORTIONAL
STYLE_PLAIN
STYLE_BOLD
STYLE_ITALIC

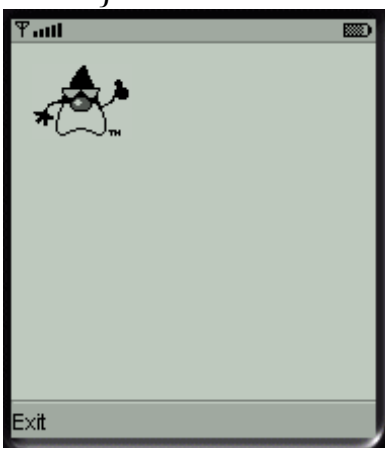
STYLE_UNDERLINED
SIZE_SMALL
SIZE_MEDIUM
SIZE_LARGE
Các tham số kiểu dáng có thể được kết hợp thông qua toán tử hay. Ví dụ:
Font font = Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD | Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
Sau khi bạn có một tham chiếu đến một đối tượng Font, bạn có thể truy vấn nó để
xác định thông tin của các thuộc tich của nó.
int getFace()
int getStyle()
int getSize()
boolean isPlain()
boolean isBold()
boolean isItalic()
boolean isUnderlined()
Kích thước của các font chữ được xác định bởi chiều cao của font chữ, bề dài tính bằng điểm ảnh của một chuỗi ký tự trong một font xác định. Một số các phương thức sau hỗ trợ khi tương tác với một đối tương font
int getHeight()
int getBaselinePosition()
int charWidth(char ch)
int charsWidth(char[] ch, int offset, int length)
int stringWidth(String str)
int substringWidth(String str, int offset, int length)

4. Vẽ ảnh

Lớp Graphics hỗ trợ một phương thức duy nhất để vẽ hình ảnh lên màn hình, đó là:
*void **drawImage** (Image img, int x, int y, int anchor):* Vẽ đối tượng Image *img* lên màn hình
Đây chỉ là bước cuối cùng để vẽ một hình ảnh lên màn hình, các bước trước đó dùng để tạo đối tượng Image cũng như load hình ảnh lên đối tượng đó. Đối tượng Image gồm hai loại: có thể thay đổi hình ảnh (Mutable Image) và hình ảnh cố định (Immutable Image).
Các bước để vẽ đối tượng Image lên màn hình:
- Immutable Image:

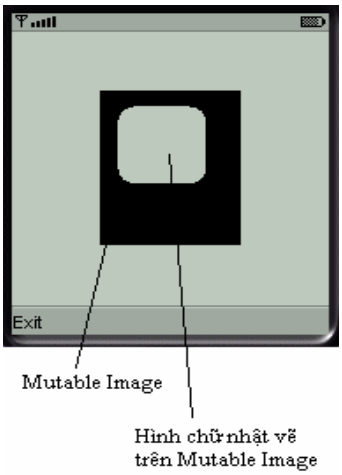
+ Tạo đối tượng hình ảnh (thường là load trực tiếp từ file):
Image img= Image.createImage ("\imageTest.png");
+ Hiển thị đối tượng lên màn hình:

protected void paint (Graphics g) {
…
g.drawImage(img, 10, 10,
Graphics.LEFT | Graphics.TOP);
}



- Mutable Image:

Khởi tạo đối tượng, yêu cầu chương trình cấp một vùng nhớ với kích thước cho trước của hình ảnh:
Image img = Image.createImage(80,100);
Vẽ hình ảnh lên đối tượng :
//lấy đối tượng Graphics tương ứng của đối tượng Image:
Graphics g = img.getGraphics();
//dùng đối tượng Graphics vừa lấy vẽ các hình ảnh lên đối tượng Image:
g.fillRoundRect (0, 0, 50, 50, 20, 20);
…
Hiển thị hình vừa vẽ lên màn hình:
protected void paint (Graphics g) {
…
g.drawImage(img, 10, 10,
Graphics.VCENTER | Graphics.HCENTER);
}



Khi gọi phương thức vẽ đối tượng Image lên màn hình, ta cũng phải truyền các tham số x, y, anchor như khi vẽ text. Tuy nhiên, các loại anchor của hình ảnh tương đối khác với text, đó là không sử dụng anchor BASELINE(đường chẩn của ký tự / chuỗi) mà thay vào đó là VCENTER để chỉ điểm giữa theo chiều dọc của hình ảnh.

B.Ứng dụng Game trong J2ME

Xuất phát từ niềm đam mê và kiến thức sẵn có về lập trình game, nhóm chúng em đã lựa chọn việc xây dựng một số game Mobile để làm ứng dụng minh họa cho đề tài nghiên cứu này

I.Sơ lược về lập trình Game trên J2ME

<u>* J2ME Game API :</u>

Thực tế ta hoàn toàn có thể làm game J2ME với kiến thức về lập trình di động J2ME, và đặc biệt là thành phần giao diện mức thấp Canvas của nó. Lớp Canvas cho phép ta vẽ lên màn hình điện thoại, nó hỗ trợ vẽ các đối tượng như đường thẳng, đường tròn, tô màu một vùng màn hình, hay có thể copy một vùng màn hình vào vùng đệm và đặt nó vào chỗ khác …. Với những công cụ như thế ta có thể làm được một chương trình game từ đơn giản đến phức tạp. Tuy nhiên thời gian làm game sẽ rất tốn kém và khối lượng công việc liên quan đến vẽ giao diện là rất lớn.Qua tìm hiểu em thấy J2ME có một gói hỗ trợ tốt cho lập trình game gọi là chung là Game API. Các lớp của nó nằm trong *javax.microedition.lcdui.game*

Một số lớp quan trọng của nó như: GameCanvas, Layer, Sprites, TiledLayer, và LayerManager.

- GameCanvas: kế thừa từ lớp Canvas, có thể coi đay là màn hình điện thoại
- Layer: ít được sử dụng trực tiếp nhưng được kết bởi lớp Sprite và TiledLayer
- Sprites: lớp để tạo các nhân vật và các đối tượng chuyển động

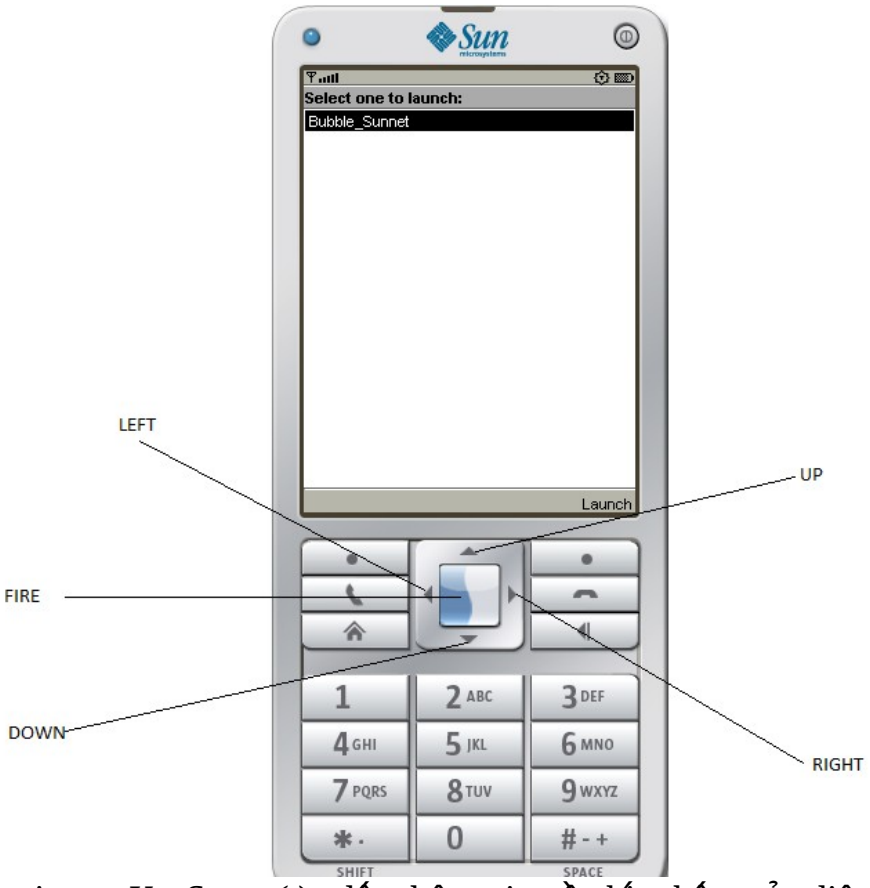
• TiledLayer: lớp để tạo các khung cảnh khác nhau của game và tạo bản đồ
• LayerManager: lớp quản lý các Sprites và các TiledLayer

1. Lớp GameCanvas :

GameCanvas kết thừa các thuộc tính và phương thức của lớp Canvas và có thêm một số phương thức quan trọng giúp cho việc giảm thời gian tính toán nhờ đó tăng tốc độ refresh của màn hình. Nó có thêm hai mở rộng so với Canvas là Polling Input và Frame Buffer.

1.1 Polling Input :

Trong game di động người ta thường sử dụng các phím chức năng để thao tác. Một game hoàn chỉnh có thể chỉ thao tác với 5 phím chức năng là UP, DOWN, LEFT, RIGHT và phím FIRE (phím bắn). Ta có thể dùng các phím số tương ứng thay cho các phím chức năng như bảng trên. Điện thoại bố trí các phím chức năng đặt gần nhau để dễ thao tác.



Phương thức *int getKeyStates( )* lấy thông tin về phím bấm của điện thoại để biết người chơi nhấn nút gì.

Trong lớp Canvas có 1 phương thức tương tự để xác định người dung sử dụng phím gì đó là *getKeyCodes( )* nhưng phương thức này có một điểm hạn chế đó là nó sẽ kiểm tra tất cả các phím xem phím nào nhấn, trong khi đó

getKeyStates( ) của GameCanvas chỉ kiểm tra với các phím chức năng do đó thời gian nhận biết được phím bấm của lớp GameCanvas sẽ nhanh hơn của lớp Canvas.

Ví dụ nhận biết phím bấm RIGHT

```
int keyStates = getKeyStates();
if ((keyStates & GameCanvas.RIGHT_PRESSED) != 0) {
    System.out.println("right");
}
```

1.2 Frame Buffer :

Kỹ thuật này sử dụng phương thức *protected Graphics getGraphics()* để lấy nội dung của Canvas và cho vào bộ đệm. Ý tưởng của Frame Buffer là mọi thao tác vẽ trên Canvas sẽ được gián tiếp thao tác với vùng đệm sau đó dùng phương thức *public void flushGraphics()* để đẩy nội dung vùng đệm ra Canvas. Với cải tiến này thì Frame Buffer có tốc độ vẽ lên Canvas nhanh hơn là vẽ trực tiếp lên Canvas vì thao tác với vùng đệm nhanh là hơn thao tác với một đối tượng Canvas cụ thể, ta chỉ mất thêm hai thao tác là lấy nội dung Canvas vào đối tượng của lớp Graphics với phương thức *getGraphics()* và đẩy nội dung từ đối tượng Graphics vào Canvas với phương thức *flushGraphics()*.

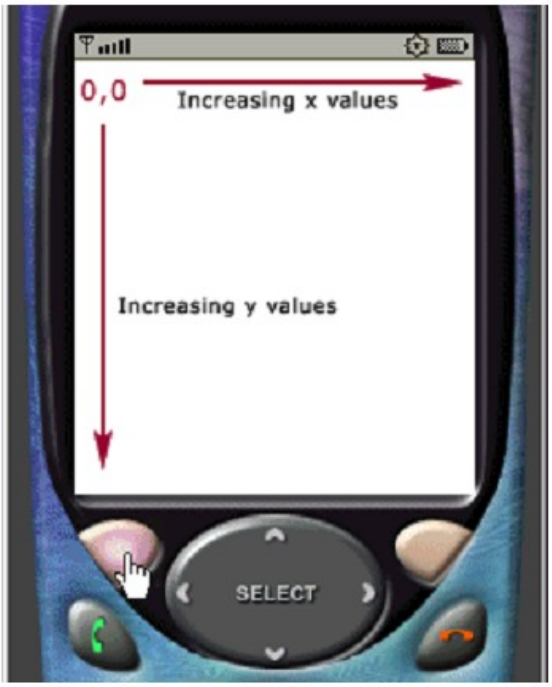
2. Layer :

Layer được hiểu là một khối hình ảnh trong Game. Layer là một lớp trừu tượng được chứa trong gói *javax.microedition.lcdui.game.Layer* .Tất cả các hình ảnh có thể hiện được trên màn hình đều kế thừa lớp này (2 lớp rất quan trọng kế thừa từ Layer là Sprite và TiledLayer). Các Layer là những ảnh mà ta có thể vẽ lên màn hình, ẩn đi, di chuyển hay là xắp xếp chúng theo độ xâu (tức là khi nhiều ảnh chồng chéo lên nhau thì ảnh có độ xâu lớn hơn sẽ bị ảnh có độ xâu nhỏ hơn che khuất)

Các phương thức của lớp Layer là :

public final int getX() : Trả về giá trị là tọa độ x của một Layer
public final int getY() : Trả về giá trị là tọa độ x của một Layer
public final int getWidth() : Trả về giá trị là chiều rộng của một Layer
public final int getHeight() : Trả về giá trị là chiều dài của một Layer
public void setPosition(int x, int y) : Đặt vị trí của Layer tại vị trí (x, y)
public void move(int dx, int dy) : Dịch chuyển Layer đến tọa độ (x+dx, y+dy)
public void setVisible (boolean visible) : Thiết lập hiện/ẩn Layer

Màn hình đồ họa của game trong J2ME :



3.Sprite

Một Sprite là một thuật ngữ chung trong game. Nó tham chiếu đến một phần tử trực quan được tạo ra bởi các Image, thường chuyển động và di chuyển xung quanh các phần tử khác một cách độc lập trong game. Lớp Sprite trong MIDP 2.0 đại diện cho khái niệm này. Nó cho phép tạo ra các Sprite dựa trên các hình ảnh với nhiều frame. Nó có thể thay đổi frame, điều khiển chuyển động và kiểm tra va chạm với các phần tử khác.

Các Sprite được nhân nhiều lên là Graphics động, các Graphics này được tạo nên từ cùng 1 phương thức Sprite nhưng nhìn từ các góc khác nhau.

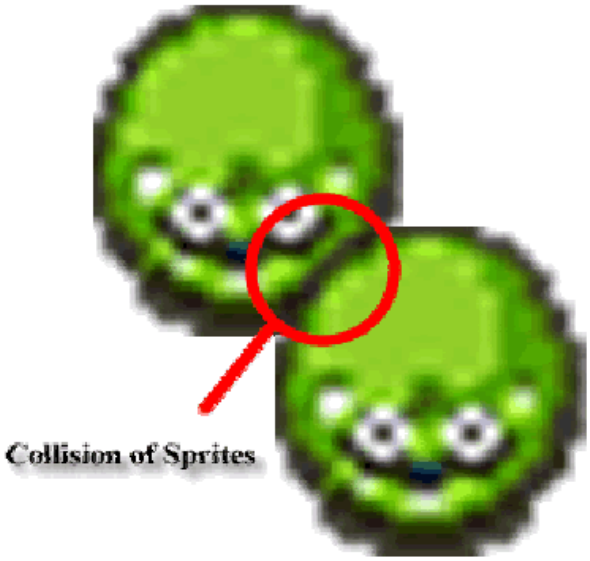
Các hàm khởi tạo:

Phương thức	**Mô tả**
Sprite(Image img)	Tạo một khung Sprite không động
Sprite(Sprite sprite)	Tạo ra một Sprite từ một Sprite

Sprite(Image img,int frameWidth,int frameHeight)	Tạo một Sprite từ 2 frame trở lên, frameWidth và frameHeight là chiều rộng và chiều cao của Sprite

Các Graphics bao gồm các Sprite mà độ rộng và chiều cao là hằng số.

a. Sprite Collision



collidesWith(Image image, int x, int y, boolean pixelLevel);

collidesWith(Sprite sprite, boolean pixelLevel);

collidesWidth(TiledLayer tiledLayer, Boolean pixelLevel);

Kiểm tra đụng độ thông qua vị trí góc trái trên của Sprite

4.LayerManager

Lớp này quản lý tất cả các đối tượng khác, thay vì trực tiếp xác định vị trí để vẽ các đối tượng lên màn hình bằng phương thức *paint()*, ta tạo một đối tượng LayerManager và cho vào nó tất cả các đối tượng bằng phương thức:
public void append(Layer lm);
Đối tượng append có thể là đối tượng Layer hoặc những đối tượng của lớp kế thừa từ Layer như Sprite và TiledLayer.Ở đây ta vẫn vẽ từng đối tượng lên màn hình nhưng khác so với vẽ trực tiếp lên màn hình ở chỗ các quá trình vẽ được thâu tóm lại chỉ trong đối tượng LayerManager mà không vẽ rời rạc từng thành phần một. Việc vẽ tất cả các đối tượng đồng loạt như vậy sẽ tốt cho tính thống nhất xuất hiện của các đối tượng khác nhau trên màn hình. Để có thể xác định vị trí vẽ ra cho các Sprite, TiledLayer ta dùng phương thức *public void setPosition(int x, int y)* thừa kế từ lớp Layer.
Lớp LayerManager cung cấp các phương thức quản lý Layer như:

- *insert (Layer l, int index)*
- *remove (Layer l)*
- *getLayerAt (int index)*

Phương thức thứ nhất chèn một Layer vào lớp ở vị trí có chiều xâu index. Ở đây một layer chèn vào có tính đến chiều sâu mà nó được đặt (layer có chiều sâu lớn hơn sẽ bị che bởi những layer có chiều xâu nhỏ hơn). Do đó việc đặt một layer lên màn hình khi sử dụng LayerManager sẽ không phụ thuộc vào thứ tự viết code, mà phụ thuộc vào biến *index* truyền vào.

5.TiledLayer

Một TiledLayer là một lưới các các ô chia ra từ một ảnh.

Các phương thức:

Phương thức	**Mô tả**
TiledLayer(int columns, int rows, Image image, int tileWidth, int tileHeight)	Tạo 1 TiledLayer có số hàng, cột và ảnh cần chia, độ rộng và cao của tile
setCell(int col, int row, int tileIndex)	đặt tile vào bức ảnh ở vị trí col,row và lấy ảnh ,có tileIndex (ở trên là từ 1,2, …6)
getCell(int col, int row)	trả về index của cell, nếu cell là empty trả về 0
getCellHeight()	trả về chiều cao của một cell (pixel)
getCellWidth();	Trả về chiều rộng của một cell
getColumns()	trả về số cột của TileLayer
getRows()	Trả về số dòng của TileLayer

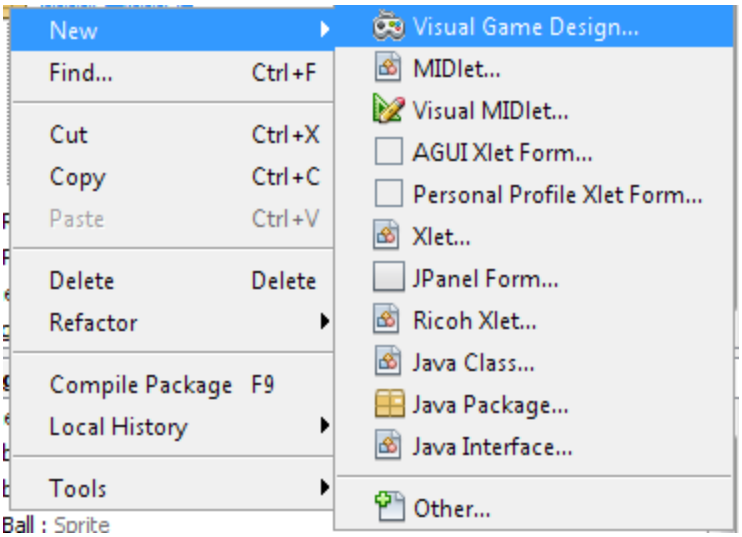
Gọi trực tiếp hàm paint() hay dùng LayerManager layerManager.paint(g,0,0)

II. Xây dựng game bằng GameBuilder trong IDE NetBeans

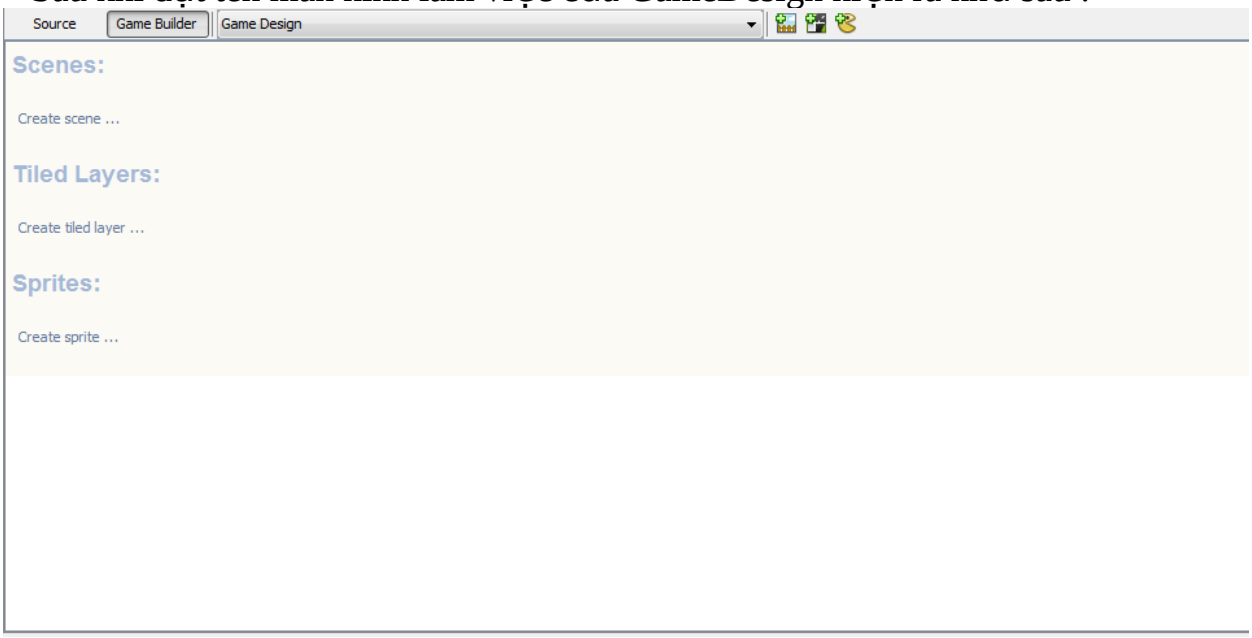
1. Giới thiệu về GameBuilder trong NetBeans :

Như trên đã biết để tạo các Sprite các TiledLayer và các bản đồ trong game chúng ta phải dùng các phương thức khởi tạo khá phức tạp. Những công đoạn đó rất dễ gây nhầm lẫn và không thuận tiện cho việc lập trình game vốn dĩ đã bao gồm nhiều thuật toán hết sức phức tạp.

Trong IDE NetBeans có cung cấp một công cụ giúp cho lập trình viên có thể tạo và quản lý các đối tượng đồ họa trong game một cách dễ dàng, đó chính là công cụ GameBuilder. Để tạo một lớp GameDesign ta thực hiện như sau :



Sau khi đặt tên màn hình làm việc của GameDesign hiện ra như sau :

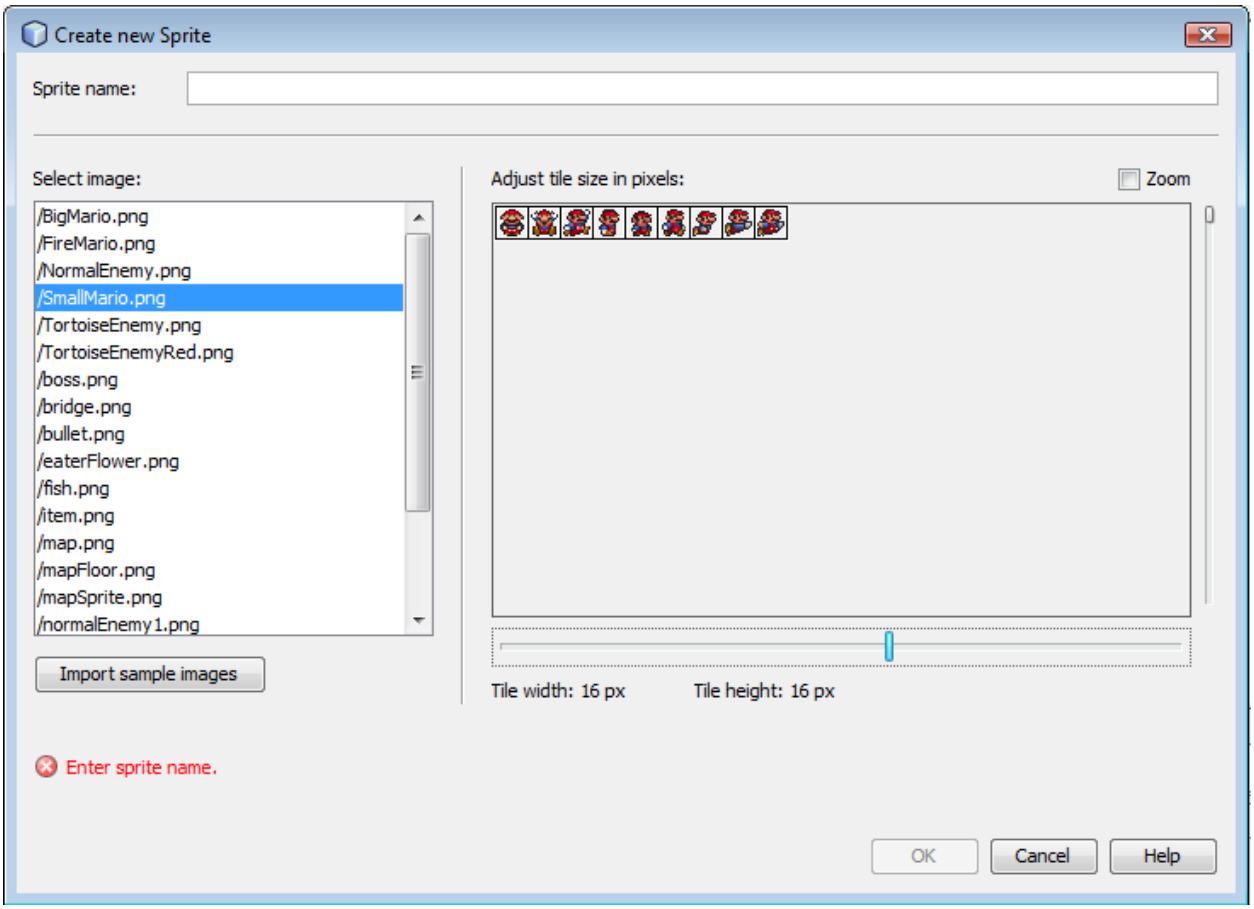


Phần Sprites là nơi chúng ta tạo các sprite của game. Phần Tiled Layers để chúng ta tạo các Tiled Layers. Phần Scenes là nơi chúng ta Add các ảnh vào để tạo cảnh hay bản đồ cho game.

2. Tạo Sprite bằng GameBuilder :

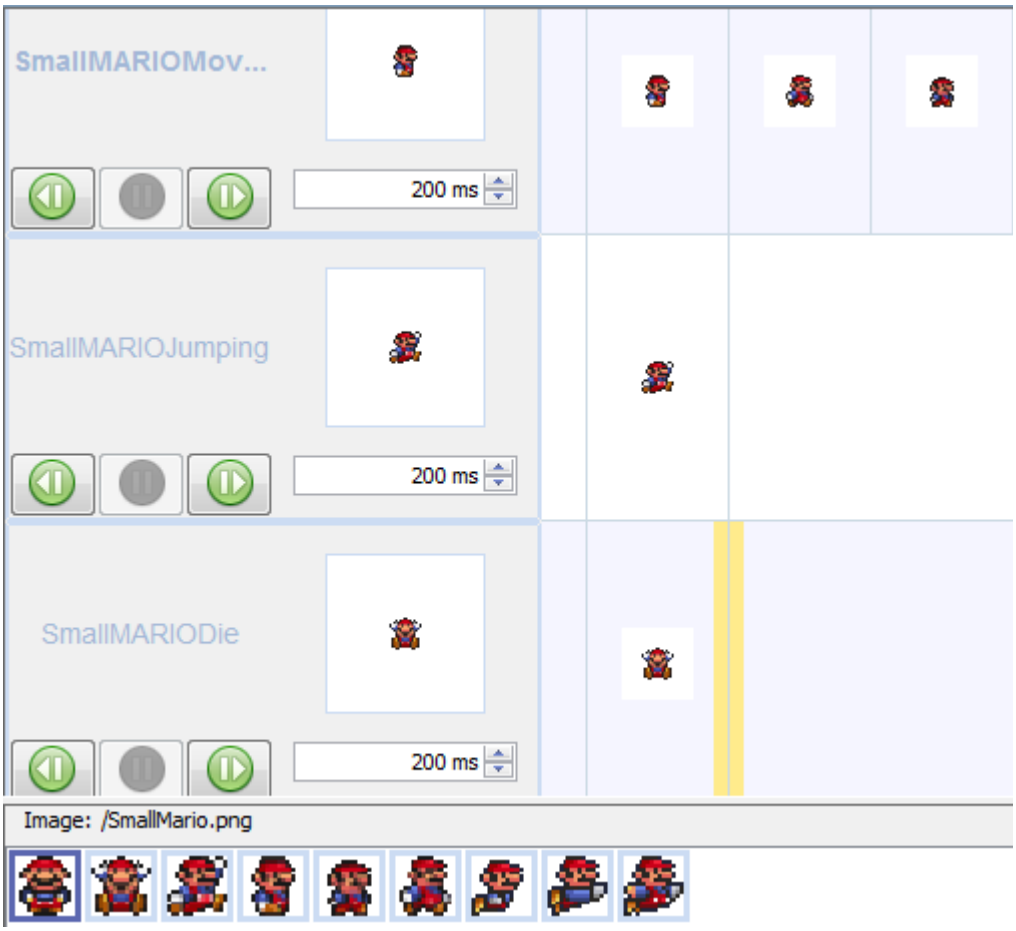
Trước khi tạo các đối tượng đồ họa bằng GameBuilder cần phải chú ý là trong thư mục src của project phải có sẵn file ảnh cần thiết.

Để tạo Sprite trong khung Sprites ta ấn vào “Create sprite” hoặc trên thanh menu ta ấn chọn biểu tượng tạo Sprite . Màn hình khởi tạo Sprite hiện ra :



Tại ô Sprite name ta đặt tên cho Sprite. Trong ô Select image ta lựa chọn ảnh cần dùng. Ảnh đó sẽ được xuất hiện tại ô bên phải, tại đó ta chỉnh sửa kích thước của Frame theo kích thước của mình, rồi ấn OK. Việc thực hiện này cũng giống như việc ta chia một bức ảnh ban đầu thành nhiều các frame như trong phần lý thuyết ở trên đã nói đến.

Tiếp đó khung màn hình tạo sequences cho sprite hiện ra.

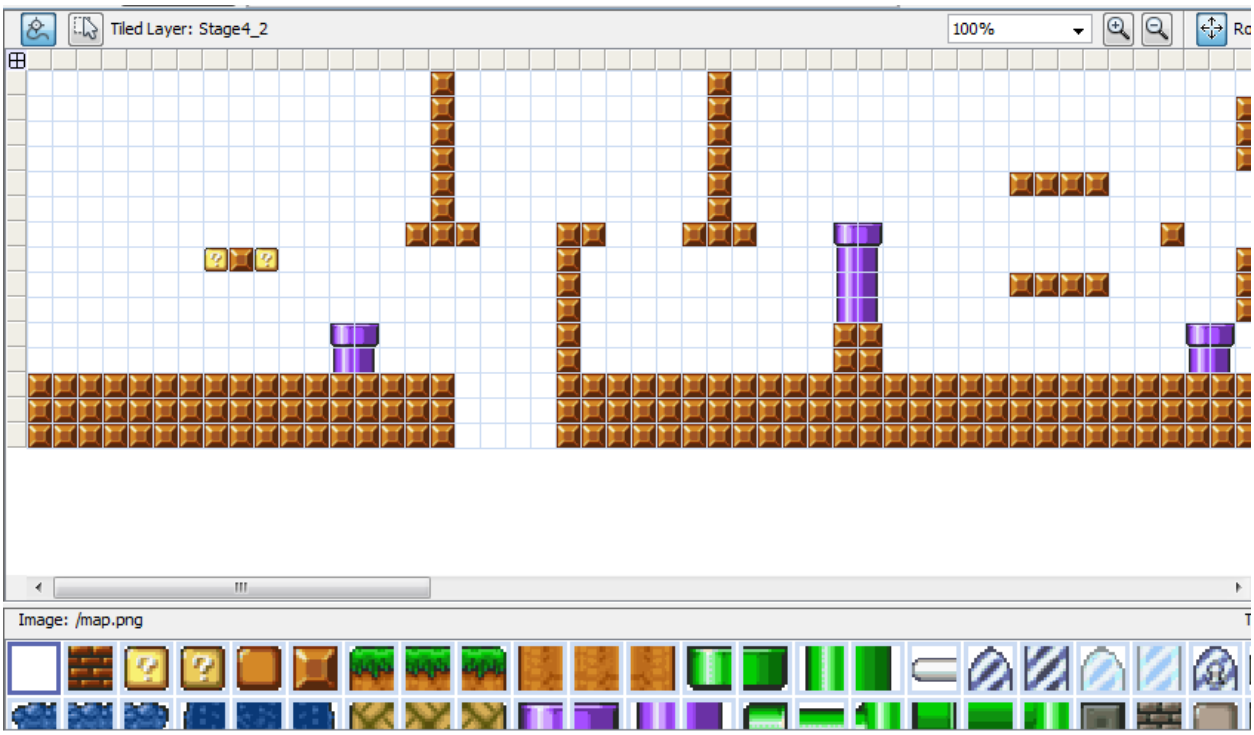


Tại đây ta dùng chuột đưa từng frame đã chia ở bước trước vào các vị trí hợp lí. Sắp xếp xong ta ấn nút Play để chạy thử sequences vừa tạo.

Ta thấy rằng với GameBulder việc tạo Sprites và sequences rất dễ dàng. Để có thể hiểu rõ hơn ta có thể xem source code mà GameBuilder sinh ra, ta sẽ thấy nó vẫn theo những lý thuyết đã được nêu ra ở trên. Chỉ có sự khác biệt ở chỗ GameBuilder hỗ trợ xây dựng một cách trực quan mà thôi. Để xem source code ta ấn vào nút Source bên cạnh nút Game Builder.

2. Tạo TiledLayer bằng GameBuilder :

Việc tạo TiledLayer gần giống với việc tạo Sprite. Trong khung TiledLayer của GameBuilder ta chọn Create Tiledlayer. Trong cửa sổ hiện ra ta chọn ảnh rồi tiến hành chia ảnh đó thành các Tile phù hợp với yêu cầu.



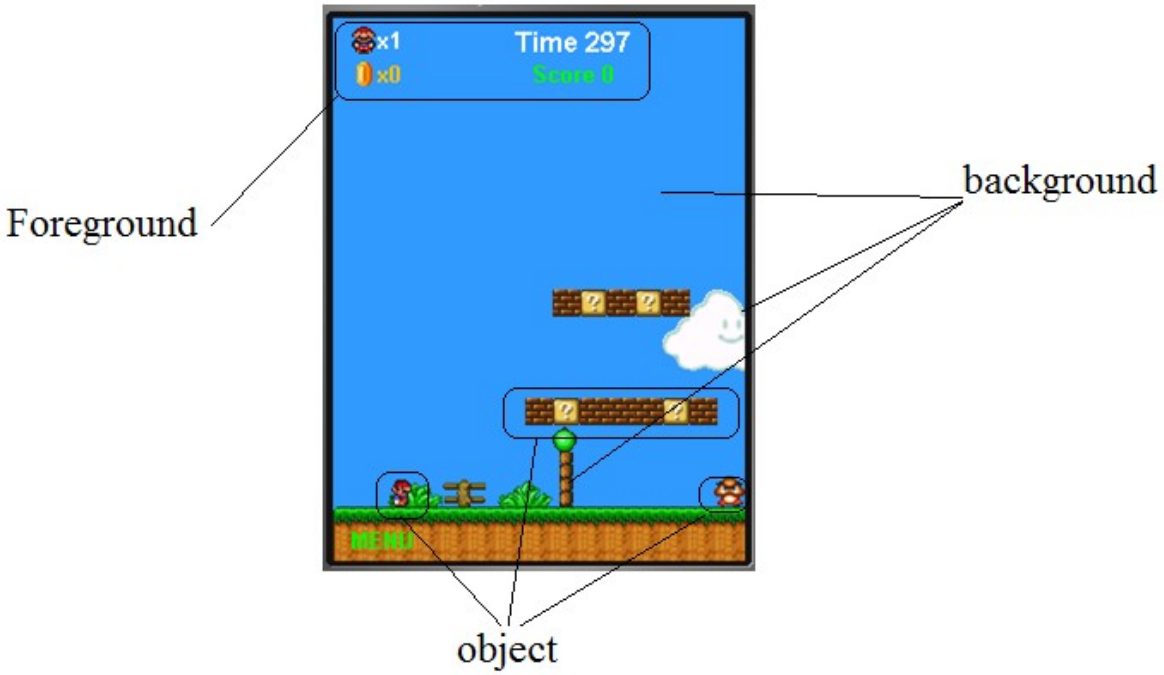
3. Tạo Scenes :

Hậu cảnh (Background): Phần hiển thị dưới cùng của một cảnh game, nó đóng vai trò làm nền cho cảnh đó. Hậu cảnh thường là hình ảnh bầu trời, mây, mặt đất, mặt nước…

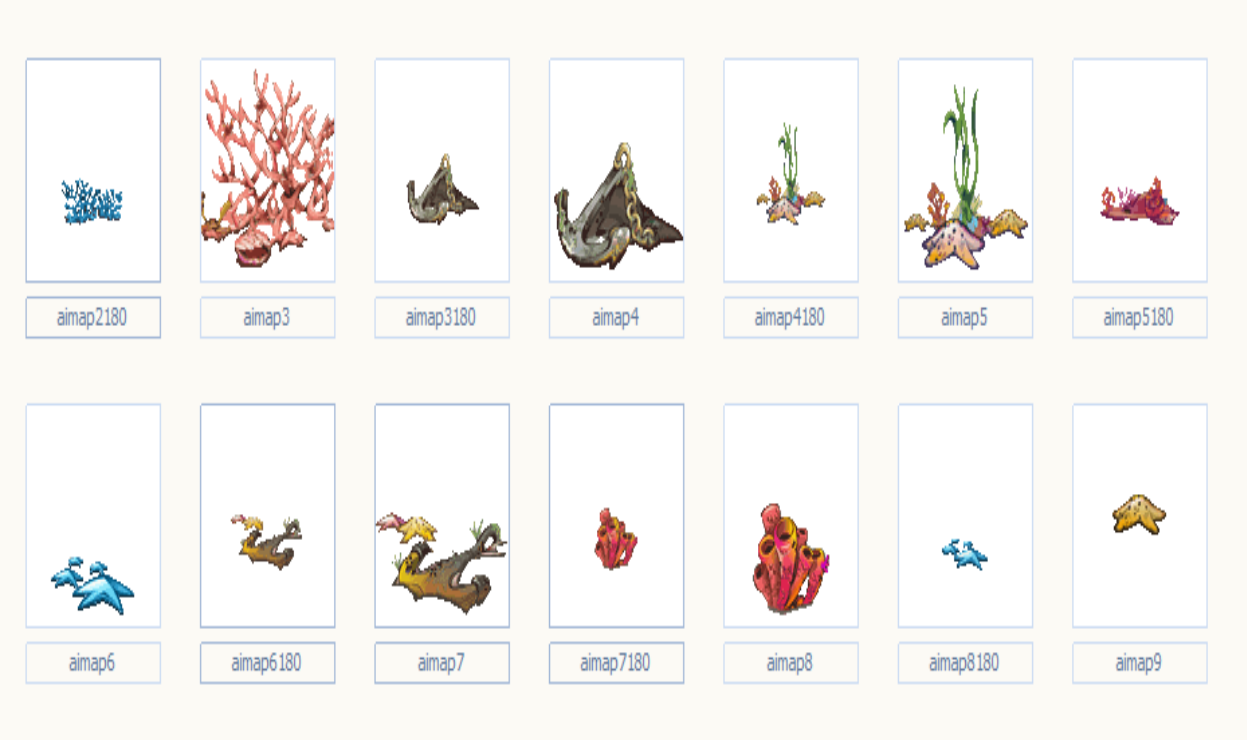
• *Vật thể (Object):* Phần được đặt trên hậu cảnh và có thể vận động hay chuyển động được. Vật thể thường bao gồm cả nhân vật của game, vật phẩm, bàn ghế, cây, đá…

• *Tiền cảnh (Foreground):* Phần nằm trên cùng của một cảnh nó nằm trên vật thể thường là để hiển thị các thông số như cột máu, các kỹ năng, mạng…

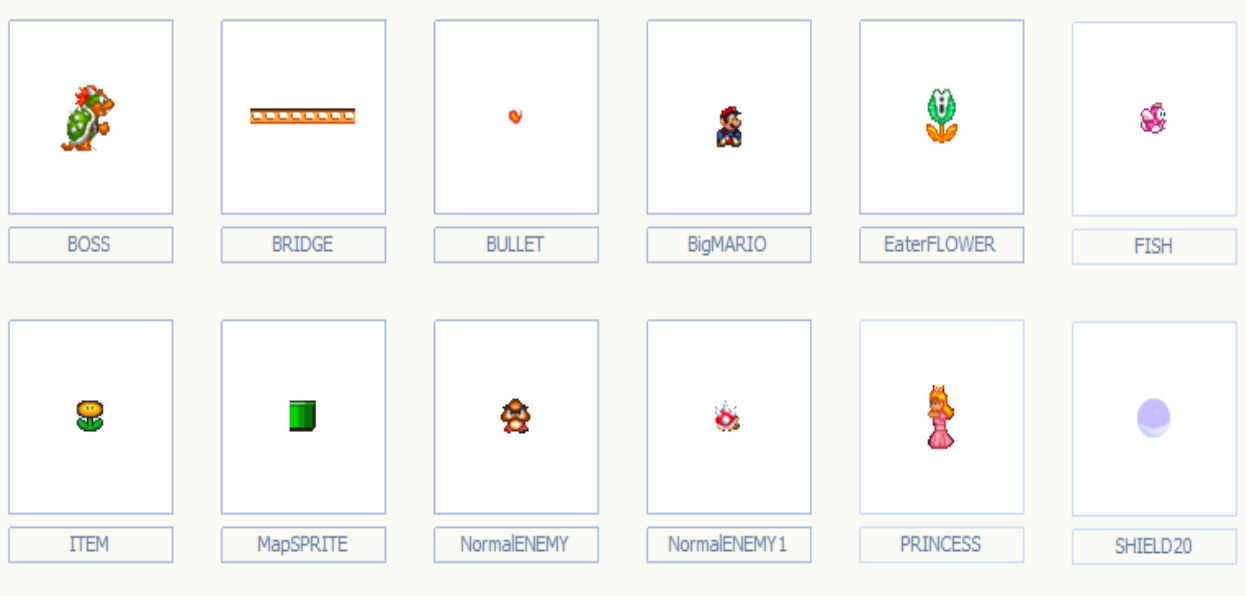
Dưới đây là một số hình ảnh trong việc thiết kể ứng dụng và hình ảnh in-game



Game FishingCraze



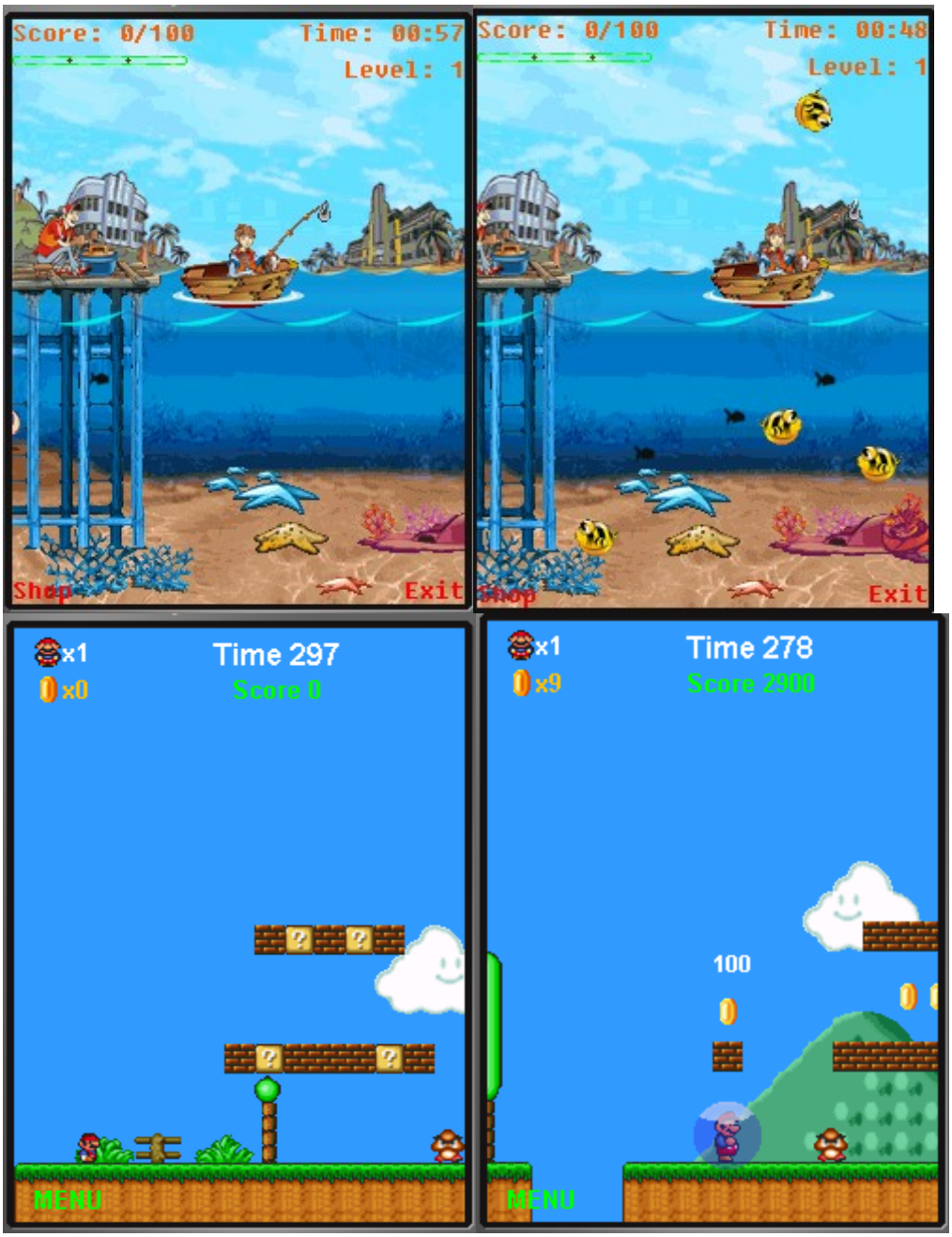
Game Mario



Game Bảo vệ biên cương

Ảnh in-game

Lập trình game trên di động với J2ME



Tổng kết và đánh giá

Qua một quá trình nghiên cứu hoàn thành Bài tập lớn này, chúng em đã có được kiến thức tổng quát về các lớp đồ họa trong J2ME và có thể viết được một số ứng dụng minh họa ,tuy lượng kiến thức thu thập được còn ít ỏi ,nhưng từ đó chúng em sẽ có cơ sở để đào sâu và phát triển những ứng dụng thành công hơn trên J2ME.

Do thời gian và trình độ có hạn , đề tài này không khỏi có những thiếu sót , rất mong nhận được sự góp ý của thầy và các bạn