Monografia

Sviluppo di un gioco J2ME

Alberto Realis-Luc
Mat. 106693

24/8/2004

Indice

Capitolo 1
Il contesto

1.1 La piattaforma J2ME

I cellulari che supportano applicazioni Java in genere, utilizzano la piattaforma J2ME (*Java 2 Micro Edition*) che è una versione di Java rilasciata da *Sun* (le altre sono: J2SE *Java 2 Standard Edition* e J2EE *Java 2 Enterprise Edition*) concepita appositamente per quei dispositivi con risorse hardware molto limitate rispetto ai tradizionali PC. La maggior parte di questi dispositivi sono appunto cellulari e palmari (ma potrebbero anche essere, per esempio, i dispositivi di controllo per forni a microonde) che per via delle dimensioni ridotte e dell'alimentazione a batteria sono costretti ad avere memorie RAM e capacità di calcolo molto limitate. Le memorie sono in genere di pochi MB o talvolta veramente poco come 128 KB, mentre la frequenza di lavoro dei processori si aggira intorno ai 100 MHz. J2ME è attualmente formato da due profili: "*The foundation profile*" e MIDP; il primo è stato ideato per la prossima generazione di dispositivi mobili, mentre il secondo è attualmente usato per sviluppare applicazioni per cellulari J2ME compatibili. Ogni profilo comprende un set minimo di librerie con le istruzioni per usare le varie funzionalità di tali dispositivi, una sua particolare *virtual machine* e una configurazione che può essere: CDC *Connected Device Configuration* oppure CLDC *Connected Limited Device Configuration*; la prima è un'implementazione del *foundation profile* mentre la seconda è un'implementazione di MIDP per dispositivi con poche risorse hardware. Esiste anche una seconda versione di MIDP, con alcune librerie in più, ma non è supportata da tutti i cellulari. In realtà ogni dispositivo utilizza, oltre alle librerie standard di MIDP, anche librerie aggiuntive, che contengono il codice per poter usare tutte le particolari funzionalità di quel dispositivo e sono fornite direttamente dal produttore. Quindi un applicazione MIDP sviluppata per un particolare telefono usando le particolari funzioni che necessitano altre librerie apposite non potrà funzionare su altri telefoni che non supportano tali librerie. Ogni telefono ha una sua *virtual machine* che esegue i programmi J2ME, e l'applicazione viene sottoposta alla *virtual machine* sotto forma di file compresso .jar, che contiene il codice compilato (files .class) e le risorse usate dal programma, come ad esempio immagini e suoni (.png, .wav, .mid). Ogni file .jar è sempre accompagnato da un .jad, ovvero un file di testo contenente alcune informazioni sul programma, come: dimensione del file jar, nome dell'applicazione, nome del file contenente l'icona del programma, classe che contiene il *main*, versione di MIDP, e altre informazioni sul produttore del programma.

1.2 L'ambiente di sviluppo

Per poter sviluppare applicazioni MIDP occorre innanzitutto il *J2ME wireless toolkit* (scaricabile dal sito www.java.com), che oltre ad una serie d'utili *tool* per compilare, fare package ed eseguire con appositi emulatori standard le applicazioni J2ME, comprende ovviamente le librerie J2ME. I produttori di cellulari mettono a disposizione tool simili, come per esempio il *Nokia Developer's Suite for J2ME*, ma la cosa più importante che si deve avere dai produttori sono gli emulatori dei telefoni con cui si testeranno le applicazioni. Gli emulatori sono in genere compatibili anche con il *J2ME wireless toolkit* della Sun. Per quanto riguarda Nokia tutto il materiale è reperibile su: www.forum.nokia.com . Quando si lancia un'applicazione MIDP occorre quindi specificare quale emulatore si intende usare. L'emulatore permette di eseguire l'applicazione su di un normale PC, ma bisogna sempre ricordarsi che il computer è molto più veloce del cellulare; è quindi bene, durante lo sviluppo, testare talvolta il programma anche sul telefono, per rendersi conto dei tempi effettivi di caricamento ed esecuzione. Ciò è di fondamentale importanza nei videogiochi, che devono sempre rispondere prontamente ai comandi dell'utente.

Dato che il codice compilato in java è facilmente disassemblabile, ci si può premunire usando un particolare tool, *proguard*, da aggiungere al *wireless toolkit*. Con esso è possibile offuscare il codice: i nomi delle classi, funzioni e variabili sono sostituite con lettere dell'alfabeto, e ciò rende quasi impossibile la comprensione del codice. Inoltre *proguard* ha anche i meriti di ottimizzare il codice, creando un package di dimensioni leggermente minori a quello non offuscato. La classe contenente il main non può essere offuscata, quindi è bene inserirvi solo il codice indispensabile per far partire il programma, rimandando il tutto su altre classi che saranno offuscate.
Per scrivere il codice in teoria andrebbe bene un qualsiasi editor di testo, ma Eclipse (www.eclipse.org) si è rivelato particolarmente adatto allo scopo dato che è possibile scaricare un apposito *plugin* da eclipseme.sourceforge.net che permette di sviluppare applicazioni J2ME usando le librerie del *J2ME wireless toolkit* e del *Nokia Developer's Suite*, e di lanciare l'applicazione con il comando *Run* direttamente nell'emulatore desiderato. Anche Nokia ha appena rilasciato una versione della sua *Developer's Suite* compatibile e integrabile con Eclipse.

Fig. 1.1 Tre emulatori della Nokia: il primo per serie 60 è molto usato specie in fase di sviluppo, il secondo è l'emulatore per serie 40. Alcuni emulatori comprendono anche tutto il sistema operativo. Nell'ultima immagine abbiamo l'emulatore per uno specifico cellulare: il 7210. Gli emulatori possono comunicare tra loro; per esempio, se si vuole testare un gioco multiplayer si può fare fra più emulatori in esecuzione su PC diversi, ma anche sullo stesso PC oppure tra emulatore ed un cellulare vero.

Capitolo 2
Il videogioco richiesto dall'azienda

2.1 Introduzione
Secondo Impressionware si vendono di più giochi semplici rispetto a quelli più elaborati e complessi, vale a dire giochi del tipo *puzzle-game* che sono piuttosto intuitivi e soprattutto immediati così che l'utente possa giocare in brevi intervalli di tempo (in ascensore, mentre si aspetta l'autobus, mentre si è in fila alla posta...) Per sua natura, un gioco di questo tipo non deve essere corposo, ma sarà più apprezzato se molto curato fin nei minimi particolari: un esempio è il menù del gioco, che darà un'immagine migliore del prodotto se ridisegnato in tema con il gioco anziché realizzato usando il menù standard del cellulare (anche se in quest'ultimo modo la sua realizzazione sarebbe molto più semplice.)

2.2 Requisiti del gioco richiesto
Il gioco deve essere MIDP 1.0 in modo che possa essere supportato da più cellulari possibili, deve occupare meno di 50 KB e deve essere inizialmente sviluppato per telefoni Nokia S60 (dovrà poi essere adattato anche per Nokia S40). Le partite devono poter essere messe in pausa, anche per l'arrivo di una telefonata, e si deve poter salvare la partita per poterla riprendere in un secondo momento. Inoltre si deve avere un menù di gioco personalizzato con sfondo in movimento che permetta di cambiare la lingua e disattivare il sonoro e la vibrazione (non presente su S60). Le più importanti caratteristiche possono essere riassunte dal file jad:

```
MIDlet-1: JammyBreeze, /icon.png, TheMain
MIDlet-Description: Tris con pinguini colorati
MIDlet-Icon: /icon.png
MIDlet-Info-URL: www.impressionware.com
MIDlet-Jar-Size: 46301
MIDlet-Jar-URL: JammyBreeze_S60.jar
MIDlet-Name: JammyBreeze
MIDlet-Vendor: Impressionware
MIDlet-Version: 1.0.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

La prima linea, di fondamentale importanza, indica il nome dell'applicazione che sarà visualizzato nel menù applicazioni del telefono, la relativa icona (contenuta nel jar) e la classe che contiene il main dalla quale inizierà l'esecuzione. Le stesse informazioni sono contenute nel *manifest* file .MF, contenuto anch'esso nel jar.

2.3 Le regole del gioco
Lo scopo del gioco consiste nel fare più tris possibili in una matrice di pinguini colorati. Muovendo un cursore con i tasti direzionali del telefonino si sceglie un punto (nel gioco rappresentato da un pallino di neve), quindi si possono ruotare di una sola posizione i quattro pinguini ad esso adiacenti, in senso orario o antiorario. Se dopo la rotazione si sono allineati almeno tre pinguini dello stesso colore in orizzontale o verticale o in una delle due diagonali, allora questi esplodono e vengono sostituiti dai pinguini sovrastanti, che cadono prendendone il posto (ogni pinguino esploso incrementa il punteggio di 5 punti). In cima alla matrice vengono generati nuovi pinguini a caso. Per abbellire il gioco i pinguini devono eseguire determinate animazioni mentre esplodono, cadono e ruotano, ed a caso ogni tanto qualche pinguino deve "fare l'occhiolino". Se dopo la caduta si formano altri tris anche questi esploderanno, e daranno così origine ad altre cadute. Se si fa una mossa che non forma alcun tris questa sarà annullata, ed il tempo disponibile per completare il

livello sarà penalizzato di 3 secondi (questo per evitare che i giocatori provino a caso le mosse per trovarne una che permetta di fare tris). Ogni livello è caratterizzato dal numero di possibili colori dei pinguini (più colori ci sono più sarà difficile fare tris), dal tempo disponibile e dai punti necessari per completare tale livello. A partire dal livello 3 iniziano a comparire elementi speciali che possono essere "eschimesi" o blocchi di ghiaccio. Gli eschimesi possono essere fatti ruotare, ma non esplodere, e i blocchi di ghiaccio non si possono neanche ruotare e quindi impediscono tutte le mosse possibili dai quattro punti che li circondano. Eschimesi e blocchi di ghiaccio possono comunque cadere: l'unico modo per sbarazzarsi dei blocchi di ghiaccio è farli cadere fino al fondo della matrice, e solo a questo punto esplodono, incrementando di 20 punti il punteggio totale. Se non si riesce a raggiungere il punteggio prefissato del livello entro un certo tempo, che tende a diminuire per i livelli più difficili, la partita termina con la classica schermata "Game over". Queste sono in sintesi le regole del gioco; da qui si può incominciare ad analizzare i principali problemi, costruire un diagramma di flusso e vedere a prima vista quali possano essere le soluzioni migliori.

Figura 2.1 Una schermata del gioco ripresa dall'emulatore S60. In alto a sinistra l'indicatore di punteggio, mentre a destra viene indicato il livello e subito sotto una sbarretta indica la percentuale di livello completato. Sul lato destro della schermata la sbarra del tempo: la parte verde indica il tempo ancora disponibile per completare il livello. In basso i bottoni virtuali per fare ruotare in senso orario o antiorario i pinguini; tali bottoni si animano nel momento in cui l'utente preme i pulsanti reali corrispondenti, cioè i due softkey, i due tasti che stanno subito sotto lo schermo del cellulare. Nel gioco sono presenti pinguini di 6 colori diversi, 4 eschimesi e 4 blocchi di ghiaccio. In tal caso se si vuole fare un tris, per esempio, si potrebbe muovere il cursore in alto di 2 posizioni, a sinistra di una posizione e quindi fare una rotazione in senso antiorario; i primi 3 pinguini della prima riga darebbero così origine ad un tris.

2.4 Principali problemi a prima vista

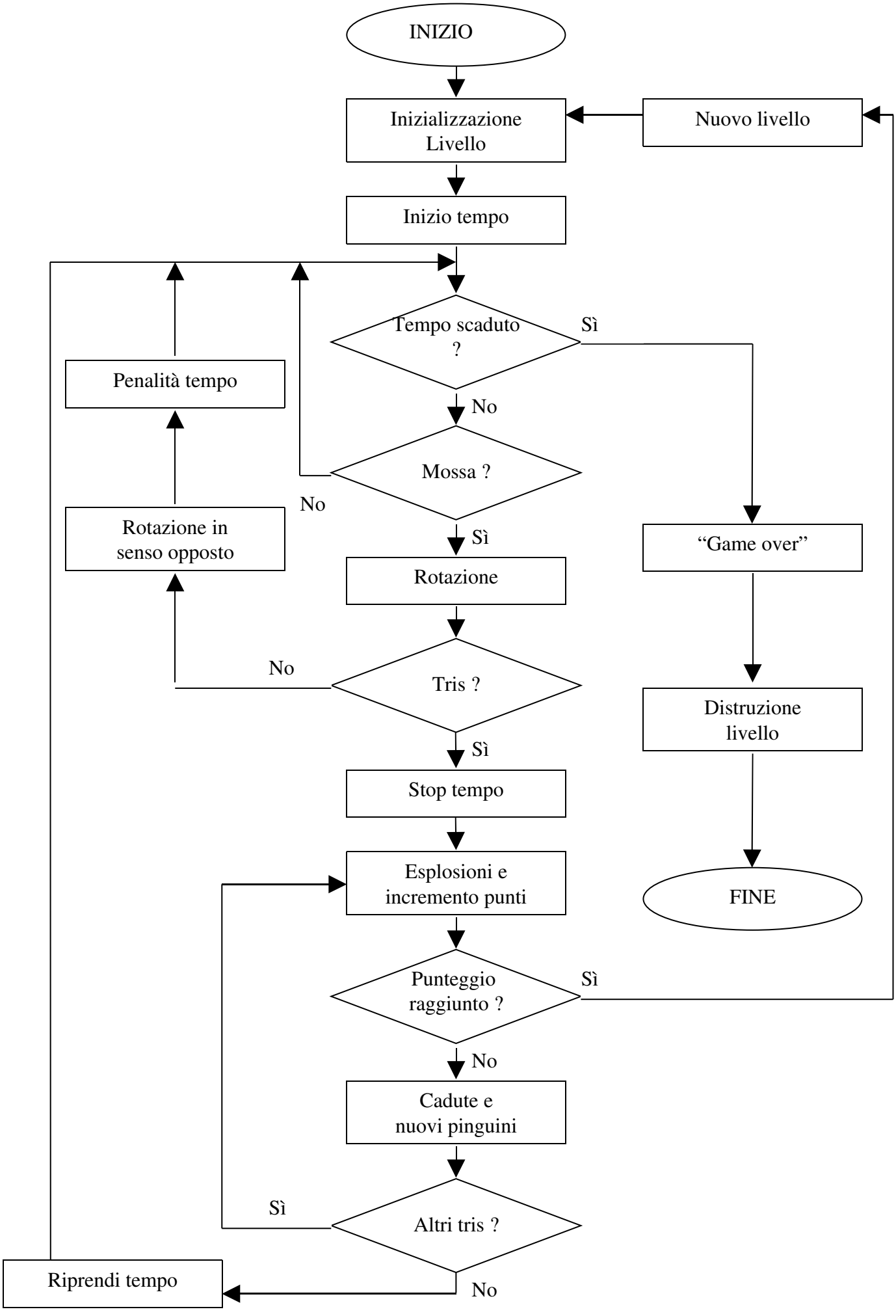
I primi problemi affrontati sono stati costruire il menù e utilizzando la grafica fornita dall'azienda, provare a visualizzare una schermata di gioco per trovare la gusta posizione in pixel per ogni elemento.

Per quanto riguarda il funzionamento del gioco, invece, il primo problema che si presenta è quello di realizzare una struttura dati per memorizzare ogni informazione riguardo lo stato della partita durate ogni sua fase. In questo caso si pensa subito ad una matrice, ed in java risulta comodo creare un oggetto "pinguino", con la sua classe, in cui ci si potrà memorizzare il colore oppure se è un eschimese o un blocco di ghiaccio, ed altre eventuali informazioni. La matrice sarà dunque formata da oggetti pinguino. Bisogna poi pensare ad una funzione "tris" che verifichi se si sono formati tris e che "segni" quali pinguini dovranno esplodere. Un altro problema importante è dato dal fatto che, all'inizio del gioco, la posizione dei pinguini non deve essere completamente casuale, perché in tal modo si potrebbero formare sin dal principio dei tris che darebbero avvio al gioco senza alcun intervento attivo dell'utente. Infatti nei primi livelli, essendoci pochi colori, è molto probabile che ad ogni tris ne seguano diversi altri a catena. Occorre quindi una funzione che inizializzi la matrice dei pinguini casualmente, ma evitando la formazione di tris. Una soluzione a prima vista può essere continuare a generare a caso la matrice verificandola con la funzione tris, finché non si trova una matrice senza tris; ma questa soluzione non è molto efficiente, perché questi ripetuti tentativi

potrebbero richiedere molto tempo, specie su un dispositivo con poche risorse di calcolo come un cellulare.

2.5 La gestione dello stato del gioco

Facendo un diagramma di flusso si può rendere bene l'idea di come funziona il gioco. Le fasi e quindi gli stati del gioco sono: attesa, rotazione, esplosioni e cadute. Il tempo disponibile per completare il livello viene fatto trascorrere solo nelle fasi in cui l'utente deve decidere quale mossa fare, mentre viene fermato durante le esplosioni e le cadute dei pinguini, fasi durante le quali il giocatore non può fare nuove mosse, ma deve attendere che la situazione si stabilizzi. Per quanto riguarda la parte dove ci si aspetta una mossa dall'utente, il programma chiama le funzioni per la rotazione quando l'utente preme i due *softkey*, destro e sinistro, che comandano rispettivamente la rotazione oraria e la rotazione antioraria. L'utente non potrà fare rotazioni mentre sono in corso esplosioni o cadute ma in ogni caso potrà muovere il cursore con i tasti direzionali, per prepararsi ad eventuali mosse successive già pensate.


INIZIO
Inizializzazione Livello
Nuovo livello
Inizio tempo
Tempo scaduto ?
Sì
Penalità tempo
No
Mossa ?
No
Rotazione in senso opposto
Sì
"Game over"
Rotazione
Tris ?
No
Distruzione livello
Sì
Stop tempo
Esplosioni e incremento punti
FINE
Punteggio raggiunto ?
Sì
No
Cadute e nuovi pinguini
Altri tris ?
Sì
Riprendi tempo
No

Capitolo 3
Lo sviluppo del gioco

3.1 Introduzione
A partire da tutte le considerazioni fatte nel secondo capitolo si può iniziare lo sviluppo del gioco; questo capitolo tratterà di ogni singolo problema con la relativa soluzione adottata. Prima di analizzare le varie soluzioni dei problemi è utile dare uno sguardo alle varie classi che formano il programma che sono: Campo, Pinguino, ScrollingBack, Sprite, TheCanvas, TheGame, TheMain e TheOptions.

3.2 Descrizione delle classi

3.2.1 La classe TheMain
Questa classe contiene il main che avvierà il programma e dato che non può essere offuscata contiene poche linee di codice:

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

public class TheMain extends MIDlet implements CommandListener {
private Display display = null;
private TheCanvas canvas = new TheCanvas(this);

public TheMain() {
    display = Display.getDisplay(this);
    display.setCurrent(canvas);
}

protected void startApp() throws MIDletStateChangeException {}

protected void pauseApp() {}

protected void destroyApp(boolean p0) throws MIDletStateChangeException {
    display.setCurrent(null);
    canvas.Destroy();
}

public void commandAction(Command p0, Displayable p1) {}

public void exit() {
    notifyDestroyed();
}

}
```

Infatti in questo caso questa classe contiene solo l'indispensabile per avviare un applicazione MIDP 1.0, e tutte le altre operazioni vengono subito redirette sulla classe TheCanvas.

3.2.2 La classe TheCanvas
Questa classe si occupa di gestire i *thread*, necessita della libreria *FullCanvas* della Nokia che viene così importata : `import com.nokia.mid.ui.FullCanvas;` Dentro questa classe ci sono funzioni per creare e distruggere *thread*, la funzione *paint* che ridisegna tutto lo schermo appoggiandosi sulle varie funzioni *render* di ogni oggetto che deve essere visualizzato in quel momento. Anche le funzioni *run* e *tick* sono contenute in TheCanvas. La procedura *tick*, che si appoggia alla classe TheGame, ha il compito principale di aggiornare, per esempio: se è in corso una partita verrà

chiamata la funzione *update* della classe Campo che farà avanzare di alcuni pixel i pinguini in movimento o ne cambierà l'animazione di altri ecc. In pratica la funzione *tick* va ad aggiornare ogni elemento dinamico del programma servendosi delle varie procedure *update* di ogni oggetto interessato. La funzione *run* in una specie di ciclo infinito continua ad invocare le funzioni *tick* e *paint*, e quindi fa girare il programma.

3.2.3 La classe TheGame

Qui è contenuto il codice del menù principale, le varie funzioni per caricare immagini e suoni, avviare, riprendere, salvare e caricare partite di gioco. In questa classe si trovano anche le procedure per riprodurre suoni e visualizzare le schermate introduttive di ogni livello e la schermata "*game over*". Sono in questa classe anche le varie funzioni *destroy* che vengono chiamate all'uscita dal programma per liberare la memoria sia da immagini e suoni precedentemente caricati sia da strutture dati particolarmente ingombranti come matrici e vettori. Per fare queste operazioni di pulizia prima si impostano a *null* tutti questi elementi e poi si usa il comando: System.gc() che va fisicamente a deallocare la memoria da tutti i dati messi a *null*.

3.2.4 La classe TheOptions

In questa classe si trova una matrice di stringhe che ricreano tutte le varie voci di menù nelle varie lingue. Qui ci sono anche le procedure per salvare e caricare le impostazioni del gioco, che vengono mantenute anche se si esce dal gioco.

3.2.5 La classe Pinguino

Come già detto inizialmente il gioco si basa su una matrice di oggetti pinguino; vediamo dunque cosa c'è dentro ogni pinguino:

```
public class Pinguino {

...

private int colore;  //colore del pinguino
private Sprite anim; //immagine corrente del pinguino
private boolean morto; //indica se il pinguino è esploso completamente
private int currentPos=0; //pixel percorsi durante le animazioni
private boolean turning,falling,exploding;
            //indicano se il pinguino sta girando, cadendo o esplodendo
private boolean axis,direct; //se il pinguino gira dicono dove sta andando
private boolean eye; //indica se il pinguino sta facendo l'occhiolino

...
}
```

3.2.6 La classe Campo

La matrice dei pinguini non è altro che l'oggetto campo. Qui ci sono le funzioni: *init* che inizializza la matrice dei pinguini senza tris, la procedura tris che verifica se ci sono tris nella matrice, la cascaPinguini che fa cadere i pinguini a seguito delle esplosioni e le funzioni per fare ruotare i pinguini.

3.2.7 La classe Sprite

Come si vedrà nei successivi paragrafi la classe Sprite serve per le animazioni. Ecco i dati che comprende:

```
public class Sprite {
      private int width;      //larghezza del frame (in pixel)
      private int height;     //altezza del frame (in pixel)
      private int nFrames;    //numero totale di frame dell'animazione
```

```
        private int XcurFrame;  //frame corrente orizzontatale
        private int YcurFrame;  //frame corrente verticale
        private Image image;    //immagine che comprende tutti frame

...
}
```

Un *frame* è una parte di un'immagine più grande, formata da tanti *frame* allineati, che tipicamente rappresenta un fotogramma di un animazione.
Il numero di *frame* corrente orizzontale si usa solo per i pinguini; come si vedrà tutti i *frame* dei pinguini sono disposti su di una sola immagine, su più colonne dove ogni colonna rappresenta i *frame* di un pinguino di un certo colore. Infatti questo numero corrisponde con quello che indica il colore del pinguino.

3.2.8 La classe ScrollingBack

Questa classe, come si vedrà successivamente, si occupa di rendere animato lo sfondo del menù usando un'immagine ripetuta (come sulla carta delle confezioni regalo) che scorre diagonalmente sotto le voci del menù.

3.3 La grafica

3.3.1 Le immagini png

La grafica del gioco è principalmente costituita da immagini png, che verranno anch'esse messe dentro al file jar che porterà il gioco completo.

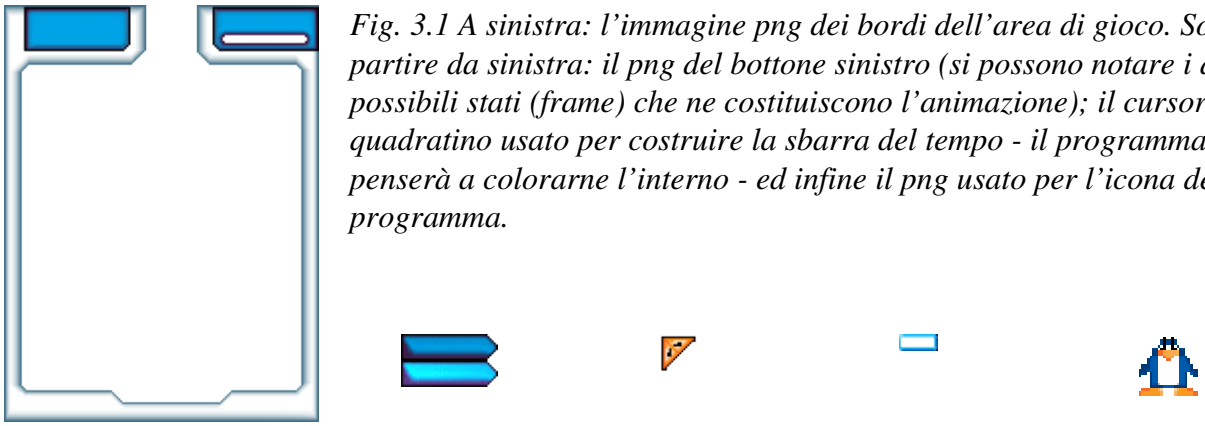


Fig. 3.1 A sinistra: l'immagine png dei bordi dell'area di gioco. Sotto a partire da sinistra: il png del bottone sinistro (si possono notare i due possibili stati (frame) che ne costituiscono l'animazione); il cursore; il quadratino usato per costruire la sbarra del tempo - il programma penserà a colorarne l'interno - ed infine il png usato per l'icona del programma.

Dato che il gioco deve occupare poco spazio è importante che anche la grafica non sia troppo ingombrante. Un buon modo per diminuire le dimensioni totali della grafica è quello di inserire nella stessa immagine più oggetti, per esempio: le immagini del bottone premuto e rilasciato sono nello stesso file come si può vedere in fig. 3.1. Nel programma si dividerà l'immagine in *frame*, visualizzando solo quello desiderato. Con particolari *tool* è inoltre possibile ridurre drasticamente le dimensioni dei png. Se si ha la grafica condensata in pochi file png caricandoli solo una volta all'inizio del programma si riducono anche i tempi di caricamento; caricare tante immagini di singoli oggetti richiederebbe più tempo per il fatto che si dovrebbe accedere a tanti file.

Fig. 3.2 Il png con i pinguini di tutti i colori e le relative animazioni. Le colonne rappresentano i colori: 0 per verde, 5 per viola mentre per colore 6 si ha l'eschimese. Le righe invece rappresentano le varie animazioni. Riga 0: stato normale; righe 1 e 2: animazione del pinguino che cade; righe 3 e 4: animazione del pinguino che cammina verso destra; righe 5 e 6: animazione del pinguino che cammina verso sinistra;righe 7 e 8: animazione del pinguino che fa l'occhiolino. Questo file occupa 7 KB.

3.3.2 Gestire frame e sprite

Come si può vedere in figura tutte le immagini dei pinguini e degli eschimesi con tutte le loro animazioni sono in un unico file png, che viene caricato in memoria nel costruttore di Campo con l'istruzione: `allping = Image.createImage("/pinguini.png");` .
Per gestire i frame si usa la classe Sprite; ogni oggetto pinguino ha uno *sprite* che è l'immagine visualizzata in quel momento per quel pinguino. Nel costruttore di pinguino lo *sprite* viene inizializzato così: `this.anim=new Sprite(Campo.allping,L_PING,L_PING,colore,3);` .
Dove anim è lo *sprite* corrente di ogni pinguino, Campo.allping è l'immagine di tutti i pinguini caricata all'inizio, L_PING sono le dimensioni del *frame* nel nostro caso ogni pinguino deve occupare un quadrato di 19x19 pixel. Colore è un intero da 0 a 6 che indica quale colonna della fig. 3.2 dovrà essere usata, 6 per disegnare l'eschimese. 3 è il numero di *frame* per fare l'animazione; in questo caso viene predisposto per l'animazione della caduta, che utilizza le prime tre righe della figura 3.2. E' comunque bene usare solo la prima volta che si crea la matrice l'istruzione new Sprite, dato che ogni volta ne allocherebbe lo spazio in memoria,; per le volte successive si cambia semplicemente il frame, scegliendo dall'immagine allping quello giusto a seconda del colore e dell'animazione in corso, questo lo si fa con la funzione init della classe Sprite:

```
public void init(Image image, int w, int h, int colonna, int frames) {
    this.image = image;
    this.width = w;
    this.height = h;
    this.XcurFrame = colonna;
    this.YcurFrame = 0;
    this.nFrames = frames;
}
```

Sempre dalla classe Sprite con la funzione *render* si disegnano queste animazioni sullo schermo nella posizione desiderata (xPos, yPos).

```
public void render(Graphics g, int xPos, int yPos) {
    if(xPos<TheGame.screen_width && xPos+width>0 && yPos<TheGame.screen_height
&& yPos+height>0) {
      g.setClip(xPos, yPos, width, height);
      g.drawImage(image, xPos-XcurFrame*width, yPos-YcurFrame*height,
Graphics.TOP|Graphics.LEFT);
    }
}
```

Fig. 3.3 Altri esempi di immagini png formate da più frame. A sinistra il png formato da 5 frame usato per fare l'animazione del pinguino che esplode. A destra in alto l'animazione del bottone destro. Mentre in basso abbiamo il png usato per disegnare le palline di neve selezionabili con il cursore per ruotare i pinguini intorno; quando si seleziona una pallina e si fa una rotazione,e mentre i pinguini girano, questa diventa più piccola usando il frame alto dell'immagine.

Anche i caratteri usati nel menù sono degli *sprite*, e vengono caricati da un immagine fonts.png con tutto l'alfabeto.

3.3.3 Rendere più veloce la grafica

Grazie all'istruzione setClip si sceglie quale parte dell'immagine si vuole disegnare, ma dopo occorre reimpostare il *clip* a pieno schermo; usare troppo l'istruzione setClip può rallentare il gioco, quindi occorre impostare il *clip* a pieno schermo una sola volta, dopo aver disegnato tutti i pinguini. Come si può notare dal codice della funzione render si devono fare spesso moltiplicazioni, e anche questo potrebbe rallentare la visualizzazione, dato che mentre il gioco è in esecuzione la render viene continuamente invocata per disegnare i pinguini ogni volta; per ovviare a questo problema si possono fare moltiplicazioni e divisioni con operazioni di shifting a patto che si moltiplichi o si divida per multipli di 2. Purtroppo in questo caso le dimensioni dei pinguini sono di 19x19 pixel. Infatti per quanto riguarda lo sfondo animato in movimento del menù è stata appositamente creata un'immagine di 64x64 pixel, cioè 2^6. Il codice (dalla classe ScrollingBack) della render dello sfondo del menù è:

```
public void render(Graphics g) {
      for(int y=0;y<m_nYtiles;y++)
            for(int x=0;x<m_nXtiles;x++)
                  g.drawImage(m_image, (x<<6)+m_offX-64, (y<<6)+m_offY-64,
Graphics.TOP|Graphics.LEFT);
}
```

Dove m_nYtiles e m_nXtiles sono il numero di immagini ripetute mentre m_offX e m_offY sono lo scostamento orizzontale e verticale, che vengono continuamente aggiornati da un'altra funzione. Lo *shifting* x<<6 è come fare la moltiplicazione x*64 molto più costosa in termini di calcolo, dato che in questo caso lo *shift* non fa altro che aggiungere 6 zeri a destra del numero binario x. Dato che mentre viene visualizzato il menù questa render dello sfondo viene continuamente invocata i vantaggi dati dall'uso dello *shifting* sono notevoli.

Fig. 3.4 A sinistra la schermata iniziale del menù di gioco con lo sfondo animato, che scorre diagonalmente verso il basso. A destra l'immagine di 64x64 pixel usata per disegnare lo sfondo; tale immagine viene continuamente ridisegnata su tutto lo schermo ogni volta leggermente sostata per dare l'impressione del movimento.

3.3.4 Misurare i fotogrammi per secondo

A seconda dei vari stati del gioco la funzione *paint*, della classe TheCanvas, chiama nel giusto ordine tutte le varie *render* dei vari oggetti che devono essere disegnati in quel momento. Per verificare l'efficienza della *paint* si può visualizzare un indicatore di fotogrammi per secondo istantanei; per far ciò occorre inserire il seguente codice nella *paint*:

```
g.setColor(0xffffff);
g.drawString(""+fps, 0, 0, Graphics.TOP|Graphics.LEFT);
```

fps viene calcolato nella funzione *run* sempre nella classe TheCanvas in questo modo:

```
long oldTime = 1;
long curTime;
long fps = 0;

...

curTime = System.currentTimeMillis();
fps = 1000 / (curTime - oldTime);
oldTime = curTime;
```

Tali istruzioni visualizzano nell'angolo in alto a destra un numero (di colore bianco, come vuole l'istruzione setColor) che indica i fotogrammi per secondo istantanei effettivamente visualizzati; questa misura va presa in considerazione solo usando il telefono e non l'emulatore. Con l'emulatore si hanno sempre valori alti, intorno ai 25-30 fps, che calano drasticamente se si usa il telefono, specie se si devono visualizzare molti oggetti (come spesso accade nei *puzzle-game*). Bisogna cercare di tenere questo valore sopra i 10 fps. Il principio fondamentale per accelerare la visualizzazione è quello di ridisegnare solo gli oggetti che si sono modificati; per esempio, nella sbarra del tempo si colora di rosso solo il quadratino corrispondente a quel momento temporale, ma non si ridisegnano tutti gli altri ogni volta.

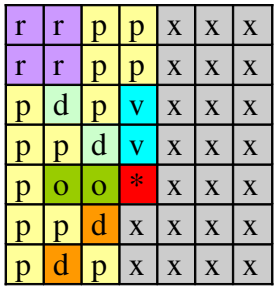
3.3.5 Caricare le immagini png e i suoni midi

A questo punto tutte le immagini verranno caricate in memoria solo una volta, all'avvio del programma: bisogna tener conto del tempo necessario per farlo, che risulta impercettibile sull'emulatore ma è notevole sul telefono. Nel nostro caso con il nokia 3660 sono necessari circa 7 secondi; durante questo tempo sarà necessario avvertire l'utente che si è in fase di "*loading*" altrimenti potrebbe credere che il programma si sia bloccato. In questo gioco l'azienda ha richiesto che all'avvio vengano mostrati i marchi dell'azienda stessa, ed eventualmente quello dell'operatore di telefonia che venderà il gioco, cosicchè durante questa fase l'utente è costretto ad attendere qualche secondo, mentre i diversi loghi compaiono sullo schermo in successione. Proprio durante questi pochi secondi, dopo il caricamento dei loghi che devono essere visualizzati nel frattempo, si possono caricare anche tutti i dati necessari al gioco e al menù: le immagini png e i file midi per i suoni. In tal modo la schermata "*loading*" non sarà più necessaria, e dopo aver visto i loghi l'utente può cominciare subito a giocare, senza ulteriori attese.

3.4 Generare la matrice dei pinguini

Come già detto all'inizio uno dei primi problemi affrontati è stato quello di costruire la matrice iniziale dei pinguini evitando i tris. Questo lavoro viene svolto dalla funzione *init* nella classe Campo. La soluzione adottata si basa su due cicli for annidati, con i contatori i per le colonne e j per le righe, che scandiscono la matrice di colonna in colonna. Il seguente schema illustra la situazione in cui i=3 e j=4.

Fig. 3.4 Schema della matrice



Legenda:

r

Pinguini inizializzati con colore completamente casuale.

p

Pinguini già inizializzati con colore casuale che non formano tris.

*

Pinguino corrente in corso di inizializzazione.

o

Pinguini già inizializzati che potrebbero formare, un tris orizzontale.

v

Pinguini già inizializzati che potrebbero comporre un tris verticale.

d

Pinguini già inizializzati che potrebbero realizzare un tris in diagonale 1.

d

Pinguini già inizializzati che potrebbero formare un tris in diagonale 2.

x

Pinguini non ancora inizializzati.

I primi quattro pinguini in viola, per cui i<2 e j<2, possono avere un qualsiasi colore casuale in quanto non possono formare tris; tutti gli altri invece vengono inizializzati con un colore a caso, a patto che non formi tris. Per esempio: dobbiamo trovare il colore per il pinguino in posizione i,j; se i due pinguini precedenti sulla stessa riga (quelli verde scuro dello schema) sono dello stesso colore, se facciamo comparire anche il nostro pinguino corrente in i,j di quello stesso colore formeremo un tris in orizzontale. Quindi quando si sceglie a caso il colore per il pinguino in i,j bisogna escludere i colori delle possibili coppie, di colore uguale, in orizzontale, verticale e nelle due diagonali. Qui però si presenta un problema: se abbiamo a disposizione solo 4 o meno possibili colori per i pinguini, e in orizzontale, verticale e in entrambe le diagonali sono presenti coppie di colori diversi, qualunque colore daremo al pinguino i,j si formerebbe di sicuro un tris. Questo problema si può risolvere in due modi. Il primo consiste nel posizionare in i,j un eschimese o un blocco di ghiaccio e decrementare di uno il numero di eschimesi o blocchi di ghiaccio richiesti in quel livello; ma questo metodo non può essere usato se non sono richiesti tali elementi speciali. Il secondo modo consiste nel decrementare di 2 posizioni i e j, in questo modo si riprenderebbe la sistemazione da due colonne indietro e i pinguini che formavano le coppie dello stesso colore verrebbero reinizializzati. Potrebbe capitare che ciò avvenga più volte, soprattutto se si usano solo 3 colori. Per questo motivo tutte queste operazioni avvengono su di una matrice di numeri interi, dove gli interi rappresentano i

colori: si cerca così di velocizzare questa procedura, componendo la matrice definitiva di oggetti pinguino basandosi sulla soluzione appena calcolata dalla matrice di interi. Prima di costruire la matrice finale, se richiesti, si calcolano a caso le posizioni degli oggetti speciali rimanenti.

3.4 Le rotazioni

Quando il giocatore seleziona un pallino di neve con il cursore e fa una rotazione, il programma entra in fase di rotazione: prima i dati dei pinguini vengono spostati nelle nuove posizioni della matrice, poi viene eseguita l'animazione dei pinguini che camminano a destra e a sinistra per quelli che si devono muovere orizzontalmente, e quella dei pinguini che cadono per quelli che si devono muovere verticalmente (in entrambi i casi lo spostamento è di una sola posizione).

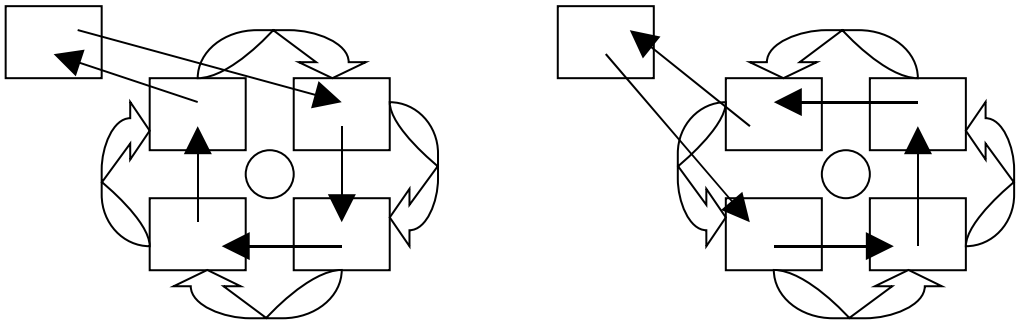


Fig. 3.5 Schema delle rotazioni oraria e antioraria. Le frecce sottili indicano come vengono copiati i dati; per prima cosa facendo uso di un pinguino allocato temporaneamente si salvano i dati del pinguino in alto a sinistra, che al passo successivo verrà soprascritto; i dati salvati temporaneamente serviranno per chiudere il ciclo. Le frecce più spesse indicano come avviene l'animazione: i pinguini che si muovono orizzontalmente usano l'animazione per camminare nella direzione della freccia, mentre quelli che si spostano verticalmente usano l'animazione per la caduta, spostandosi nel verso indicato dalla freccia.

Come illustrato nel diagramma di flusso del capitolo precedente, finita l'animazione la mossa viene verificata con la funzione tris: se ci sono dei tris il programma entra in fase di esplosioni e poi cadute. Se invece non si sono formati tris la mossa viene annullata con una rotazione in senso opposto, il tempo restante viene penalizzato e, finita l'animazione della contromossa, il programma ritorna in stato d'attesa. Oltre al senso della rotazione il programma memorizza anche la posizione del cursore, in modo che l'utente lo possa muovere anche durante l'animazione, senza però poter fare altre rotazioni.

3.5 La procedura tris

La funzione tris della funzione Campo ha il compito, a seguito di una mossa da parte dell'utente, di verificare se si sono formati dei tris e di impostare i pinguini che formano tali tris con exploding=true. Dopodiché il gioco entra nella fase delle esplosioni e quindi delle cadute. Questa funzione scandisce quattro volte la matrice con due cicli for annidati; le varie scansioni servono per rilevare i tris orizzontali, verticali ed in fine nelle due diagonali. I cicli for annidati eseguono le quattro scansioni come in figura:

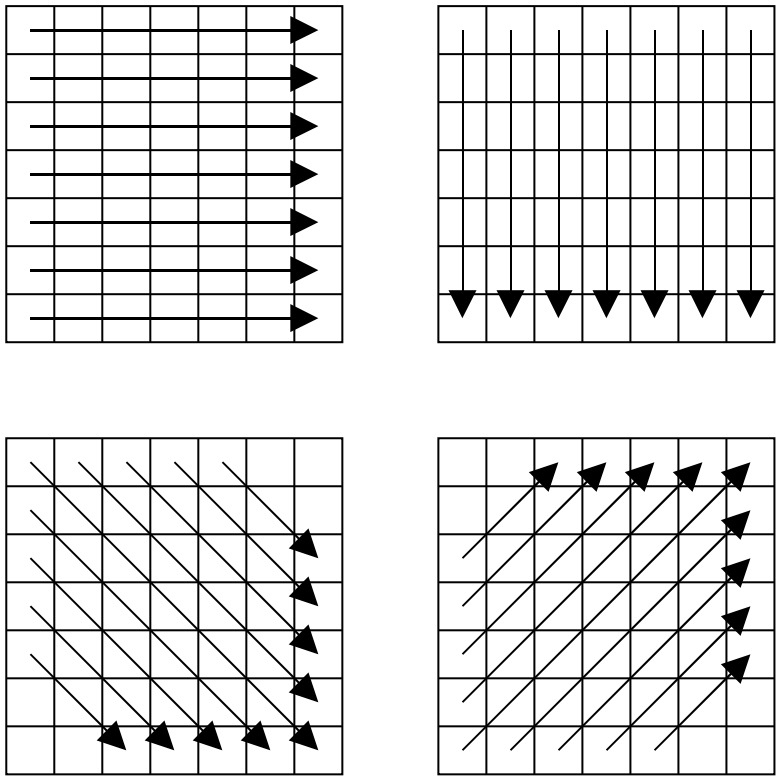


Fig. 3.6 Schema delle scansioni eseguite sulla matrice per cercare i tris. In alto a sinistra scansione dei tris orizzontali, a destra scansione dei tris verticali, in basso a sinistra scansione delle diagonali 1 mentre a destra scansione delle diagonali 2.

Se durante la scansione vengono rilevati 3 o più pinguini dello stesso colore in successione, allora questi vengono considerati come un tris. Segue una versione semplificata del codice eseguito su di ogni pinguino durante la scansione orizzontale.

```
if(colour==pinguini[i][j].getColour()) count++;
else {
      if(count>=3) {
            for(k=i-1;k>=i-count;k--) pinguini[k][j].kill();
            trovato=true;
      }
      count=1;
      colour=pinguini[i][j].getColour();
}
```

In pratica quando si trova un colore diverso da quello precedente, si controlla se c'è né sono almeno tre uguali precedenti. La funzione getColour della classe pinguino ritorna il numero che indica il colore del pinguino mentre la funzione *kill* sempre della classe pinguino va ad impostare exploding=true, e lo sprite del pinguino viene cambiato nel primo frame dell'animazione per le esplosioni di fig. 3.3.

3.6 Le esplosioni

Dopo che sono stati rilevati dei tris il programma da il via alle esplosioni, la funzione tris ha già stabilito quali pinguini devono esplodere; la loro animazione è ora lo spruzzo d'acqua in figura 3.3. Durante questa fase l'animazione per le esplosioni viene aggiornata con tutti i frame dello spruzzo in fig. 3.3, dopodiché questa fase termina: i pinguini esplosi sono ora considerati "completamente morti", e dovranno essere rimpiazzati dai pinguini soprastanti nella matrice che cadranno al loro

posto: inizia ora la fase delle cadute. La funzione *update* della classe Campo aggiorna le esplosioni ed una volta che sono finte passa alla fase delle cadute.

3.7 Le cadute

Appena si entra in stato di cadute la funzione *update* chiama la cascaPinguini - sempre nella classe Campo - che stabilisce dove ogni pinguino dovrà arrestare la propria caduta. In pratica, in ogni colonna tutti i pinguini cadono compattandosi e occupano il posto di quelli esplosi, mentre in cima alla colonna rimarrà un numero di posti liberi pari al numero di pinguini esplosi. Qui si faranno cadere dall'alto i nuovi pinguini, con colore completamente casuale. Anche in questo caso prima vengono copiati tutti i dati nelle posizioni desiderate, poi si procede a tutte le varie animazioni dei pinguini in caduta; in questo modo non c'è bisogno di allocare i nuovi pinguini in posizioni temporanee, ma si continua a lavorare sempre sulla stessa matrice.

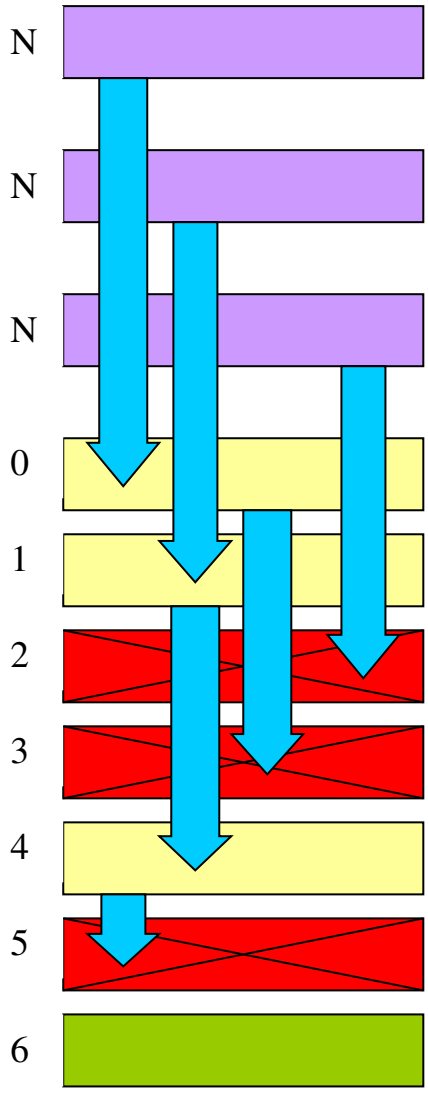


Fig. 3.7 Un esempio di cadute in una colonna in cui sono esplosi i pinguini in posizione 2, 3 e 5 (caselle rosse). Il pinguino più in basso in posizione 6 (casella verde) ovviamente rimane dov'è mentre i pinguini in posizione 0, 1, e 4 (caselle gialle) non sono esplosi ma hanno posti liberi al di sotto, quindi devono cadere compattandosi. In cima alla colonna rimarrebbero dunque tre posti liberi che vengono presi da tre nuovi pinguini (caselle viola).

Subito dopo che tutti i dati dei pinguini sono stati spostati nelle giuste posizioni della matrice vengono però visualizzati nelle loro posizioni precedenti grazie all'intero currentPos contenuto in ogni oggetto pinguino, che indica di quanti pixel deve essere disegnato più in alto rispetto alla posizione dei dati in matrice. In fase di cadute currentPos viene man mano decrementato in modo da far scendere i pinguini. Si potrebbe dare una parvenza di accelerazione di gravità decrementando currentPos in modo appropriato. Quando currentPos di un pinguino raggiunge 0 quel pinguino cessa la sua caduta. Quando sono finite tutte le cadute il programma esce da questa fase, e verifica con la

funzione tris se si sono formati altri tris a seguito di queste cadute, comportandosi esattamente come mostrato nel diagramma di flusso del capitolo precedente.

3.8 Il tempo

L'istruzione System.CurrentTimeMillis restituisce il numero di millisecondi trascorsi in quel momento dal 1 gennaio 1970. Quindi per memorizzare il momento d'inizio della partita si fa semplicemente così:

```
startTime = System.currentTimeMillis();
```

Mentre è in corso una partita, in tutti quei momenti in cui non sono in corso esplosioni o cadute e il gioco non è in pausa, ovvero in tutti quei momenti in cui il giocatore deve pensare a quale mossa sia più opportuna, la funzione *update* della classe Campo provvede ad aggiornare il tempo trascorso in tal modo:

```
time = System.currentTimeMillis()-startTime;
```

Dopo che la giocata è stata messa in pausa o dopo che ci sono state esplosioni e cadute, dato che è trascorso del tempo prima di riprendere a giocare va aggiornato il tempo d'inizio in tal modo:

```
startTime = System.currentTimeMillis()-time;
```

3.9 I suoni

I suoni vengono caricati da file *midi*. Anche le suonerie polifoniche non sono altro che dei file *midi*. Altri modelli di telefoni supportano anche i suoni in formato *wave*: per questi telefoni si potrà abbellire il gioco usando file wav. I file dei suoni .mid e quelli delle immagini .png sono tutti contenuti nel direttorio res (che sta per *resources*),contenuto nell'archivio jar con cui viene distribuito il gioco. Ogni suono viene caricato, come uno *stream*, in un oggetto *Player* che si potrà riprodurre durante il gioco al momento più opportuno. Dopo aver eseguito un suono, se si intende rifarlo, occorre "riavvolgerlo". Le varie istruzioni per gestire i suoni corrispondono ai comandi di un videoregistratore, con la differenza che per portarsi ad un certo punto del suono basta usare l'istruzione: setMediaTime(tempo). Quindi per riavvolgere e riprodurre un suono bastano queste istruzioni:

```
player[which].setMediaTime(0);
player[which].start();
```

Dove *player* è un vettore di oggetti *Player* in cui sono stati caricati i suoni e *which* indica il numero del suono che si intende riprodurre.

Capitolo 4
Conclusioni

Sviluppare applicazioni J2ME, per chi conosce Java, non è particolarmente difficile: occorre imparare ad usare le librerie J2ME (ad esempio javax.microedition.midlet) e quelle fornite dai produttori del dispositivo (ad es. com.nokia.mid.ui.FullCanvas). I produttori forniscono oltre alle librerie anche molto materiale utile per gli sviluppatori, la cosa più importante sono senza dubbio gli esempi: ottimo punto di partenza per chi inizia. Gli esempi sono semplicemente codici sorgenti di semplici programmi dimostrativi. Il codice di queste demo contiene molti commenti, che illustrano il compito delle istruzioni più complesse e dei passaggi più difficili. Questi programmi sono molto semplici e ridotti all'essenziale, ma mostrano come usare gran parte delle funzioni e librerie utili a sviluppare videogiochi molto più complessi. Esaminando e poi eseguendo il codice di questi esempi ci si rende conto subito di cosa fanno determinate istruzioni; poi, magari, si può provare a modificare qualcosa ,per vederne le conseguenze nel programma in esecuzione. Questo è, probabilmente, uno dei metodi più veloci per imparare a sviluppare videogiochi. Tutta la documentazione delle librerie è utile soprattutto da consultare durante lo sviluppo, per esempio quando si vuole saperne di più su di una certa funzione. Java offre indubbiamente molti vantaggi nello sviluppo di questi applicativi.
Un programma creato in java dovrebbe poter funzionare su qualsiasi dispositivo: la *virtual machine* si occupa di eseguirlo usando le istruzioni appropriate del processore di ogni dispositivo; questo dovrebbe essere uno dei principali vantaggi dell'uso di java. Il problema però è che dispositivi come i cellulari sono molto diversi tra loro, e per poter usare particolari funzioni di un telefono bisogna solitamente fare ricorso alle librerie fornite dal produttore. Per cui lo stesso programma va adattato ad ogni tipo di telefono, eliminando o aggiungendo caratteristiche a seconda dalle capacità hardware di ogni modello; per esempio, se il cellulare lo supporta, si potrà comporre, il sonoro con file *wave* piuttosto che con *midi*, si potrà modificare la grafica in modo che funzioni anche con altre risoluzioni,ecc. Per cui non è affatto detto che una volta sviluppato un applicativo J2ME questo poi funzioni su tutti i cellulari, soprattutto se si tratta di un videogioco. Ci sono quindi due problemi: uno è dato dalle semplici diversità tra cellulari come per esempio la risoluzione dello schermo e la diversa disposizione dei tasti; l'altro è dato invece dalle differenti librerie richieste da cellulari di marche diverse. Nel nostro caso, se vuole adattare il gioco per Nokia S40, cosa che l'azienda intende fare, occorre innanzitutto adattare la grafica. La grafica è stata pensata per lo schermo S60 che ha misura 176x208 pixel mentre quello per S40 misura 128x128 pixel, quindi dovrà essere completamente ridisegnata: o si fanno i pinguini più piccoli o se ne disegnano di meno. Molto probabilmente ogni programma dovrà essere rimodellato per ogni marca di cellulare, se non addirittura per ogni modello. Se non si vuole realizzare una versione apposta per ogni serie di cellulari se ne potrebbe fare una versione unica, che si informi direttamente dall'utente sul modello che possiede per poi adattarsi, però questa versione occuperà più spazio. Il problema delle librerie differenti è molto frequente nei videogiochi che fanno quasi sempre uso. La seconda versione di MIDP cerca di risolvere questo problema: tutte le funzioni utili per fare giochi sono già incluse nel profilo MIDP 2.0 e non sarà necessario importare altre librerie. Se avessimo sviluppato il nostro gioco in MIDP 2.0 non sarebbe stato necessario importare com.nokia.mid.ui.FullCanvas anzi: non avremmo nemmeno dovuto costruirci le classi TheCanvas e Sprite, infatti in MIDP 2.0 è possibile importare javax.microedition.lcdui.game.* che comprende Sprtite e Canvas (e altre utilissime funzioni per i giochi). Purtroppo però la gran parte dei cellulari attualmente usati non supporta MIDP 2. Chi sviluppa videogiochi J2ME deve dunque tener conto del tempo necessario per adattare il gioco a più modelli di cellulari possibili, accorgimento che ovviamente renderà il prodotto più vendibile. Visto che non è del tutto vero che un applicativo J2ME (MIDP 1.0) sia compatibile con

tutti i dispositivi, viene da chiedersi come mai vada così di moda usare java su questi telefonini così scarsi di risorse hardware che per lanciare un applicativo java sono costretti ad eseguire in parallelo anche la *virtual machine*. A questo punto non sarebbe meglio sviluppare questi applicativi in C o C++, facendone specifiche versioni per ogni serie di cellulari, che al momento si è costretti a fare anche con java, compilandole con appositi compilatori specifici per ogni dispositivo? In tal modo non sarebbe più necessario usare la *virtual machine*. Attualmente è possibile fare programmi per il sistema operativo *Symbian* usato su alcuni telefonini come i Nokia S60 ed i SonyEricsson p800 e p900: sono di questo tipo infatti alcuni giochi 3D molto complessi, e questa sembrerebbe la soluzione migliore. Ma attualmente il mercato e gli operatori non sono ancora pronti a distribuire contenuti *Symbian*.
Dunque è veramente agevole sviluppare in java applicativi di questo tipo, sia per via delle potenzialità offerte da questo linguaggio di programmazione, che si presta molto bene allo scopo, sia per il materiale fornito da parte dei produttori di cellulari. Invece è veramente fastidioso dover adattare il programma a molti dispositivi: risulta una problematica di non poca importanza, e durante lo sviluppo non bisogna sottovalutare i tempi richiesti per risolvere questi problemi.