MOBILE GAMING

A final project submitted to the University of Wales in partial fulfilment of the requirements of BA (Hons) Digital Media in Game Development

June 2013

The Frog Hunt

Pavankumar Chopra

2nd Year UG – Game Development

11UG03040

ICAT, Design and Media College,

Bangalore

Table of Contents

THE FROG HUNT

OVERVIEW

The Frog Hunt is Zuma Style game with a muddy-sandy ground theme. It is basically a never ending marble shoot out game. It is simple and addictive. Your target is to eliminate as many as marbles before they reach the frog and destroy it. The marbles spawn from the skull structures located in each corner of the screen.

MARKETING

Target audience

Age: 5-plus

Gender: Both

Geographic target: World-Wide

Platform- J2ME

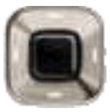
GAME PLAY

The player plays the role of a frog who is sitting in a muddy playground. The frog can't move around. His position is fixed. However, he can rotate along four angles – 0, 90, 180 and 360 degrees. Marbles of different colours spawn from skull shaped structures located in four corners of the screen. These marbles move randomly in all direction. Frog loses his health if hit by marble. Shooting red and blue marbles gives score bonus, whereas green marbles gives health bonus. The player has to shoot as many as marbles possible before the frog's health is out.

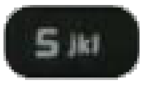
GAME RULES

1. Shoot the marbles.
2. Avoid getting hit by marbles.
3. Red and blue marbles gives score bonus, whereas green marbles gives health bonus.

GAME CONTROLS



Rotate Around



Shoot

GAME TITLE

The Frog Hunt

TECHNICAL SPECIFICATIONS

NetBeans IDE 7.3 Beta 2

J2ME

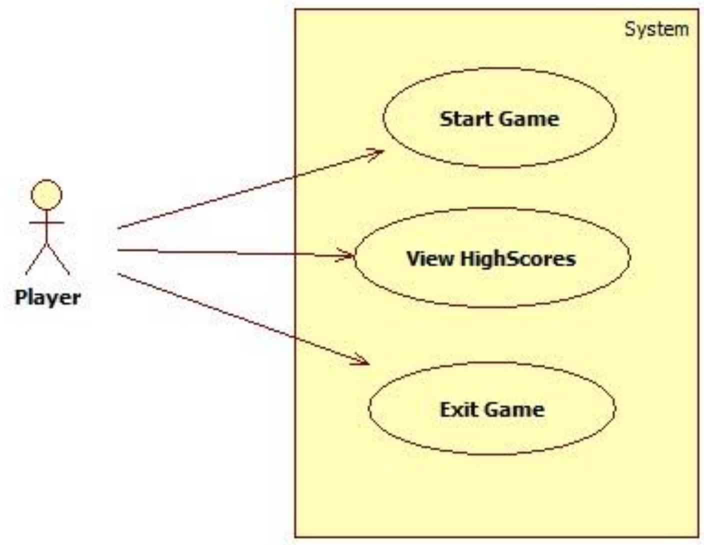
Photoshop

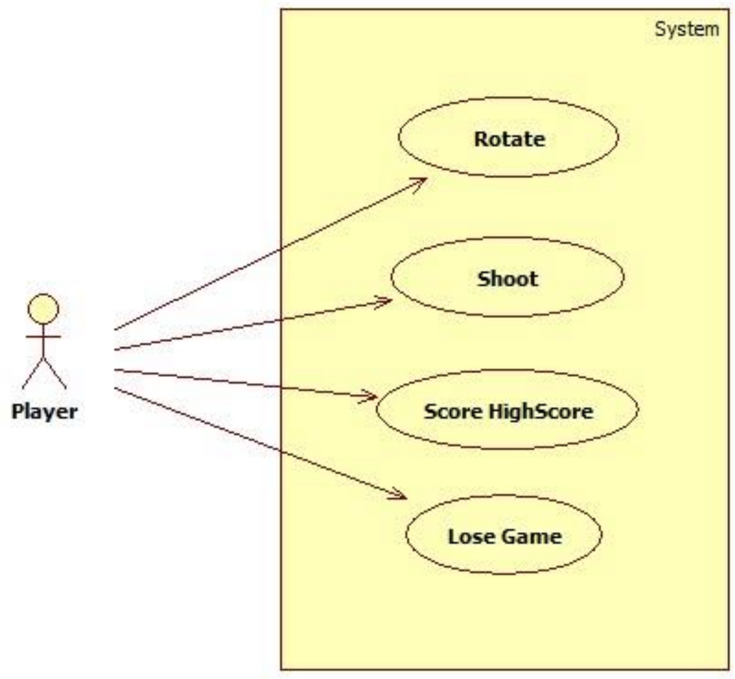
UML DIAGRAMS

USE CASE DIAGRAM

Description: The below diagram represents the actions, user can performed in the game. This diagram helps us to know all the actions user can perform in the game.

Diagram:

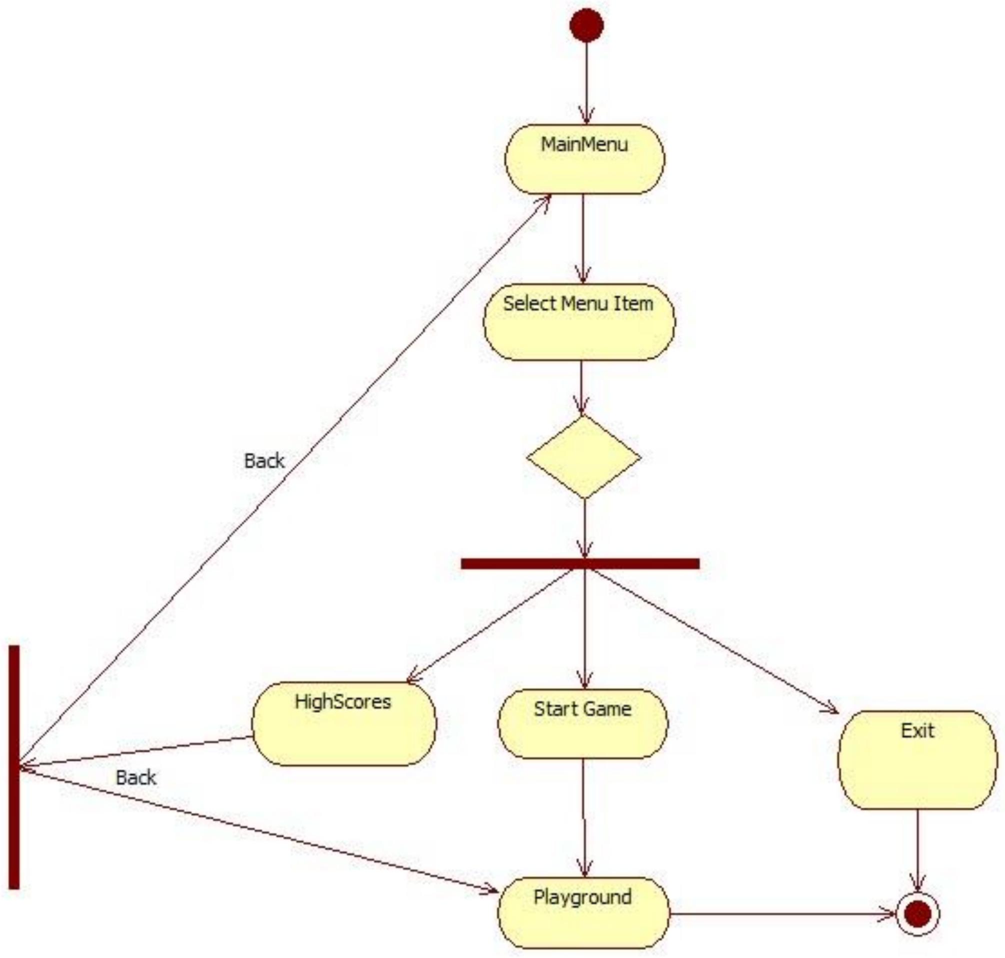


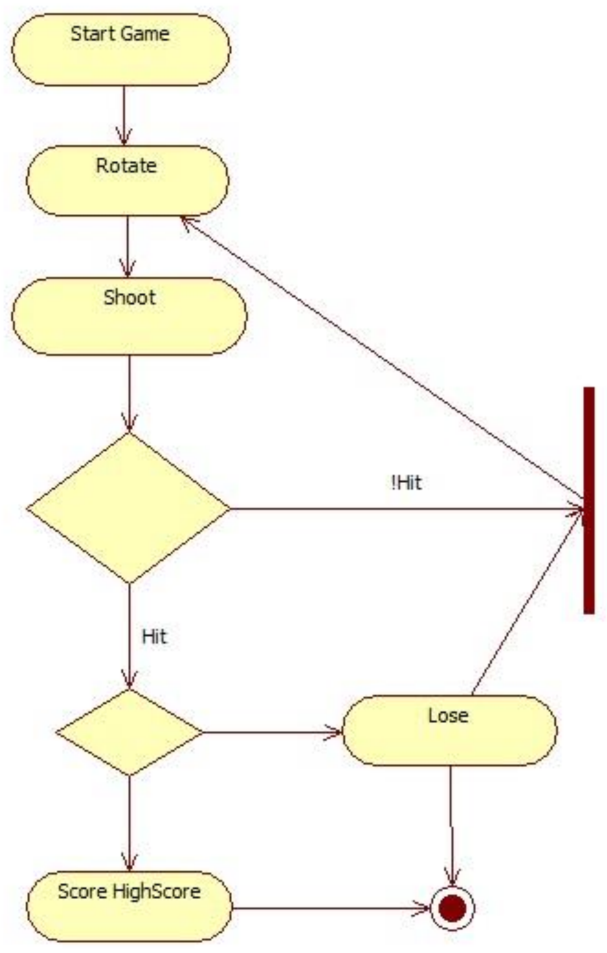


ACTIVITY DIAGRAM

Description: Activity Diagram is used to show the game-play mechanics working parallel.

Diagram:

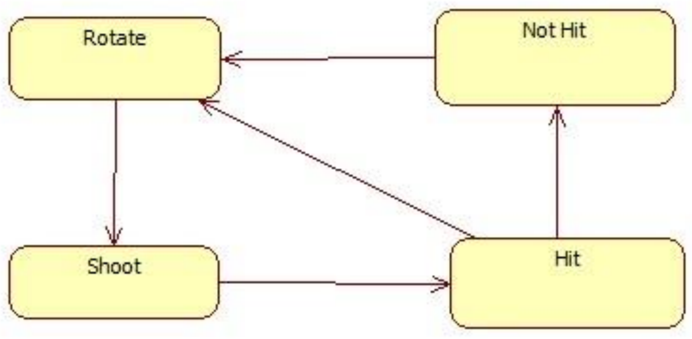




STATE DIAGRAM

Description: The diagram represents States of the Player. Activity Diagram helps us to know what activities can be performed by a Player or an AI.

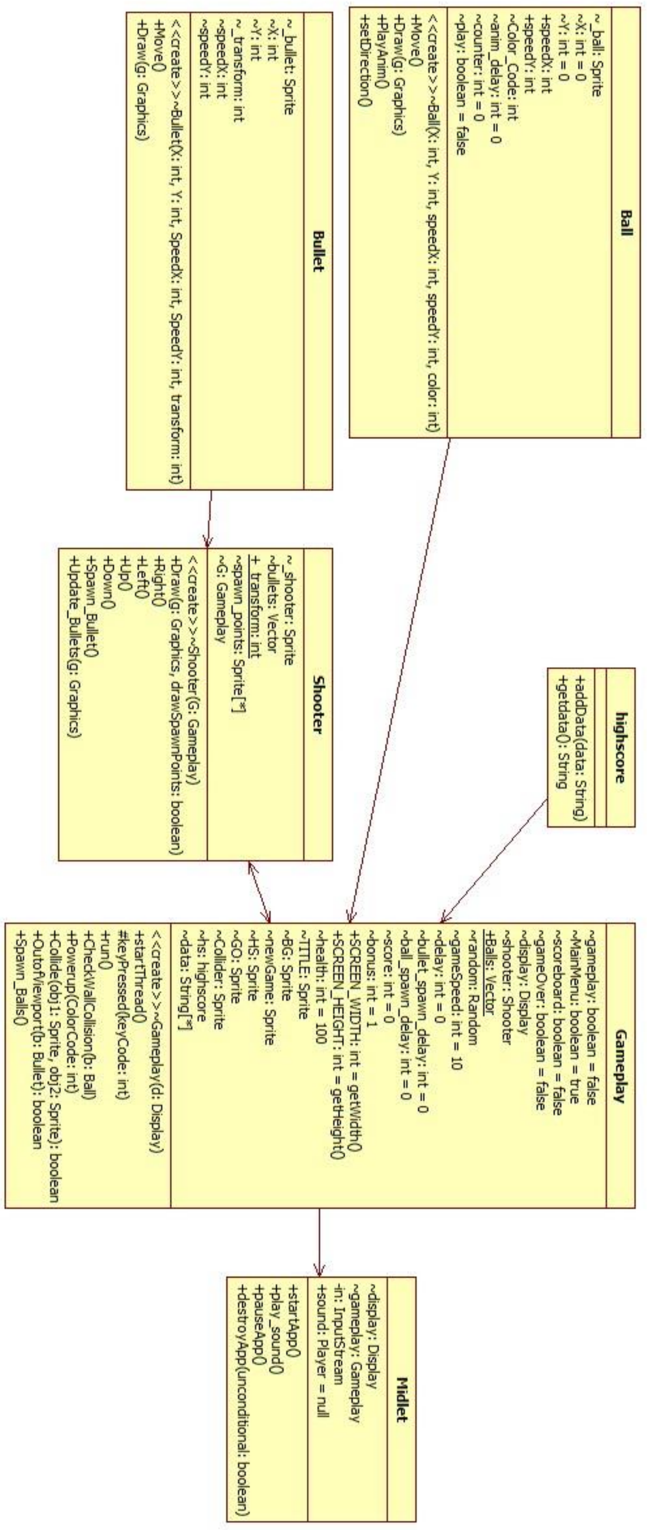
Diagram:



CLASS DIAGRAM

Description: The below diagram represents the class diagram. Then some important variables in the class and then functions declared in the class.

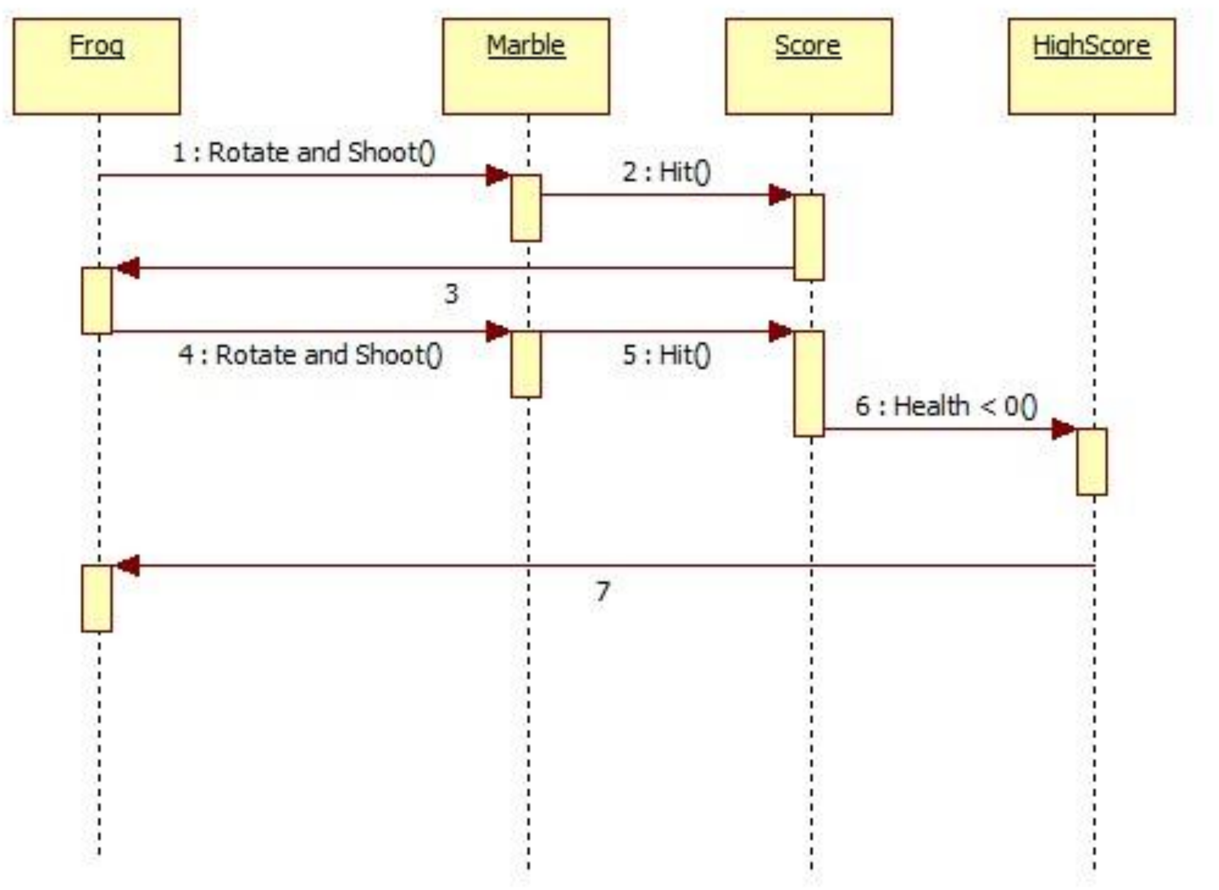
Diagram:



SEQUENCE DIAGRAM

Description: Sequence Diagram helps user to know the interaction between Player and the varies Puzzles in the game.

Diagram:



SOURCE CODE

Ball.java

```
package java_game;
import java.io.IOException;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class Ball
{
    Sprite _ball;
    int X = 0;
    int Y = 0;
    public int speedX;
    public int speedY;
    int Color_Code;   // Red(Points) - 0   Green(Health) - 1  Blue(Points) - 2
//Gray (Normal Points)
    int anim_delay = 0;
    int counter = 0;
    boolean play = false;

    Ball(int X, int Y, int speedX, int speedY, int color) throws IOException
    {
        this.Color_Code = color;
        switch(this.Color_Code)
        {
            case 0:
                _ball = new Sprite(Image.createImage("java_game/red.png"), 20,
20);
                break;
            case 1:
                _ball = new Sprite(Image.createImage("java_game/green.png"), 20,
20);
                break;
            case 2:
                _ball = new Sprite(Image.createImage("java_game/blue.png"), 20,
20);
                break;
            case 3:
                _ball = new Sprite(Image.createImage("java_game/gray.png"), 20,
20);
                break;
        }
        this.X = X;
        this.Y = Y;
        this.speedX = speedX;
        this.speedY = speedY;
    }

   //Move the Ball
    public void Move()
    {
        X += speedX;
        Y += speedY;
        _ball.setPosition(X, Y);
    }

    public void Draw(Graphics g)
    {
        anim_delay++;
        _ball.paint(g);
    }

//Play Destroy Animation
    public void PlayAnim()
    {
        if(anim_delay > 30 && play)
        {
                anim_delay = 0;
                counter++;
                this._ball.nextFrame();

                if (counter == 2) {
                    counter = 0;
                    Gameplay.Balls.removeElement(this);
                    play = false;
                }
        }
    }

//Set the direction of animation after destroying
    public void setDirection()
    {
        switch(Shooter._transform)
        {
            case Sprite.TRANS_NONE:
                this.speedX = 0;
                this.speedY = -1;
                break;

            case Sprite.TRANS_ROT90:
                this.speedX = 1;
                this.speedY = 0;
                break;

                 case Sprite.TRANS_ROT180:
                this.speedX = 0;
                this.speedY = 1;
                break;

                     case Sprite.TRANS_ROT270:
                this.speedX = -1;
                this.speedY = 0;
                break;

            default:
                break;
        }
    }
}
```

Bullet.java

```
package java_game;
import java.io.IOException;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;

public class Bullet {
    Sprite _bullet;
    int X;
    int Y;
    int _transform;

    int speedX;
    int speedY;

    Bullet(int X, int Y, int SpeedX, int SpeedY, int transform) throws IOException
    {
        _bullet = new Sprite(Image.createImage("java_game/bullet.png"));
         this._transform = transform;
         _bullet.setTransform(this._transform);
        this.X = X;
        this.Y = Y;
        _bullet.setPosition(this.X, this.Y);
        this.speedX = SpeedX;
        this.speedY = SpeedY;
    }

//Move the Bullet
    public void Move()
    {
        X += speedX;
        Y += speedY;
        _bullet.setPosition(X, Y);
    }
    public void Draw(Graphics g)
    {
        _bullet.paint(g);
    }
}
```

Gameplay.java

```
package java_game;
import java.io.IOException;
import java.util.Random;
import java.util.Vector;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Font;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.GameCanvas;
import javax.microedition.lcdui.game.Sprite;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
```

```
public class Gameplay extends GameCanvas implements Runnable {

    boolean gameplay = false;
    boolean MainMenu = true;
    boolean scoreboard = false;
    boolean gameOver = false;

    Display display;
    Shooter shooter;

    public static Vector Balls;
    Random random;

    int gameSpeed = 10;    //Ball Movement Speed
    int delay = 0;         //balls delay variable
    int bullet_spawn_delay = 0;   //Ball Spawn Delay
    int ball_spawn_delay = 0;
    int score = 0;
    int bonus = 1;
    public int SCREEN_WIDTH = getWidth();
    public int SCREEN_HEIGHT = getHeight();
    int health = 100;

    Sprite TITLE;
    Sprite BG;
    Sprite newGame;
    Sprite HS;
    Sprite GO;
    Sprite Collider;

    highscore hs;
    String[] data;


    Gameplay(Display d) throws IOException {
        super(false);
        this.display = d;
        shooter = new Shooter(this);

        Balls = new Vector();
        random = new Random();

        BG = new Sprite(Image.createImage("java_game/BG.png"));

        TITLE = new Sprite(Image.createImage("java_game/TITLE.png"));
        TITLE.setPosition(45, 20);

        newGame = new Sprite(Image.createImage("java_game/newGame.png"));
        newGame.setPosition(70, 80);

        HS = new Sprite(Image.createImage("java_game/hs.png"));
        GO = new Sprite(Image.createImage("java_game/go.png"));

        Collider = new Sprite(Image.createImage("java_game/collider.png"));
        Collider.setPosition(shooter._shooter.getX(), shooter._shooter.getY());
//-13 offset

        hs = new highscore();
    }
    public void startThread() {
        Thread t = new Thread(this);
        t.start();
    }

    protected void keyPressed(int keyCode) {
        super.keyPressed(keyCode);

        if (keyCode == -1) {
            shooter.Up();
        }
        if (keyCode == -2) {
            shooter.Down();
        }
        if (keyCode == -4) {
            shooter.Right();
        }
        if (keyCode == -3) {
            shooter.Left();
        }
        if (bullet_spawn_delay > 500) {
            if ((keyCode == 103 || keyCode == KEY_NUM5) && (gameplay || MainMenu))
{
                bullet_spawn_delay = 0;
                try {
                    shooter.Spawn_Bullet();
                } catch (IOException ex) {
                }
            }
        }

        if (scoreboard && keyCode == -7) {
            scoreboard = false;
            MainMenu = true;
        }

        if (gameOver && keyCode == -7) {
            try {
                hs.addData(score + "");
                System.out.println(score);
            } catch (RecordStoreFullException ex) {
            } catch (RecordStoreNotFoundException ex) {
            } catch (RecordStoreException ex) {
            }
            gameOver = false;
            MainMenu = true;
            health = 100;
            score = 0;
        }
    }

    public void run() {
        Graphics g = getGraphics();
        while (true) {
        g.setFont(Font.getFont(Font.FACE_MONOSPACE,Font.STYLE_PLAIN
,Font.SIZE_SMALL));
        g.setColor(255,180,0);
```

```
            delay++;
            bullet_spawn_delay++;

            if (delay >= gameSpeed) {
                delay = 0;
                BG.paint(g);

                if (gameplay) {
                    g.drawString("Score: " + score, 0, 0, 0);
                    g.drawString("Health: " + health, 160, 0, 0);
                }
                if (MainMenu) {
                    TITLE.paint(g);
                    newGame.paint(g);
                    HS.setPosition(70, 200);
                    HS.paint(g);
                    g.drawString("Use Arrow-keys to rotate the frog", SCREEN_WIDTH
/ 2 - 115, SCREEN_HEIGHT - 40, 0);
                    g.drawString("Press 5 to Shoot", SCREEN_WIDTH / 2 - 55,
SCREEN_HEIGHT - 20, 0);
                }
                if (gameOver) {
                    GO.setPosition(65, 100);
                    GO.paint(g);
                    g.drawString("Score: " + score, SCREEN_WIDTH / 2 - 35,
SCREEN_HEIGHT / 2, 0);
                    g.drawString("Press Back to Return to Main Menu!", 2,
SCREEN_HEIGHT / 2 + 20, 0);
                }

                if (MainMenu || gameplay) {
                    shooter.Draw(g, gameplay);
                    shooter.Update_Bullets(g);
                    Collider.setPosition(shooter._shooter.getX(),
shooter._shooter.getY());
                    //Collider.paint(g);
                }

                if (scoreboard) {
                    HS.setPosition(75, 50);
                    HS.paint(g);
                    if (data != null) {
                        for (int i = 0, y = 100; i < data.length; i++, y += 20) {
                            g.drawString("Player :" + data[i], 50, y, 0);
                            System.out.println("Player :" + data[i] + " Length:" +
i + " Data Length: " + data.length);
                        }
                    }
                }

                for (int i = 0; i < Balls.size(); i++) {
                    Ball b = (Ball) Balls.elementAt(i);

                    if (b._ball.collidesWith(Collider, false)) {
                        health = health - 10;
                        Balls.removeElement(b);
                        break;
                    }
                    b.Move();
                    b.PlayAnim();
                    b.Draw(g);
                    if (!(b.play)) {
                        CheckWallCollision(b);
                    }
                }
            }

            if (gameplay) {
                ball_spawn_delay++;
                if (health <= 0) {
                    gameplay = false;
                    MainMenu = false;
                    gameOver = true;
                    Balls.removeAllElements();
                    shooter.bullets.removeAllElements();

                }
                if (ball_spawn_delay > 2000 && !(Balls.size() > 20)) {
                    ball_spawn_delay = 0;
                    try {
                        Spawn_Balls();
                    } catch (IOException ex) {
                    }
                }
            }

            for (int i = 0; i < shooter.bullets.size(); i++) {
                Bullet bu = (Bullet) shooter.bullets.elementAt(i);

                //New Game
                if (MainMenu && Collide(bu._bullet, newGame)) {
                    gameplay = true;
                    MainMenu = false;
                    shooter.bullets.removeElementAt(i);
                }

                //HighScore
                if (MainMenu && Collide(bu._bullet, HS)) {
                    MainMenu = false;
                    scoreboard = true;
                    data = hs.getdata();
                    shooter.bullets.removeElementAt(i);
                }

                for (int j = 0; j < Balls.size(); j++) {
                    Ball b = (Ball) Balls.elementAt(j);

                    if (Collide(b._ball, bu._bullet)) {
                        score++;
                        b.setDirection();
                        Powerup(b.Color_Code);
                        b.play = true;
                        shooter.bullets.removeElementAt(i);
                        break;
                    }
                }
            }
```

```
            for (int i = 0; i < shooter.bullets.size(); i++) {
                Bullet b = (Bullet) shooter.bullets.elementAt(i);
                if (OutofViewport(b)) {
                    shooter.bullets.removeElementAt(i);
                    break;
                }
            }
            flushGraphics(0, 0, getWidth(), getHeight());
        }
    }

    public void CheckWallCollision(Ball b) {
        if (b._ball.getX() + b._ball.getWidth() > this.getWidth()) {
            b.speedX *= -1;
            b._ball.setPosition(this.getWidth() - b._ball.getWidth(),
b._ball.getY());
        }

        if (b._ball.getX() < 0) {
            b.speedX *= -1;
            b._ball.setPosition(0, b._ball.getY());
        }

        if (b._ball.getY() + b._ball.getHeight() > this.getHeight()) {
            b.speedY *= -1;
            b._ball.setPosition(b._ball.getX(), this.getHeight() -
b._ball.getHeight());
        }

        if (b._ball.getY() < 30) {
            b.speedY *= -1;
            b._ball.setPosition(b._ball.getX(), 0);
        }
    }

    public void Powerup(int ColorCode) {
        switch (ColorCode) {
            case 0:
                score += bonus;
                break;

            case 1:
                if (health <= 95) {
                    health += 5;
                }
                break;

            case 2:
                score += bonus;
                break;
        }
    }

    public boolean Collide(Sprite obj1, Sprite obj2) {
        if (obj1.collidesWith(obj2, true)) {
            return true;
        } else {
            return false;
        }
    }

    public boolean OutofViewport(Bullet b) {
        if (b._bullet.getX() > this.getWidth()) {
            return true;
        }

        if (b._bullet.getX() < 0) {
            return true;
        }

        if (b._bullet.getY() > this.getHeight()) {
            return true;
        }

        if (b._bullet.getY() < 30) {
            return true;
        }
        return false;
    }

    public void Spawn_Balls() throws IOException {
        switch (random.nextInt(4)) {
            case 0:
                Balls.addElement(new Ball(0, 30, 1, 1, random.nextInt(4)));
                break;

            case 1:
                Balls.addElement(new Ball(SCREEN_WIDTH - 20, 30, -1, 1,
random.nextInt(4)));
                break;

            case 2:
                Balls.addElement(new Ball(0, SCREEN_HEIGHT - 20, 1, -1,
random.nextInt(4)));
                break;

            case 3:
                Balls.addElement(new Ball(SCREEN_WIDTH - 20, SCREEN_HEIGHT - 20, -
1, -1, random.nextInt(4)));
                break;
        }
    }
}
```

Shooter.java

```
package java_game;
import java.io.IOException;
import java.util.Vector;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.game.Sprite;
```

```
public class Shooter {

    Sprite _shooter;
    Vector bullets;
    public static int _transform;
    Sprite spawn_points[];
    Gameplay G;
    //Sprite Collider;

    Shooter(Gameplay G) throws IOException {
        this.G = G;

        _shooter = new Sprite(Image.createImage("java_game/shooter.png"));
        if(G.MainMenu)
        {
            _shooter.setPosition(G.SCREEN_WIDTH / 2 - 13, G.SCREEN_HEIGHT / 2 -
13);
        }
        else
        {
            _shooter.setPosition(G.SCREEN_WIDTH / 2 - 13, G.SCREEN_HEIGHT / 2 +
75);
        }

        spawn_points = new Sprite[4];

        for (int i = 0; i < 4; i++) {
            spawn_points[i] = new
Sprite(Image.createImage("java_game/spawn_point.png"));
        }
        spawn_points[0].setPosition(0, 30);
        spawn_points[1].setPosition(G.SCREEN_WIDTH, 30);
        spawn_points[1].setTransform(Sprite.TRANS_ROT90);
        spawn_points[2].setPosition(0, G.SCREEN_HEIGHT);
        spawn_points[2].setTransform(Sprite.TRANS_ROT270);
        spawn_points[3].setPosition(G.SCREEN_WIDTH, G.SCREEN_HEIGHT);
        spawn_points[3].setTransform(Sprite.TRANS_ROT180);
    }

    public void Draw(Graphics g, boolean drawSpawnPoints) {
        //_shooter.setPosition(G.SCREEN_WIDTH / 2 - 13, G.SCREEN_HEIGHT / 2 + 75);

        if(G.MainMenu)
        {
            _shooter.setPosition(G.SCREEN_WIDTH / 2 - 13, G.SCREEN_HEIGHT / 2 -
13);
        }
        else
        {
            _shooter.setPosition(G.SCREEN_WIDTH / 2 - 13, G.SCREEN_HEIGHT / 2 +
75);
        }

        _shooter.paint(g);
        if (drawSpawnPoints)
        {
            for (int i = 0; i < 4; i++) {
                spawn_points[i].paint(g);
            }
        }
    }

    public void Right() {
        _transform = Sprite.TRANS_ROT90;
        _shooter.setTransform(_transform);
    }

    public void Left() {
        _transform = Sprite.TRANS_ROT270;
        _shooter.setTransform(_transform);
    }

    public void Up() {
        _transform = Sprite.TRANS_NONE;
        _shooter.setTransform(_transform);
    }

    public void Down() {
        _transform = Sprite.TRANS_ROT180;
        _shooter.setTransform(_transform);
    }

    public void Spawn_Bullet() throws IOException {
        switch (_transform) {
            case Sprite.TRANS_NONE:
                if(G.MainMenu)
                {
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 5,
G.SCREEN_HEIGHT / 2 - 25, 0, -1, _transform));
                }
                else
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 5,
G.SCREEN_HEIGHT / 2 + 57, 0, -1, _transform));
                break;

            case Sprite.TRANS_ROT90:
                if(G.MainMenu)
                {
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 + 12,
G.SCREEN_HEIGHT / 2 - 5, 1, 0, _transform));
                }
                if(G.gameplay)
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 + 12,
G.SCREEN_HEIGHT / 2 + 83, 1, 0, _transform));
                break;

            case Sprite.TRANS_ROT180:
                if(G.MainMenu)
                {
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 5,
G.SCREEN_HEIGHT / 2 + 15, 0, 1, _transform));
                }
                if(G.gameplay)
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 5,
G.SCREEN_HEIGHT / 2 + 103, 0, 1, _transform));
                break;

            case Sprite.TRANS_ROT270:
                if(G.MainMenu)
                {
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 30, 
G.SCREEN_HEIGHT / 2 - 5, -1, 0, _transform));
                }
                if(G.gameplay)
                    bullets.addElement(new Bullet(G.SCREEN_WIDTH / 2 - 30, 
G.SCREEN_HEIGHT / 2 + 83, -1, 0, _transform));
                break;
        }
    }

    public void Update_Bullets(Graphics g) {
        for (int i = 0; i < bullets.size(); i++) {
            Bullet b = (Bullet) bullets.elementAt(i);
            b.Move();
            b.Draw(g);
        }
    }
}
```

HighScore.java

```
package java_game;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;

public class highscore {

    public void addData(String data) throws RecordStoreFullException, 
RecordStoreNotFoundException, RecordStoreException {
        RecordStore recordstore = RecordStore.openRecordStore("Game1", true);

        byte[] bytedata = data.getBytes();

        recordstore.addRecord(bytedata, 0, bytedata.length);

        recordstore.closeRecordStore();
    }

    public String[] getdata() {
        String[] records = null;
        try {

            RecordStore recordstore;
            recordstore = RecordStore.openRecordStore("Game1", false);

            if (recordstore.getNumRecords() != 0) {
                records = new String[recordstore.getNumRecords()];
                byte[] byteinputdata = new byte[1];
                int length = 0;
                for (int x = 1; x <= recordstore.getNumRecords(); x++) {

                    if (recordstore.getRecordSize(x) > byteinputdata.length) {
                        byteinputdata = new byte[recordstore.getRecordSize(x)];
                    }

                    recordstore.getRecord(x, byteinputdata, 0);
                    records[x - 1] = new String(byteinputdata, 0, 
byteinputdata.length);
                }
            }
            recordstore.closeRecordStore();

        } catch (RecordStoreException ex) {
            System.out.println("i am happy .. ur still throwing ");
        }
        return records;
    }
}
```

MIDlet.java

```
package java_game;
import java.io.IOException;
import java.io.InputStream;
import javax.microedition.lcdui.Display;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
import javax.microedition.media.Player;
import javax.microedition.midlet.*;

public class Midlet extends MIDlet {
    Display display;
    Gameplay gameplay;


     private InputStream in;
     public Player sound = null;

    public void startApp() {
        display = Display.getDisplay(this);
        try {
            gameplay = new Gameplay(display);
            gameplay.startThread();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
         play_sound();
        display.setCurrent(gameplay);
    }

     public void play_sound() {
             // TODO Auto-generated method stub
              in = getClass().getResourceAsStream("/sound/music.mid");
         try {
                   sound = Manager.createPlayer(in, "audio/midi");
```

```
                  sound.setLoopCount(-1);
                  //sound.prefetch();
                  sound.start();

            } catch (IOException e) {
                  e.printStackTrace();
            } catch (MediaException e) {
                  e.printStackTrace();
            }
      }

      public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

}
```

SCREENSHOTS

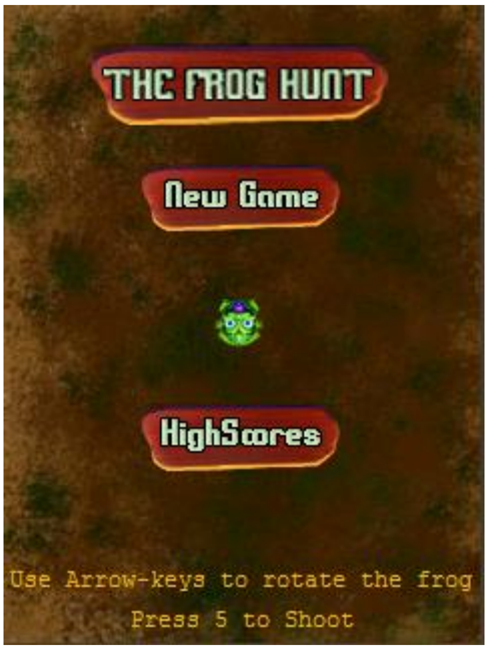


Figure 1

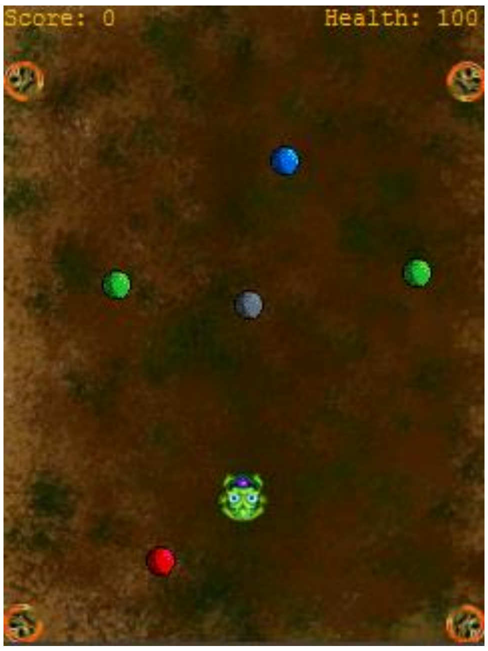


Figure 2

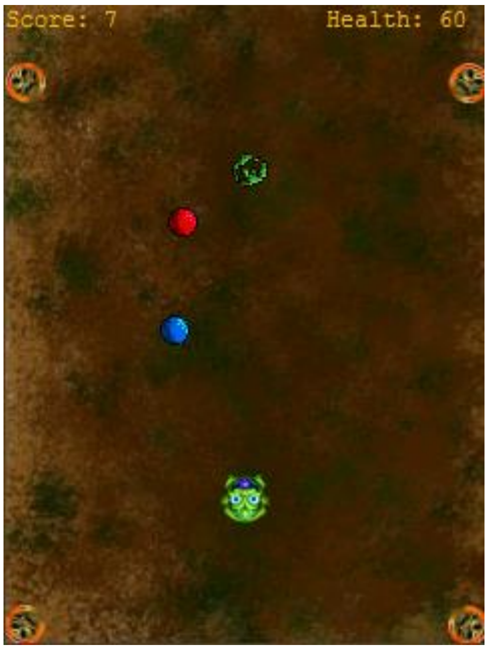


Figure 3

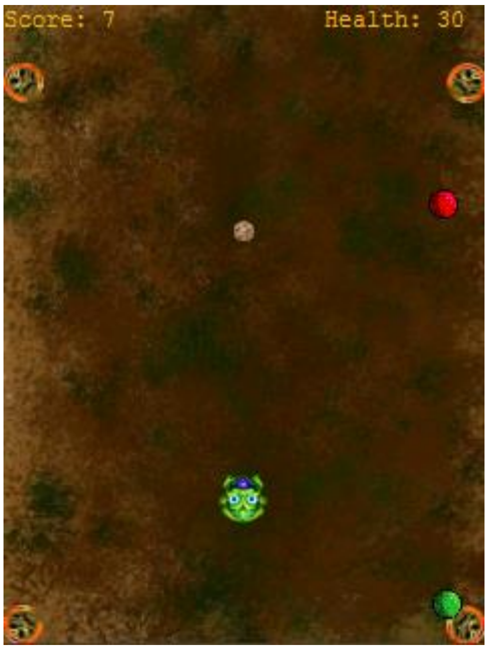


Figure 4

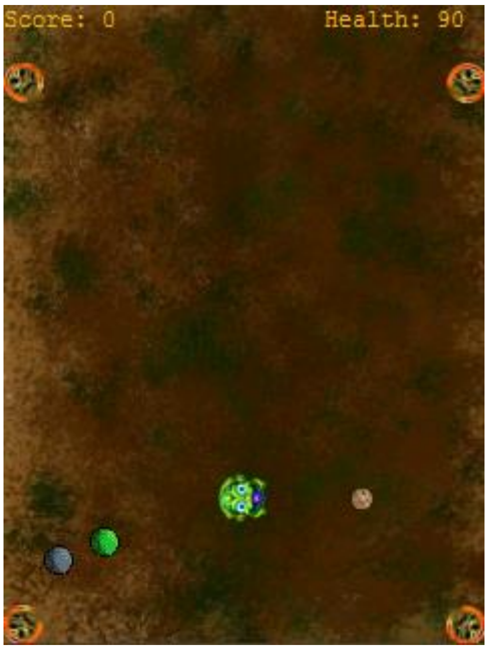


Figure 5

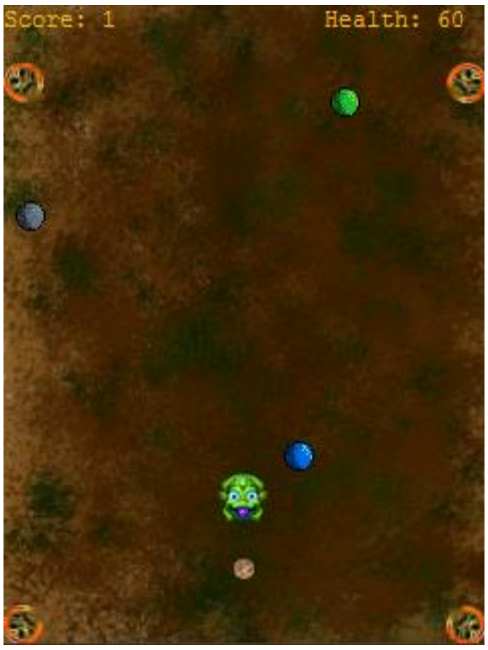


Figure 6

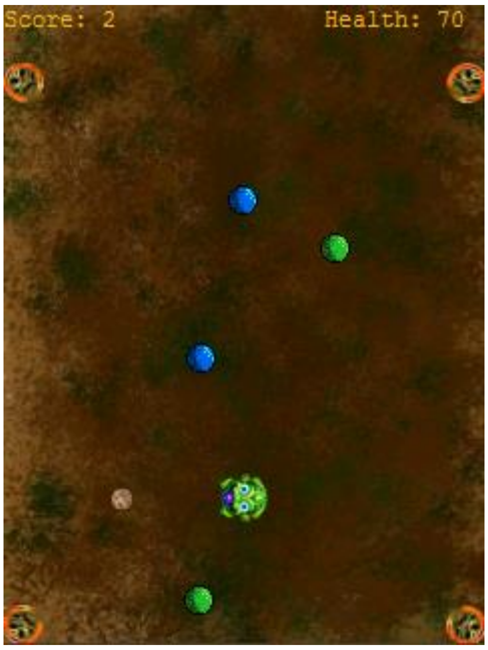


Figure 7

Figure 8