تمرین دوم درس سیستم‌های نهفته بی‌درنگ

پارمیدا ضرغامی • سروش میرزاسروری • محمد فرهی • مهدی وکیلی

خرداد 1402

نکات مهم:

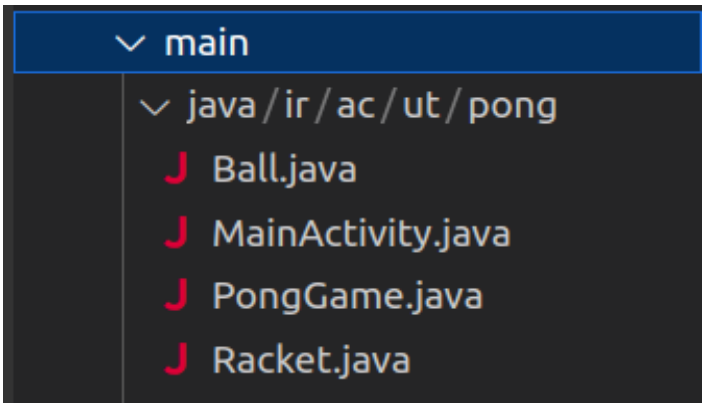
نحوه شکست کار بین اعضا:

در این پروژه ابتدا پس از بررسی ابعاد و جوانب پروژه و مشورت گروهی، وظایف به صورت زیر تقسیم گردید:

پارمیدا ضرغامی مسئول پیاده سازی رابط کاربری و گرافیک پروژه، سروش میرزاسروری مسئول پیاده سازی بک اند، منطق و چگونگی عملکرد کد پروژه، محمد فرهی مسئول ادغام دو بخش بالا و مهدی وکیلی مسئول تست و بررسی و اطمینان از صحت اجرای برنامه بودند. اعضا وظایف محوله را انجام داده و در نهایت نتیجه نهایی به تایید اعضا رسید.

نحوه پیاده‌سازی و توضیح کد :

به طور کلی منطق برنامه در ۴ فایل java پیاده سازی شده است :



فایل MainActivity.java در حقیقت به نوعی entry point برنامه است که در آن متدهای onCreate و onResume و onPause پیاده سازی و override می‌شوند. در متد onCreate از کلاس PongGame که کلاس اصلی بازی است، ساخته می‌شود و با طرح رابط کاربری (User Interface Layout) مرتبط می‌شود. همچنین در متدهای onResume و onPause منطق مربوط به وقفه و ادامه برنامه پیاده‌سازی می‌شود.

```java
public class MainActivity extends AppCompatActivity {
    private PongGame pongGame;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        pongGame = new PongGame(this);
        setContentView(pongGame);
    }

    @Override
    protected void onResume() {
        super.onResume();
        pongGame.resume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        pongGame.pause();
    }
}
```

در فایل PongGame.java منطق برنامه پیاده سازی شده است. در شکل زیر فیلدهای تعریف شده در این کلاس را مشاهده می‌کنید. همانطور که مشخص است، نرخ بروز رسانی صفحه ۲۰ میلی‌ثانیه در نظر گرفته شده است. یعنی سرعت بازی ۵۰ فریم در ثانیه در نظر گرفته شده‌است :

```java
public static final Float FRACTION = 1F;
// Variables for game objects and properties
private Thread gameThread;
private SurfaceHolder surfaceHolder;
private volatile boolean playing;
private final boolean paused = true;
private final Paint paint;

private final SensorManager sensorManager;
private final Sensor accelerometer;
private final Sensor rotationSensor;
public static Integer UPDATE_RATE_MS = 20;
public static Float DELTA_IN_SECONDS = UPDATE_RATE_MS / 1000F;
public static Float DESK_WIDTH = 0.5F;
private Racket racket;
private Ball ball;
```

در constructor این کلاس، یک سری مقداردهی اولیه برای فیلدها صورت می‌گیرد. مشاهده می‌شود که دو نوع سنسور از نوع TYPE_LINEAR_ACCELERATION و
TYPE_GAME_ROTATION_VECTOR از آبجکت sensorManager دریافت می‌شود :

```
public PongGame(Context context) {
    super(context);
    surfaceHolder = getHolder();
    paint = new Paint();

    sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
    accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
    rotationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GAME_ROTATION_VECTOR);

    this.racket = new Racket();
    this.ball = new Ball(20F);

}
```

متد run بخش اصلی برنامه است که در آن همواره و در یک loop، به ترتیب متد update و draw و control صدا زده می‌شوند. متد control وظیفه کنترل سرعت بازی را بر عهده دارد و هر ۲۰ میلی‌ثانیه thread برنامه را به خواب می‌برد :

```
// Game loop
@Override
public void run() {
    while (playing) {
        update();
        draw();
        control();
    }
}
// Control the game loop speed
private void control() {
    try {
        Thread.sleep(UPDATE_RATE_MS);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

در متد update، دو آبجکت racket و ball با توجه به عرض و طول صفحه صفحه، پارامترهای حرکت خود مثل مکان و سرعت و شتاب را بروز رسانی می‌کنند. سپس عمل برخورد توپ به راکت (در صورت امکان) انجام می‌شود :

```
// Update game objects and properties
private void update() {
    this.racket.setViewWidth(getWidth());
    this.racket.update();

    this.ball.setViewWidth(getWidth());
    this.ball.setViewHeight(getHeight());
    this.ball.update();
    this.ball.applyCollision(this.racket);


}
```

در متد draw نیز آبجکت‌های توپ و راکت، با کمک ابزار canvas روی صفحه کشیده می‌شوند. دقت شود که زمان کشیدن اجسام، باید canvas به وسیله surfaceHolder قفل و سپس unlock شود:

```
// Draw game objects on the canvas
private void draw() {
    if (surfaceHolder.getSurface().isValid()) {
        Canvas canvas = surfaceHolder.lockCanvas();
        canvas.drawColor(Color.BLACK);

        this.racket.draw(canvas, paint);

        this.ball.draw(canvas, paint);

        surfaceHolder.unlockCanvasAndPost(canvas);
    }
}
```

مقادیر سنسورها نیز، در متد onSensorChanged خوانده می‌شوند. در حقیقت این متد یک callback است که هنگامی که رویدادی از نوع تغییر مقدار سنسور رخ دهد، صدا زده می‌شود. در این متد ابتدا نوع سنسوری که باعث رویداد شده است، مشخص می‌شود سپس مقدار جدید در فیلد مناسب ذخیره می‌شود :

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_GAME_ROTATION_VECTOR) {
        Float currentRotation = event.values[2];
        this.racket.updateRotation(currentRotation);
    }
    if (event.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
        Float currentAcc = event.values[0];
        this.racket.updateAcc(currentAcc);

    }
}
```

برای پیاده سازی منطق reset بازی نیز، از متد onTouchEvent بهره بردیم که زمان رخداد رویداد ضربه بر روی صفحه، راکت و توپ ریست می‌شوند:

```
@Override
public boolean onTouchEvent(MotionEvent e) {
    if(e.getAction() == MotionEvent.ACTION_MOVE) {
        this.ball.reset();
        this.racket.reset();
    }

    return true;
}
```

در متدهای pause و resume نیز منطق برنامه زمان وقفه و ادامه یافتن thread بازی را پیاده سازی کردیم که به ترتیب در آن‌ها سنسورها را در سنسور منیجر unregister و register می‌کنیم :

```java
public void resume() {
    playing = true;
    gameThread = new Thread(this);
    gameThread.start();
    sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_GAME);
    sensorManager.registerListener(this, rotationSensor, SensorManager.SENSOR_DELAY_GAME);
}

// Method to pause the game
public void pause() {
    playing = false;
    try {
        gameThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    sensorManager.unregisterListener(this);
}
```

در فایل Ball.java منطق مربوط به توپ بازی پیاده سازی می‌شود. در شکل زیر فیلدهای این کلاس و همچنین متدهای های مربوط به مقداردهی اولیه مشاهده می‌شود :

```java
public class Ball {
    private Integer VIEW_WIDTH;
    private Integer VIEW_HEIGHT;
    private Float x;
    private Float y;
    private Float speedX;
    private Float speedY;
    private Float G = 0F;

    private final Float RADIUS;

    public Ball(Float ballRadius) {
        this.RADIUS = ballRadius;
        init();
    }

    private void init() {
        this.x = 540F;
        this.y = 0F;
        this.speedX = 0F;
        this.speedY = 0F;
    }
```

در شکل زیر متدهایی که قسمت update کلاس PongGame استفاده می‌شد، نشان داده شده است. لازم به ذکر است که چون برای حرکت توپ هیچ ورودی از بیرون نمی‌گیریم، واحد حرکت به جای متر، پیکسل در نظر گرفته شده و دیگر تبدیل متر به پیکسل را نداریم (مثلا شتاب به صورت پیکسل بر مجذور ثانیه است). در دو متد اول، طول و عرض صفحه به آبجکت داده می‌شود. در متد update نیز سرعت و مکان توپ بروز رسانی می‌شوند و همچنین برخورد به دیواره‌ها نیز چک می‌شوند. همچنین برای ست کردن G در متد setViewHeight، از آنجا که در یک ثانیه باید یک سوم طول صفحه طی شود، طبق رابطه $\Delta x = 1/2 * g\Delta t$ و قرار دادن VIEW_HEIGHT/3 به جای Δx و قرار دادن ۱ به جای Δt ، به مقدار مناسب برای g می‌رسیم :

```
public void setViewWidth(int width) {
    this.VIEW_WIDTH = width;
}

public void setViewHeight(int height) {
    this.VIEW_HEIGHT = height;
    this.G = 2F / 3F * this.VIEW_HEIGHT
}

public void update() {

    speedY += G;

    // Update ball position
    x += speedX;
    y += speedY;

    // Check for collision with walls
    if (x - RADIUS < 0 ||
    x + RADIUS > VIEW_WIDTH) {
        speedX = -speedX;
    }

    // Check for collision with walls
    if (y - RADIUS < 0 ||
    y + RADIUS > VIEW_HEIGHT) {
        speedY = -speedY;
    }
}
```

در متد applyCollision عمل برخورد (در صورت وجود شرایط) انجام می‌گیرد. در این متد ابتدا دو سر پاره‌خطی که نشان دهنده سطح بالایی راکت است به دست می‌آید. سپس در صورت وجود شرایط برخورد، جهت بردارهای سرعت توپ طبق معادلاتی که در صورت پروژه آورده شده‌است، انجام می‌گیرد :

```
public void applyCollision(Racket racket) {
    RectF racketRect = racket.getRacketRect();

    PointF leftPoint = new PointF();
    PointF rightPoint = new PointF();
    leftPoint.x = racketRect.left + (float) (1F - Math.cos(Math.toRadians(racket.getRotation()))) * racketRect.width() / 2;
    rightPoint.x = racketRect.right - (float) (1F - Math.cos(Math.toRadians(racket.getRotation()))) * racketRect.width() / 2;

    leftPoint.y = racketRect.top + (float) (Math.sin(Math.toRadians(racket.getRotation()))) * racketRect.width() / 2;
    rightPoint.y = racketRect.top - (float) (Math.sin(Math.toRadians(racket.getRotation()))) * racketRect.width() / 2;
    if (isCollided(leftPoint, rightPoint)) {
        float tempSin = (float) Math.sin(2 * Math.toRadians(racket.getRotation()));
        float tempCos = (float) Math.cos(2 * Math.toRadians(racket.getRotation()));
        float newSpeedY = -1 * this.speedX * tempSin - this.speedY * tempCos;
        float newSpeedX = this.speedX * tempCos + this.speedY * tempSin;
        this.speedX = newSpeedX;
        this.speedY = newSpeedY;
    }
}
```

همچنین در متد isCollided، شرایط وجود برخورد چک می‌شود. به این صورت که چک می‌شود آیا مختصات توپ در معادله پاره خط سطح بالایی راکت قرار می‌گیرد یا خیر.(البته چون فضا به صورت گسسته است، کمی جای noise نیز در نظر گرفته شده و اگر توپ نزدیک پاره خط بود نیز، برخورد رخ داده است) :

```
Boolean isCollided(PointF leftPoint, PointF rightPoint) {
    if (this.x > rightPoint.x || this.x < leftPoint.x)
        return false;
    float tempY = (leftPoint.y - rightPoint.y) / (leftPoint.x - rightPoint.x) * (x - leftPoint.x) + leftPoint.y;
    return this.y + Math.abs(speedY) >= tempY && this.y - Math.abs(speedY) <= tempY;
}
```

در متد draw نیز، توپ به وسیله canvas در صفحه کشیده می‌شود :

```
public void draw(Canvas canvas, Paint paint) {
    // Draw ball
    paint.setColor(Color.RED);
    canvas.drawCircle(x, y, RADIUS, paint);
    paint.setTextSize(50);
}

public void reset() {
    init();
}
```

در فایل Racket.java منطق مربوط به راکت پیاده سازی شده است. در شکل زیر فیلدهای این کلاس و همچنین متدهای های مربوط به مقداردهی اولیه مشاهده می‌شود. لازم به ذکر است که در فیلد position مختصات مرکز راکت نگهداری می‌شود :

```
public class Racket {
    private Float speed;
    private Float acc;
    private Float rotation;
    private Float position;
    private Float racketWidth;
    private final RectF racketRect;
    private Integer VIEW_WIDTH;
    public static final Float ASCENDING_ACC_SENSITIVITY_FACTOR = 10F;
    public static final Float DESCENDING_ACC_SENSITIVITY_FACTOR = 0.1F;

    public Racket() {
        this.racketRect = new RectF();
        init();
    }
    private void init() {
        this.speed = 0F;
        this.acc = 0F;
        this.rotation = 0F;
        this.position = 540F;
        this.racketWidth = 0F;
    }
```

دو متد updateAcc و updateRotation مربوط به بروزرسانی شتاب و زاویه هستند که زمانی که onSensorChanged از کلاس PongGame اجرا می‌شود، صدا زده می‌شوند. در متد updateAcc ابتدا یک فیلتر بر روی مقدار شتاب جدید اعمال می‌شود که اگر از ۰.۵ کوچک‌تر بود، مقدار شتاب راکت صفر در نظر گرفته می‌شود. در غیر این صورت شتاب جدید با اثر دادن یک SENSIVITY_FATOR به عنوان شتاب جدید در نظر گرفته می‌شود. علت تاثیر دادن این FACTOR ملموس کردن اثر شتاب اندازه گیری شده توسط سنسور در شتاب راکت است. در متد updateRotation هم مقدار چرخش جدید هم بر حسب درجه در فیلد مناسب ذخیره می‌شود :

```
public void updateAcc(Float currentAcc) {
    if (Math.abs(currentAcc) < ACC_HIGH_FILTER_VALUE)
        this.acc = 0F;
    else {
        if (Math.signum(this.speed) != Math.signum(currentAcc))
            this.acc = currentAcc  * DESCENDING_ACC_SENSITIVITY_FACTOR;
        else
            this.acc = currentAcc  * ASCENDING_ACC_SENSITIVITY_FACTOR;
    }
}

public void updateRotation(Float currentRotation) {
    this.rotation = (float) Math.toDegrees(currentRotation) * -2;
}
```

در متد update نیز، مکان و سرعت راکت تنظیم می‌شود. از آنجا که سرعت بر حسب متر/ثانیه است و مکان باید بر حسب پیکسل باشد، برای رسیدن به مقدار واقعی ضریب

VIEW_WIDTH / DESK_WIDTH نیز اعمال می‌شود. در ادامه سرعت با توجه به شتاب بروزرسانی می‌شود؛ همچنین برای یک مقدار اصطکاک نیز جهت توقف حرکت راکت نیز در سرعت راکت اثر داده شده است. در ادامه اعمال فیلتر بر روی سرعت و همچنین چک کردن برخورد راکت به دیواره ها جهت صفر کردن سرعت راکت انجام می‌شود :

```
public void update() {
    this.position += (this.speed * DELTA_IN_SECONDS) * VIEW_WIDTH / DESK_WIDTH;
    this.speed += this.acc * DELTA_IN_SECONDS;
    this.speed -= Math.signum(this.speed) * FRICTION;
    if (Math.abs(this.speed) < SPEED_HIGH_FILTER_VALUE)
        this.speed = 0F;

    if (this.position + this.racketWidth / 2 < 0) {
        this.position = -1 * this.racketWidth / 2;
        this.speed = 0F;
    }

    if (this.position - this.racketWidth / 2 > VIEW_WIDTH) {
        this.speed = 0F;
        this.position = VIEW_WIDTH - this.racketWidth / 2;
    }
}
```

در متد draw نیز ابتدا با فیلدهایی مثل position و طول و عرض مستطیل، آبجکت مستطیل راکت را ساخته و سپس آن را با کمک canvas و اعمال چرخش رسم می‌کنیم :

```
public void draw(Canvas canvas, Paint paint) {
    paint.setColor(Color.WHITE);
    this.racketWidth = canvas.getWidth() / 3F;
    this.racketRect.left = this.position - this.racketWidth / 2;
    this.racketRect.top = canvas.getHeight() - (canvas.getHeight() / 4);
    this.racketRect.right = this.position + this.racketWidth / 2;
    this.racketRect.bottom = this.racketRect.top + 10;

    canvas.save();
    canvas.rotate(this.rotation, this.racketRect.centerX(), this.racketRect.centerY());
    canvas.drawRect(this.racketRect, paint);
    canvas.restore();
}
```

در این قسمت سعی شد نحوه طراحی و پیاده سازی این برنامه همراه ذکر مفروضات، توضیح داده شوند.

مشخصات سکوی نرم‌افزاری و سخت‌افزاری در پیاده سازی:

پیکربندی برای پلتفرم نرم افزاری و سخت افزاری مورد استفاده در پروژه به نیازهای خاص و دستگاه های هدف بستگی دارد. بر اساس نیازمندی و ابعاد پروژه، مشخصات سکوهای مورد استفاده به طریق زیر بود:

بستر نرم افزاری:
- حداقل نسخه Android SDK: حداقل نسخه Android SDK مورد نیاز پروژه باید در پیکربندی پروژه مشخص شود. قدیمی ترین نسخه اندرویدی که بازی می تواند روی آن اجرا شود را تعیین می کند. طبق توضیحات ارائه شده در صورت پروژه، نیاز بود که برنامه در اندرویدهای بالاتر از 6 قابل اجرا باشد که این مورد لحاظ شد.

- Target Android SDK Version: نسخه Android SDK هدف باید بر اساس ویژگی های مورد نظر و سازگاری با آخرین پلتفرم اندروید انتخاب شود.

- ابزارهای توسعه اندروید: این پروژه را می توان با استفاده از اندروید استودیو توسعه داد که یک محیط توسعه جامع برای توسعه برنامه اندروید فراهم می کند. این شامل ابزارهایی برای کدنویسی، اشکال زدایی و آزمایش برنامه است.

-ابزار ساخت gradle: با استفاده از این ابزار وابستگی بین اجزای مختلف برنامه مدیریت و کنترل شد.

بستر سخت افزاری:
- شتاب سنج و سنسور چرخش: بازی برای کنترل راکت به شتاب سنج و سنسورهای چرخش دستگاه متکی است. پلتفرم سخت افزاری هدف باید شامل این حسگرها باشد. اکثر گوشی‌های هوشمند و تبلت‌های مدرن دارای شتاب‌سنج داخلی و سنسور چرخش هستند.

- اندازه و رزولوشن صفحه نمایش: بازی برای اجرا بر روی دستگاه هایی با اندازه ها و رزولوشن های مختلف صفحه طراحی شده است. مهم است که بازی را در اندازه های مختلف صفحه نمایش آزمایش کنیم تا مطمئن شویم که گرافیک و عناصر رابط کاربری به درستی نمایش داده می شوند.

- گرافیک و نمایشگر: بازی از نمایشگر دستگاه برای رندر کردن گرافیک استفاده می کند. پلتفرم سخت افزاری باید از قابلیت های گرافیکی مورد نیاز (به عنوان مثال OpenGL ES) برای رندرینگ و جلوه های بصری پشتیبانی کند.

ابزارهای مربوط و کتابخانه های مورد استفاده:

این پروژه از ابزارها و کتابخانه های زیر استفاده می کند:

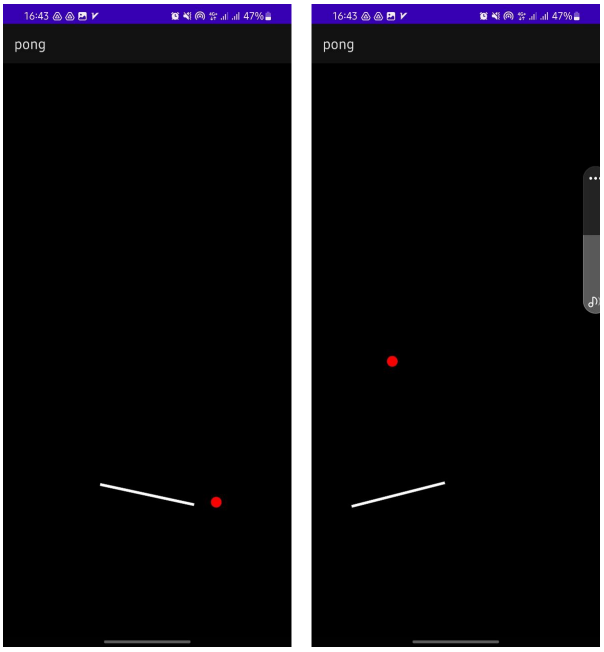
1. Android Studio: این محیط توسعه یکپارچه رسمی (IDE) برای توسعه برنامه اندروید است.
2. زبان برنامه نویسی جاوا: این پروژه با استفاده از جاوا که زبان برنامه نویسی اولیه برای توسعه اندروید است، پیاده سازی شده است.
3. Android SDK: مجموعه ای از ابزارهای توسعه، کتابخانه ها و API ها را به طور خاص برای توسعه پلتفرم اندروید ارائه می دهد.
4. Canvas and Paint: اینها کلاس های داخلی در بسته گرافیکی اندروید هستند که طراحی و نقاشی روی صفحه را امکان پذیر می کنند.
5. SensorManager and Sensor: این کلاس‌ها بخشی از چارچوب اندروید هستند و برای دسترسی و مدیریت داده‌های حسگرها مانند حسگرهای شتاب‌سنج و چرخش استفاده می‌شوند.
6. SurfaceView و SurfaceHolder: این کلاس ها برای ایجاد یک نمای سفارشی با یک سطح طراحی اختصاصی استفاده می شوند که امکان انیمیشن و رندر کارآمد را فراهم می کند.

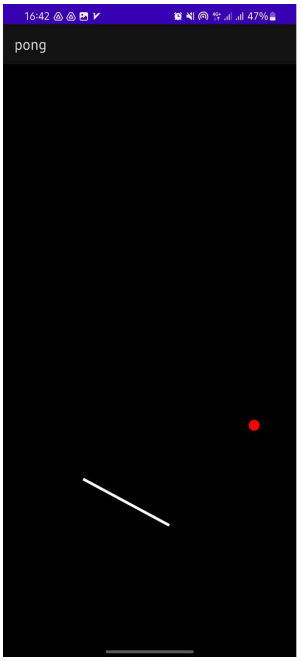
اینها ابزارها و کتابخانه های اصلی درگیر در اجرای پروژه ارائه شده هستند.

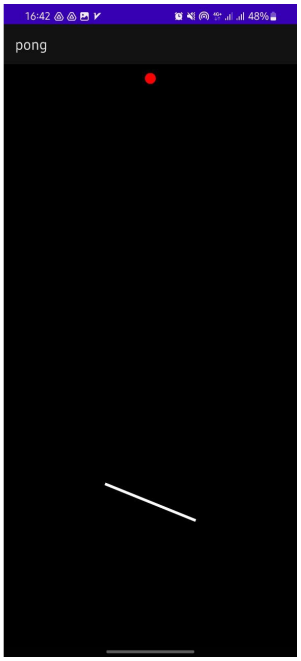
'

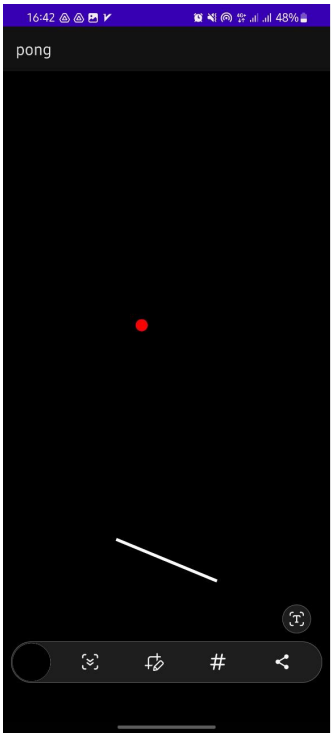
تنظیمات آزمایش ها و سناریوهای تست:

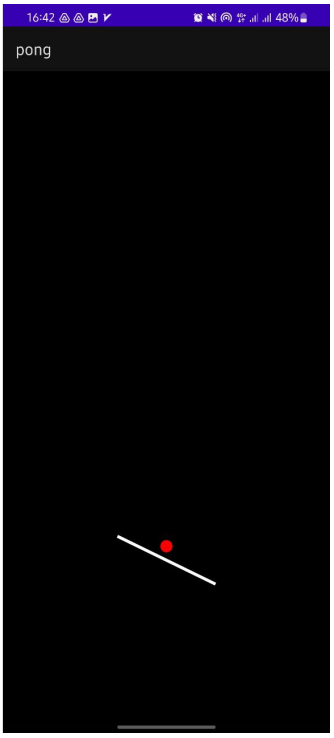
در زیر تصاویر آزمایش برنامه را مشاهده می‌کنیم.

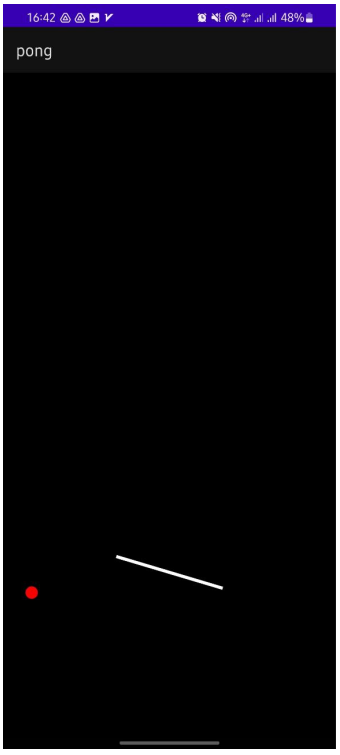


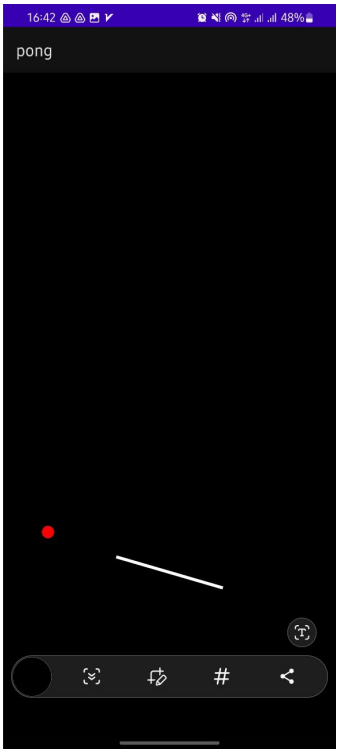


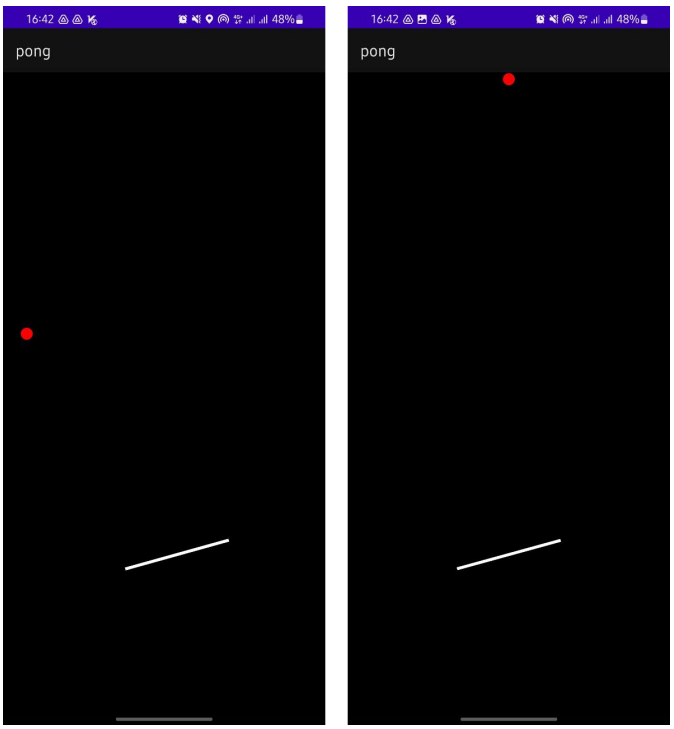



16:42
48%
pong


16:42
48%
pong


16:42
48%
pong

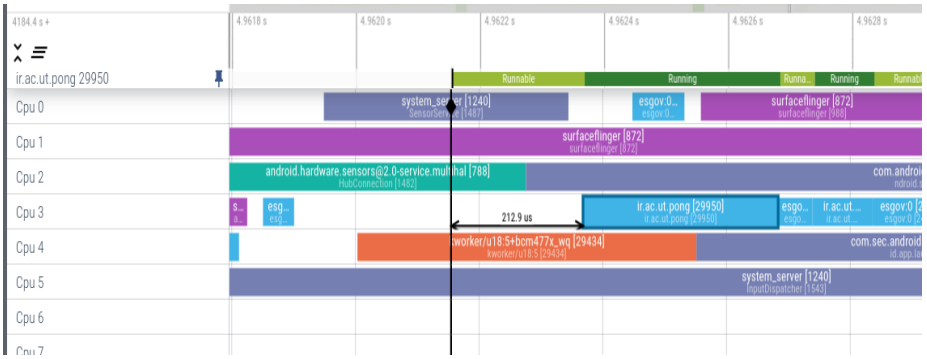

16:42
48%
pong

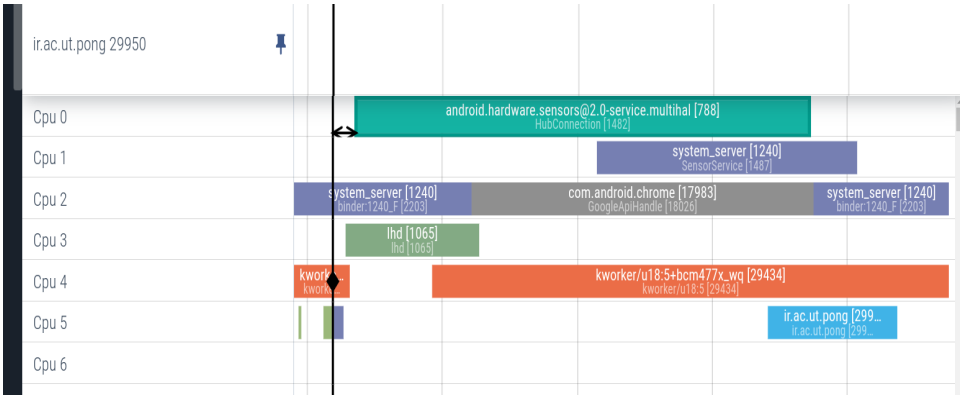


همانطور که در عکسهای بالا مشاهده می شود عملکرد برنامه در تمام حالات به درستی صورت می گیرد. توجه شود که حالات مرزی (حالاتی که توپ به گوشه راکت برخورد می کند) نیز در برنامه مورد تست قرار گرفته و عملکرد درست مشاهده می شود.

سوالات:

.1





.a

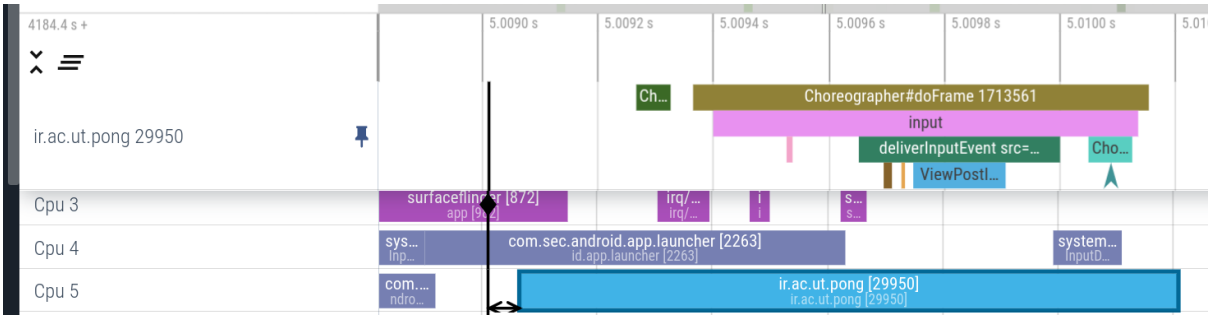
وقتی در یک تلفن هوشمند اندرویدی، دستور خواندن از یک حسگر صادر می‌شود، سیستم عامل با درایور حسگر، ارتباط برقرار کرده و داده‌های حسگر را از آن بازیابی می‌کند. درایور حسگر، به عنوان یک بخش نرم‌افزاری، با حسگر سخت‌افزاری ارتباط برقرار می‌کند و یک رابط برای دسترسی سیستم عامل به داده‌های حسگر فراهم می‌کند. برای خواندن داده‌های حسگر، سیستم عامل درخواستی به درایور حسگر ارسال می‌کند. سپس درایور، داده‌های حسگر را از سخت‌افزار حسگر بازیابی کرده و به سیستم عامل باز می‌گرداند. در ادامه، سیستم عامل داده‌های حسگر را پردازش کرده و آن‌ها را به برنامه‌ای که درخواست داده است، ارسال می‌کند. در مورد حسگر ژیروسکوپ، سیستم عامل برای دریافت داده‌های مورد نیاز برای تشخیص نرخ چرخش و جهت دستگاه، یک درخواست به درایور حسگر ژیروسکوپ ارسال می‌کند. سپس درایور حسگر ژیروسکوپ، اطلاعات مورد نیاز را از سخت‌افزار حسگر ژیروسکوپ بازیابی کرده و به سیستم عامل باز می‌گرداند. در ادامه، سیستم عامل، از این داده‌ها برای به‌روزرسانی جهت و نرخ چرخش دستگاه به صورت real-time استفاده می‌کند. به‌طور مثال، اگر دستگاه را به سمت چپ بچرخانید، حسگر ژیروسکوپ تغییرات ایجاد شده در نرخ چرخش و جهت دستگاه را از طریق داده‌هایی که به سیستم عامل ارسال شده‌اند، به برنامه‌ای که درخواست داده است، اعلام می‌کند تا برنامه بتواند به این تغییرات واکنش نشان دهد. همچنین، حسگر ژیروسکوپ در برخی برنامه‌ها و بازی‌هایی که نیاز به حرکت و چرخش دستگاه دارند، برای کنترل دستگاه توسط کاربر، مورد استفاده قرار می‌گیرد.

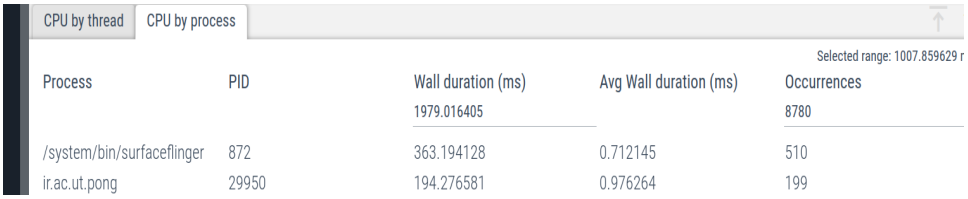
b. خیر، از آنجایی که گرفتن داده از سنسور به صورت async انجام می‌شود و در بافر نگهداری می‌شود، معمولا چنین حالتی که یک thread برا thread دیگری bussy

wait کند پیش نمی‌آید. همچنین تایم فریم‌های 20ms در این پروژه باعث می‌شود که نیاز به تابع draw بسیار sparse است.

c. بیشترین بار پردازش به پردازش‌های گرافیکی تعلق دارد. این امر به دلیل این است که برای رندر کردن انیمیشن‌ها و انجام تغییرات مورد نیاز بر اساس داده‌های گرفته شده از سنسورها، محاسبات بسیاری لازم است که حجم آن‌ها نسبت به محاسبات مورد نیاز برای خواندن داده‌های سنسورها، بیشتر است.

d. همانطور که در عکس مشاهده می‌شود تقریبا ۳ میلی‌ثانیه زمان می‌برد.





.2

دوره تناوب مناسب برای خواندن مقادیر سنسور شتاب سنج (Accelerometer) و ژیروسکوپ (Gyroscope) در برنامه‌های Android به عوامل مختلفی بستگی دارد. در نظر گرفتن دوره تناوب صحیح برای خواندن مقادیر سنسورها می‌تواند تأثیر زیادی بر بهره‌وری و عملکرد برنامه داشته باشد. در ادامه، چند نکته مهم را برای تعیین دوره تناوب مناسب بررسی می‌کنیم:

۱. دقت مورد نیاز: اگر برنامه‌ی شما نیاز به دقت بالا در تشخیص تغییرات سنسورها دارد، ممکن است نیاز داشته باشید دوره تناوب کوتاه تری را انتخاب کنید. برعکس،

اگر دقت بالا از اهمیت کمتری برخوردار است، می‌توانید دوره تناوب بزرگتری را انتخاب کنید.

۲. مصرف باتری: خواندن مقادیر سنسورها منجر به مصرف انرژی باتری می‌شود. برای کاهش مصرف باتری، می‌توانید دوره تناوب بزرگتری را انتخاب کنید. با این حال، باید توجه داشته باشید که انتخاب دوره تناوب بزرگتر ممکن است منجر به عدم تشخیص دقیق تغییرات سنسورها در زمان واقعی شود.

۳. حجم داده: در نظر داشته باشید که هر بار خواندن مقدار سنسورها، داده‌هایی تولید می‌کند که باید پردازش شوند. اگر نیاز به پردازش داده‌های حجیم دارید، ممکن است بخواهید دوره تناوب کوتاهتری را انتخاب کنید.

۴. نیازهای برنامه: نیازهای خاص برنامه‌ی شما نیز می‌تواند در تعیین دوره تناوب تأثیرگذار باشد. برای مثال، اگر برنامه‌ی شما به طور مداوم و در زمینه‌ی پشتیبانی از بازی یا شبیه‌سازی مورد استفاده قرار می‌گیرد، ممکن است نیاز به دوره تناوب کوتاهتری داشته باشید.

با توجه به موارد فوق و نیازهای خاص برنامه باید دوره تناوب را براساس معیارهای مختلفی مانند دقت، مصرف باتری، حجم داده و نیازهای برنامه‌ی خود تنظیم کنیم. ممکن است نیاز باشد برای تعیین دوره تناوب به صورت آزمایشی مقادیر مختلف را امتحان کرده و عملکرد برنامه را بررسی کنیم تا دوره تناوب مناسبی را برای برنامه‌ی خود تعیین کنیم.

3. استفاده از NDK (Native Development Kit) Android به جای SDK (Software Development Kit) Android برای توسعه برنامه‌های اندروید مزایا و معایب خاص خود را دارد. در زیر به برخی از این مزایا و معایب اشاره می‌کنیم:

مزایا:

- عملکرد بهتر: با استفاده از NDK و برنامه‌نویسی به زبان‌های Native مانند C و C++، می‌توانید عملکرد برنامه را بهبود بخشید. زبان‌های Native به‌طور کلی عملکرد سریع‌تری نسبت به زبان‌های مستقر در SDK ارائه می‌دهند.
- قابلیت استفاده از کتابخانه‌های Native: استفاده از NDK به شما اجازه می‌دهد تا از کتابخانه‌های Native و پروژه‌های ساخته شده با استفاده از زبان‌های مستقل از سکوی Android مانند C و C++ بهره‌برداری کنید. این امر به شما اجازه می‌دهد که از کد قبل نوشته شده، ابزارهای بهینه‌تر و کتابخانه‌های پر استفاده استفاده کنید.

- کنترل بیشتر بر عملکرد سخت‌افزار: با استفاده از NDK، شما به طور مستقیم به عملکرد سخت‌افزار دسترسی دارید و می‌توانید کنترل بیشتری بر روی عملکرد سخت‌افزار داشته باشید.

معایب:

- پیچیدگی بیشتر: استفاده از NDK نیاز به تسلط بر زبان‌های Native مانند C و C++ دارد که ممکن است نیاز به یادگیری و توسعه مهارت‌های جدیدی را برای توسعه‌دهندگان مطرح کند. همچنین، پیچیدگی بیشتری نسبت به توسعه با استفاده از SDK Android و زبان‌های مستقل از سکو به همراه دارد.
- کاهش قابلیت همراهی: با استفاده از NDK، برنامه شما به زبان‌های Native وابسته خواهد بود و به همین دلیل قابلیت همراهی برنامه بر روی سکوهای مختلف، مانند دستگاه‌هایی با سیستم‌عامل‌های مختلف، کاهش خواهد یافت.
- پایداری: به عنوان یک زبان بهینه و کم‌سطح، برنامه‌نویسی به زبان‌های Native نیاز به مراقبت و مدیریت دقیق‌تری دارد. اگر کدی ناصحیح یا نامناسب بنویسید، ممکن است به پایداری برنامه آسیب برسد.

به طور کلی، استفاده از NDK Android به دلیل ارائه عملکرد بهتر و قابلیت استفاده از کتابخانه‌های Native می‌تواند برای پروژه‌هایی که نیاز به عملکرد بالا و کنترل دقیق‌تری بر روی سخت‌افزار دارند، مناسب باشد. با این حال، باید در نظر داشت که استفاده از NDK پیچیدگی بیشتری دارد و قابلیت همراهی و پایداری برنامه را کاهش می‌دهد.

4. سنسورهای مبتنی بر سخت‌افزار (Hardware-Based Sensors):
سنسورهای مبتنی بر سخت‌افزار، دستگاه‌های فیزیکی را برای تشخیص و اندازه‌گیری ویژگی‌های محیطی مورد استفاده قرار می‌دهند. این سنسورها به صورت مستقیم به سخت‌افزار دستگاه متصل شده و از طریق سیگنال‌های الکتریکی و تفسیر داده‌هایی که توسط آنها تولید می‌شود، اطلاعات مربوطه را در اختیار برنامه‌های نرم‌افزاری قرار می‌دهند. برخی از سنسورهای مبتنی بر سخت‌افزار شامل شتاب‌سنج (Accelerometer) و ژیروسکوپ (Gyroscope) می‌شوند.

سنسور شتاب‌سنج: این سنسور قادر است تغییرات شتاب را در سه جهت (سه بُعد) دریافت کند. از اطلاعاتی که توسط شتاب‌سنج ارائه می‌شود، می‌توان برای تشخیص حرکت، تشخیص جهت گرانش و تشخیص تغییرات در حالت دستگاه استفاده کرد. در تمرین مذکور

نیز از سنسور شتاب‌سنج برای تشخیص حرکت و تغییرات مکانی راکت (راکت) استفاده می‌شود.

سنسور ژیروسکوپ: این سنسور قادر است نرخ چرخش دستگاه را در سه جهت (سه بُعد) اندازه‌گیری کند. با استفاده از اطلاعات ارائه شده توسط ژیروسکوپ، می‌توان تغییرات در جهت و نرخ چرخش دستگاه را تشخیص داد. در تمرین مذکور نیز از سنسور ژیروسکوپ برای تشخیص و تعقیب جهت و نرخ چرخش راکت استفاده می‌شود.

سنسورهای مبتنی بر نرم‌افزار (Software-Based Sensors):
سنسورهای مبتنی بر نرم‌افزار، از اطلاعاتی که توسط سنسورهای سخت‌افزاری جمع‌آوری می‌شوند، استفاده می‌کنند و با استفاده از الگوریتم‌ها و پردازش‌های نرم‌افزاری، اطلاعات دقیق‌تر و تفصیلی‌تری را ارائه می‌دهند. این سنسورها می‌توانند از طریق تحلیل داده‌های حسگرهای سخت‌افزاری اطلاعاتی را در مورد شرایط محیطی، وضعیت دستگاه و سایر ویژگی‌ها فراهم کنند.

در تمرین مذکور، استفاده از سنسورهای شتاب‌سنج و ژیروسکوپ، که به صورت مبتنی بر سخت‌افزار عمل می‌کنند، وجود دارد. این سنسورها به صورت مستقیم با سخت‌افزار دستگاه متصل شده و اطلاعات مربوطه را از تغییرات شتاب و نرخ چرخش دریافت می‌کنند.

5. تفاوت بین تعریف سنسور به صورت Wake-up و Non-Wake-up به شرح زیر است:
   - تعریف سنسور به صورت Wake-up:
     - وقتی سنسور به صورت Wake-up تعریف می‌شود، به طور معمول این به معنی فعال بودن مداوم سنسور است و برنامه از به خواب رفتن(Sleep) سنسور جلوگیری می‌کند.
     - سنسور به صورت مداوم داده‌های خود را بروزرسانی می‌کند و در صورت تغییر در داده‌های سنسور، به برنامه اطلاع می‌دهد (به صورت Asynchronous).
     - این روش باعث کاهش تأخیر در دریافت بروزرسانی‌ها و دقت بالاتر در کنترل راکت می‌شود. همچنین، برنامه می‌تواند بدون نیاز به بررسی مداوم سنسور، بر روی تغییرات آن واکنش نشان دهد.
   - تعریف سنسور به صورت Non-Wake-up:

- وقتی سنسور به صورت Non-Wake-up تعریف می‌شود، سنسور در حالت خواب (Sleep) قرار دارد و تنها در زمانی که برنامه درخواست خواندن داده‌های سنسور را دارد، فعال می‌شود.
- در این حالت، برنامه به صورت معمولی برای خواندن داده‌های سنسور درخواست می‌دهد و منتظر پاسخ سنسور می‌ماند (به صورت Synchronous).
- این روش باعث کاهش مصرف انرژی در صورت عدم نیاز مداوم به داده‌های سنسور می‌شود. با این حال، ممکن است تأخیر در دریافت بروزرسانی‌ها و دقت کنترل راکت کاهش یابد.

تاثیر تعریف سنسور به صورت Wake-up و Non-Wake-up بر نحوه دریافت بروزرسانی سنسورها و نتیجه کنترل راکت به شرح زیر است:

- در تعریف سنسور به صورت Wake-up، بروزرسانی‌ها به صورت ناگهانی و غیرقابل پیش‌بینی دریافت می‌شوند. این باعث می‌شود که برنامه بلافاصله واکنش نشان دهد و به تغییرات در داده‌های سنسور پاسخ دهد. این تأثیر می‌تواند در دقت بالا در کنترل راکت تاثیرگذار باشد.
- در تعریف سنسور به صورت Non-Wake-up، برنامه برای دریافت بروزرسانی‌ها درخواست می‌دهد و پس از دریافت پاسخ سنسور، عملیات کنترل راکت انجام می‌شود. این باعث می‌شود که ممکن است تأخیری در دریافت بروزرسانی‌ها به وجود بیاید و دقت در کنترل راکت کاهش یابد، اما به دلیل کاهش مصرف انرژی در حالت خواب سنسور، مصرف انرژی کاهش می‌یابد.

مراجع:

مطالب تدریس‌شده در کلاس

لینک‌های موجود در صورت پروژه

https://www.vogella.com/tutorials/AndroidSensor

https://developer.android.com/

https://perfetto.dev//android/android_sensors.html