UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Joc „Blobs” pentru platforma Android

Proiect practică

Student
Dragoș Popa

Iași, 2014

Cuprins

Capitolul 1. Introducere

Tema proiectului este realizarea unui joc bazat pe mișcări generate de senzori. Jocul își propune ca utilizatorul să controleze o bilă cu scopul realizării unui punctaj cât mai mare. Bila va fi controlată prin intermediul valorilor generate de accelerometru ca urmare a mișcării device-ului, dar se vor putea folosi și senzorii tactili ai ecranului pentru mișcarea acesteia (jocul a fost conceput, în principal, pentru folosirea accelerometrului, astfel încât se recomandă evitarea părții tactile). Utilizatorul va trebuie să se ferească sau să adune alte bile în vederea acumulării unui scor cât mai mare.

Am ales această temă întrucât dezvoltarea jocurilor este un domeniu ce mă motivează. De asemenea, am ales sa utilizez senzorii pentru a oferi o experiență cât mai interactivă utilizatorului, dar și pentru că îi consider ca fiind o facilitate care nu trebuie neglijată pentru că poate fi foarte utilă. Cu această ocazie am urmărit să dobândesc și cunoștințe legate de lucrul cu senzori.

Aplicația a fost concepută pentru a rula pe device-uri ce suportă minim nivelul 10 de Android SDK (Android 2.3.3) și maxim nivelul 19 SDK (Android 4.4.2). A fost testată pe un device cu sistem de operare Android 2.3.6 (Samsung Galaxy S5660). Device-urile pe care va fi rulat proiectul trebuie să aibă integrat un accelerometru, însă, în general, aproximativ toate dispozitivele care ruleaza minim Android 2.3.3 au și acest senzor integrat. De notat însă că device-ul va trebui sa aibă și un card SD sau un mediu de stocare externă, deoarece aplicația salvează numele și scorurile utilizatorilor în memoria externă.

Capitolul 2. Fundamentarea teoretica si documentarea bibliografica pentru tema propusa

În proiectul realizat am folosit următoarele tehnologii: Java, Android, Android SDK, XML, JSON.

Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor ’90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise in Java, iar noile evoluții tehnologice permit utilizarea sa si pe dispozitive mobile gen telefon, agenda electronica, palmtop etc. In felul acesta se creeaza o platforma unica, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat in prezent cu succes si pentru programarea aplicatiilor destinate intranet-urilor

Android este un sistem de operare mobil, bazat pe kernel-ul de Linux, și este dezvoltat în prezent de Google. Cu o interfață bazată pe manipularea directă de către utilizator, Android este devoltat în principiu pentru dispozitive mobile cu touchscreen și interfețe specifice pentru televizoare, automobile și ceasuri. Sistemul de operare folosește mișcări tactile care corespund celei din lumea reală(swiping, tapping, pinching) și care ajută la manipularea obiectelor de pe ecran și a tastaturii virtuale.

Android SDK este un pachet de dezvoltare software, care include un debugger, biblioteci, un emulator, documentație, tutoriale. Principalul mediu de dezoltare integrat este Eclipse care foloseste extensia Android Development Tools(ADT).

Extensible Markup Language (XML) este un meta-limbaj de marcare recomandat de Consorțiul Web pentru crearea de alte limbaje de marcare, cum ar fi XHTML, RDF, RSS, MathML, SVG, OWL etc. Aceste limbaje formează familia de limbaje XML. XML este acum și un model de stocare a datelor nestructurate și semi-structurate în cadrul bazelor de date native XML. Datele XML pot fi utilizate în limbajul HTML, permit o identificare rapidă a documentelor cu ajutorul motoarelor de căutare. Cu ajutorul codurilor javascript, php etc. fișierele XML pot fi înglobate în paginile de internet, cel mai elocvent exemplu este sitemul RSS care folosește un fișier XML pentru a transporta informațiile dintr-o pagină web către mai multe pagini web.

JSON este un acronim în limba engleză pentru JavaScript Object Notation, și este un format de reprezentare și interschimb de date între aplicații informatice. Este un format text, inteligibil pentru oameni, utilizat pentru reprezentarea obiectelor și a altor structuri de date și este folosit în special pentru a transmite date structurate prin rețea, procesul purtând numele de serializare. JSON este alternativa mai simplă, mai facilă decât limbajul XML. Eleganța formatului JSON provine din faptul că este un subset al limbajului JavaScript (ECMA-262 3rd Edition) , fiind utilizat alături de acest limbaj. Formatul JSON a fost creat de Douglas Crockford și standardizat prin RFC 4627. Tipul de media pe care trebuie să îl transmită un document JSON este application/json. Extensia fișierelor JSON este .json.

Capitolul 3. Proiectarea aplicației

3.1. Descriere generală a arhitecturii

Aplicația are la bază clasa GameActivity, căreia îi este asignată o altă clasă ce extinde SurfaceView(GameSurface), pe care se vor petrece animațiile. Calculele necesare afișării cadrelor vor fi efectuate pe un fir secundar(MyThread) care va permite să se deseneze un nou cadru la o frecvență de 60fps. Un alt fir paralel cu acesta(care va fi apelat printr-un obiect din clasa Timer) va genera obiecte la intervale variabile de timp.

3.2. Descrierea interfeței cu utilizatorul

Aplicația (Jocul) are numele „Blobs” și după instalarea pe device va fi lansată prin apasărea inconiței sugestive, ce va declanșa activitatea inițială, adică un *splash screen* / o

Figura 1: Splash Screen

imagine ce va apărea pe tot ecranul pentru 2 secunde.

După terminarea activității inițiale, apare o nouă activitate ce va reprezenta meniul

Figura 2: Meniu

principal al aplicației, format din 3 butoane: *Start, Leaderboards și Exit*.

La apăsarea butonului *Start* va fi lansată activitatea care va inițializa un nou joc. Bila roșie din centrul ecranului este cea controlată de utilizator. Utilizatorul va putea controla bila sa prin înclinarea telefonului în direcția dorită. Pe masură ce trece timpul pe ecran apar alte bile mai mici aflate în mișcare pe o anumită direcție.

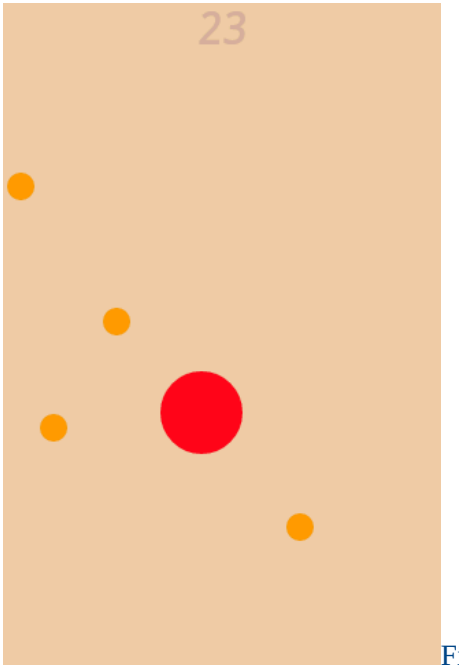

Figura 3: Joc

În partea de sus a ecranului se poate vedea scorul acumulat. Scorul crește atunci când

sunt generate noi bile sau când utilizatorul se ciocnește de bile roșii. Dacă se ciocnește de bile galbene, scorul va scădea în funcție de câte bile galbene a ciocnit până atunci și, în același timp, culoarea bilei sale se va deschide. Atunci când culoarea devine galben deschis un cerc roșu va apărea în jurul bilei atenționând că la următoarea bilă galbenă ciocnită, jocul se va termina. Însă dacă utilizatorul va ciocni bile roșii, culoare bilei sale va crește înapoi spre roșul inițial, îndepărtându-l de pericol.

Atunci când bila trece de ultima culoare galbenă, jocul se încheie, iar ca urmare apare o alertă cu scorul realizat și cu un câmp pentru introducerea numelui, care vor fi salvate după

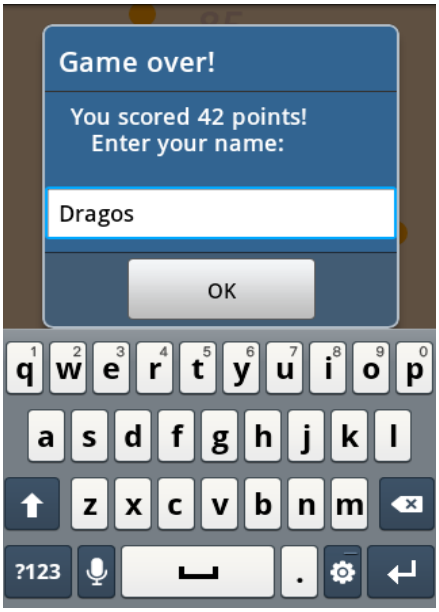

Figura 4: Alerta sfarsit joc

apăsarea butonului *OK*.

La apăsarea celui de-al doilea buton din meniul principal, *Leaderboards*, va apărea o nouă activitate ce va afișa o listă cu scorurile jucătorilor, sortate descrescător.

Figura 5: Tabelă scoruri

Desigur, ultimul buton din meniul principal, *Exit*, va ieși din aplicație.

Capitolul 4. Implementarea aplicației

Pentru implementarea aplicației am folosit mediul de dezvoltare Eclipse cu extensia ADT și limbajul Java. O alternativă ar fi fost și limbajul C++, sau Android Studio (Beta) ca mediu de programare.

Principala activitate a proiectului este GameActivity, a cărei view este clasa GameSurface ce extinde SurfaceView, o clasă care la rândul ei este derivată din View.

Aproximativ toată logica jocului este implementată în GameSurface, pe două fire secundare. Acesta este un avataj pe care ni-l oferă SurfaceView, astfel încât pe un fir secundar se efectuează calculele pentru fiecare pas de afișare/randare, iar apoi este transmis către firul principal cadrul final ce trebuie afișat pentru pasul respectiv. Această structură este inspirată din cartea *Appress Beginning Android Games(2011)[1]*.

În clasa MyThread este implementată logica firului secundar care se ocupă de aceste lucruri.

```
public void run() {
            Canvas c = null;

            while (isRunning) {
                  if(mySurfaceHolder.getSurface().isValid()) {

                        long timeStart = System.currentTimeMillis();

                        try {
                              c = mySurfaceHolder.lockCanvas(null);
                              synchronized(mySurfaceHolder) {

                                    mySurface.draw(c);
                                    mySurface.update();
                              }
                        } finally {
                              if (c!=null)
                                    mySurfaceHolder.unlockCanvasAndPost(c);
                        }

                        long timeEnd = System.currentTimeMillis();
                        long frameTime = timeEnd-timeStart;
                        long sleepTime = (int) ((1000/fps)-frameTime);

                        if (sleepTime>0){
                              try {
                                    Thread.sleep(sleepTime);
                              } catch (InterruptedException e) {}
                        }
                  }

            }
      }
```

Mai sus este prezetată implementarea metodei *run()* a clasei MyThread. Este necesar apelul funcției isValid() pentru a verifica dacă suprafața este disponibilă, întrucât ea există doar între apelurile onResume() și onPause() ale activității din care face parte. La apelul lockCanvas() este returnat canvas-ul SurfaceView-ului, care va fi modificat pe acest fir secundar, urmând ca la

apelul unlockCanvasAndPost() să fie afișat pe firul de UI canvas-ul rezultat. Un astfel de pas de randare este măsurat în milisecunde, pentru a calcula apoi cât timp mai trebuie așteptat până la un anumit interval prestabilit pentru acest proces (în acest caz 1000/fps, unde fps=60, pentru a putea fi afișate 60 de cadre pe secundă), acest lucru limitând numărul de pași de randare per interval de timp, pentru a nu se executa necontenit sau mai mult decât este nevoie.

O instanță a clasei MyThread este creată și pornită în metoda surfaceCreated(), care împreună surfaceDestroyed() constituie metode ale SurfaceHolder.Callback ce trebuiesc implementate pentru a manipula resursele și firele de execuție atunci când suprafața este creată (după onResume()) și când este distrusă (după onPause()).

```
public void surfaceCreated(SurfaceHolder holder) {
	// TODO Auto-generated method stub
	playerBlob = new Blob(100, 100,30);
	blobs = new ArrayList<Blob>();

	paintThread = new MyThread(this,mySurfaceHolder);
	paintThread.isRunning = true;
	paintThread.start();

	manager.registerListener(this, sensor,
			SensorManager.SENSOR_DELAY_FASTEST);

	t = new Timer();
	t.schedule(new TimerTask() {
		public void run() {
			timerJob();
		}
	}, 1500,genTime);
}

public void surfaceDestroyed(SurfaceHolder holder) {

	manager.unregisterListener(this);
	stopTasks();

	blobs.clear();
	blobs = null;
	score = 0;
	greenIntensity = 0;
}
```

După cum se poate observa în metoda surfaceCreated(), sunt instanțiate playerBlob (care este obiectul controlat de utilizator) și blobs, în care vor fi stocate toate obiectele de tipul Blob generate aleator în metoda timerJob(). TimerJob() este de asemenea apelată pe un fir secundar la un interval de timp indicat de variabila genTime, prin intermediul unui obiect t, de tip Timer.

O problemă întâmpinată aici a fost faptul că schimbarea variabilei genTime nu afecta automat intervalul la care timerul apela metoda respectivă, singura soluție fiind să se renunțe la timerul curent și să se apeleze schedule() pentru o altă instanță a timer-ului, cu noul interval de timp reprezentat de genTime.

```
public void timerJob() {
	generateBlob();
	score += new Random().nextInt(10);
	level++;
```

```
            switch (level) {

            case 40: genTime = 450; rescheduleTimer(); break;
            case 80: genTime = 250; rescheduleTimer(); break;
            case 120: genTime = 170; rescheduleTimer(); break;
            case 200: genTime = 140; rescheduleTimer(); break;
            default: break;
            }
      }

      public void rescheduleTimer() {

          Log.i("Gen_TIME", ""+genTime);
            t.cancel();
            t = new Timer();
            t.schedule(new TimerTask() {
                  public void run() {
                        timerJob();
                  }
            }, 0, genTime);
      }
```

În timp ce firul lansat de timer populează ArrayList-ul blobs, firul MyThread apelează metodele update() și draw() pentru randare. Metoda update() calculează noile poziții ale obiectelor din blobs și verfică eventualele coliziuni cu obiectul playerBlob, adică o coliziune între cercuri. Aceasta se calculează foarte simplu, făcând diferența dintre distanța centrelor cercurilor și suma razelor acestora. În cazul de față, cele două valori de scăzut sunt ridicate la pătrat pentru a nu mai fi nevoie de efectuarea radicalului pentru calcularea distanței.

```
      int dX = playerBlob.x - blob.x;
            int dY = playerBlob.y - blob.y;
            int distanceSquared = dX * dX + dY * dY;

            boolean collision = distanceSquared < (playerBlob.r + blob.r) *
(playerBlob.r + blob.r);
```

Dacă există coliziune (collision = true), se elimină elementul ciocnit din blobs și se efectuează alte schimbări în contextul jocului (culoarea bilei playerBlob, dată de valoarea variabilei greenIntensity, scorul, vibrații). Variabila greenIntensity poate lua valori între 5 și 245 și reprezintă intensitatea culorii verde, care în combinație cu roșu va tinde spre galben atunci când se va apropia de 255. Mai jos este secvența de cod care setează culoarea rezultată și afișează noua bilă.

```
      p.setColor(Color.rgb(255, greenIntensity, 30));
      canvas.drawCircle(playerBlob.x, playerBlob.y, playerBlob.r, p);
```

Deoarece coliziunile se verifică pe firul secundar și nu pe cel de UI, pentru a afișa o alertă de terminare a jocului pe ecran (ca urmare a unei coliziuni) , a fost nevoie de utilizarea metodei runOnUiThread() a activității de care aparține SurfaceView-ul.[2]

```
((Activity)context).runOnUiThread(new Runnable() {
```

```
                @Override
                public void run() {
                      // TODO Auto-generated method stub
                      stopTasks();
      .....

                      alertDialog = alert.create();
                      alertDialog.show();
                }
          });
```

Clasa GameSurface implementează și metoda onSensorChanged() a clasei SensorEventListener, care returnează schimbări ale acceleraţiei senzorului.

Sezorul de mișcare folosit este accelerometrul:

```
manager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
sensor = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Acesta începe ascultarea mișcărilor la apelul registerListener(), efectuat după crearea suprafeței la surfaceCreated(). Pentru a opri firul care înregistrează mișcările accelerometrului este necesar apelul metodei unregisterListener(), care în acest caz este efectuat în surfaceDestroyed().

Metoda onSensorChanged() este apelată atunci când survin modificări ale valorilor accelerometrului:

```
      public void onSensorChanged(SensorEvent event) {
            // TODO Auto-generated method stub

            final float alpha = 1.0f;
            float gravity[] = {0,0};
            float acceleration[] = new float[2];

            gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
            gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];

            acceleration[0] = event.values[0] - gravity[0];
            acceleration[1] = event.values[1] - gravity[1];


            int newX = playerBlob.x - (int)acceleration[0];
            int newY = playerBlob.y + (int)acceleration[1];

            if (newX-playerBlob.r >= 0 && newX+playerBlob.r <= getWidth() &&
                        newY-playerBlob.r >= 0  && newY+playerBlob.r <=
getHeight())
            {
                  playerBlob.x = newX;
                  playerBlob.y = newY;
            }

      }
```

Valorile event.values de indecși 0 și 1 reprezintă accelerația pe axele x și y ale senzorului. Pentru a folosi aceste valori se aplică anumite filtre în funcție de ce este dorit a fi observat în mișcarea senzorului. Filtrul de mai sus este unul din cele mai simple și este dat ca exemplu pe

site-ul *developer.android.com[3]*. Acesta este un filtru trece-sus, însă aici am setat alpha ca fiind 1, pentru a obține rezultatele dorite, deci practic valorile nu mai sunt filtrate. În mod normal accelerația gravitațională este eliminată și ea, însă în acest caz este considerată 0 pentru axele x și y, și ~9,8 m/s^2 pentru z.

Valorile obținute sunt adăugate la coordonatele curente ale obiectului playerBlob pentru a-l deplasa în funcție de mișcarea accelerometrului.

La apăsarea butonului OK din alert view-ul de la sfarșitul jocului, se apelează metoda registerScore(), aparținând activității curente. Aici se salvează scorul realizat, precum și numele jucătorului, într-un fișier JSON aflat la root-ul Sdcard-ului. Astfel se poate vizualiza un clasament la apăsarea butonului *Leaderboards,* care încarcă scorurile înregistrate în fișierul JSON.

Capitolul 5. Concluzii

Ca orice aplicație de acest gen, jocul realizat poate fi îmbunătățit și îmbogățit cu diferite elemente. Un joc trebuie să fie cât mai captivant și cât mai puțin plictisitor pentru utilizator, trebuie să arate bine și să atragă cât mai mult jucătorii. Un element care ar ajuta aplicația în acest sens ar fi adăugarea de puteri pentru bilă. De exemplu, după un anumit număr de bile roșii atinse, bila utilizatorului sa poată emite o rază care să distrugă bilele din calea sa. Ar putea fi adăugate și alte tipuri de bile în afară de roșii și galbene, care să aibă diferite funcții.

Partea de generare a bilelor și a traictoriei acestora ar putea fi îmbunătățită pentru a oferi o experiență mai echilibrată. De asemenea, scorurile ar putea fi salvate și încărcate de pe un server și cu ajutorul unei baze de date.

Jocul oferă o experiență interesantă datorită senzorului, care poate testa îndemânarea utilizatorilor și poate compara abilitațile acestora prin intermediul scorului.

Bibliografie

[1] Mario Zechner, „*Beginning Android Games*”, Apress, 2011.

[2] Google, „API Guides” [Online], Disponibil la adresa: http://developer.android.com/guide/index.html, Accesat: 2014.

[3] Google, „Motion Sensors” [Online], Disponibil la adresa: http://developer.android.com/guide/topics/sensors/sensors_motion.html, Accesat: 2010.