به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس سیستم های نهفته و بی درنگ

تمرین دوم

نام و نام خانوادگی	کیمیا فخاری – محمدحسین عطایی - دانیال سعیدی – محمد قره حسنلو
شماره دانشجویی	810197632 – 810197650 – 810198461 - 810198571
تاریخ ارسال گزارش	1402.03.06

فهرست

توضیحات کلی کد

کلاس Ball

در کلاس Ball بر اساس فرمول $\Delta x = \frac{1}{2} a\, \Delta t$ مقدار x و y توپ و همچنین سرعت توپ در راستای y به شکل زیر تعریف میشوند:

```
public void updatePosition(float timeDeltaSeconds) {
    // Update position based on current velocity
    x += velocityX * timeDeltaSeconds;
    y += velocityY * timeDeltaSeconds;

    // Update velocity due to gravity
    velocityY += GRAVITATIONAL_ACCELERATION * timeDeltaSeconds;

    // Update polygon
    updatePolygon();
}
```

شکل 1: تغییرات سرعت و مکان توپ

کلاس ResetButton

برای ریست کردن، دکمه ریست به شکل زیر در صفحه بازی تعبیه شده است:

```
public class ResetButton {
    private RectF rect;
    private Paint paint;

    public ResetButton(float left, float top, float right, float bottom) {
        rect = new RectF(left, top, right, bottom);
        paint = new Paint();
        paint.setColor(Color.BLUE); // Red color
    }

    public void draw(Canvas canvas) { canvas.drawRect(rect, paint); }

    public boolean isClicked(MotionEvent event) {
        return rect.contains(event.getX(), event.getY());
    }
}
```

شکل 2: دکمه ریست

کلاس Polygon

این کلاس شامل یک لیست از نقاط با فرمت Vector2 می‌باشد. در آن، یک لیست از نقاط با استفاده از کلاس ArrayList ایجاد می‌شود.

تابع setAsBox که با دریافت چهار پارامتر به عنوان مختصات چهارضلعی، لیست نقاط را با نقاط متناظر با این چهار ضلع مقداردهی می‌کند .

تابع getAxes که لیست بردارهای عمود بر هر هر ضلع از چهارضلعی را برمی‌گرداند. برای این کار، ابتدا برای هر ضلع، برداری از دو نقطه آن را محاسبه و سپس بردار نرمال به آن را به دست می‌آورد.

کلاس Vector2

این کلاس دو متغیر خصوصی x و y را به عنوان مختصات نقطه ذخیره می‌کند .

تابع subtract که با گرفتن یک بردار دیگر به عنوان ورودی، بردار جدیدی با محاسبه تفاضل دو بردار به دست می‌آورد.

تابع perpendicular که برای بردار عمود بر بردار جاری را با محاسبه معکوس مقدار x و y به دست می‌آورد.

تابع dotProduct ضرب داخلی دو بردار را برمی‌گرداند.

تابع rotateAround که با گرفتن مختصات یک مرکز و یک زاویه به عنوان ورودی، بردار جدیدی بر اساس چرخش بردار جاری حول مرکز مشخص شده و به اندازه زاویه مشخص شده در جهت مثبت یا منفی محور x چرخش می‌کند و بردار جدید را بازمی‌گرداند.

```
public Vector2 subtract(Vector2 other) { return new Vector2( x: x - other.x,  y: y - other.y); }

public Vector2 perpendicular() { return new Vector2(-y, x); }

public float dotProduct(Vector2 other) { return x * other.x + y * other.y; }

public Vector2 rotateAround(float centerX, float centerY, float angleInRadians) {
    float xPrime = (float) ((x - centerX) * Math.cos(angleInRadians) - (y - centerY) * Math.sin(angleInRadians) + centerX);
    float yPrime = (float) ((x - centerX) * Math.sin(angleInRadians) + (y - centerY) * Math.cos(angleInRadians) + centerY);
    return new Vector2(xPrime, yPrime);
}
```

شکل 3: توابع استفاده شده در کلاس **Vector2**

کلاس SATCollision

برای تشخیص برخورد بین دو شکل یا چند شکل در فضای دو بعدی استفاده می‌شود.

تابع checkCollision با دو پارامتر Polygon و RectF برای تشخیص برخورد یک چندضلعی با یک مستطیل تعریف شده است. ابتدا یک چندضلعی متناظر با مستطیل بوسیله تابع setAsBox ایجاد شده و سپس تابع checkCollision با دو چندضلعی ورودی فراخوانی می‌شود.

تابع checkCollision با دو پارامتر Polygon برای تشخیص برخورد دو چندضلعی تعریف شده است. ابتدا بردارهایمحور برای هر دو چندضلعی استخراج شده و در یک لیست جمع‌آوری می‌شوند. سپس برای هر محور، با استفاده از توابع project وoverlap ، برخورد بین دو چندضلعی بررسی می‌شود. اگر برخوردی پیدا نشد، تابع false برمی‌گرداند و در غیر این صورت برخورد را تأیید می‌کند و true برمی‌گرداند.

```
public static boolean checkCollision(Polygon a, RectF b) {

    Polygon rectPolygon = new Polygon();
    rectPolygon.setAsBox(b.left, b.top, b.right, b.bottom);

    return checkCollision(a, rectPolygon);
}

public static boolean checkCollision(Polygon a, Polygon b) {
    List<Vector2> axes = new ArrayList<>();
    axes.addAll(a.getAxes());
    axes.addAll(b.getAxes());

    for (Vector2 axis : axes) {
        if (!overlap(project(a, axis), project(b, axis))) {
            return false;
        }
    }

    return true;
}
```

شکل 4: تشخیص برخورد دو چندضلعی

تابع project با استفاده از ضرب داخلی بین بردار محور و نقاط چندضلعی، نقاطی را برمی‌گرداند که بیشترین و کمترین فاصل را در جهت محور مشخص شده دارند .تابع overlap برای بررسی همپوشانی دو بازه روی یک محور مشخص شده است.

کلاس **PongViewGame**

کلاس اصلی این بازی میباشد که توابع مختلفی دارد که کارهای مختلفی ازجمله استفاده از سنسورها در این قسمت قرار گرفته شده است.

```
private void checkCollisions() {
    Polygon ballPolygon = ball.getPolygon();
    Polygon paddlePolygon = getPaddlePolygon();

    if (SATCollision.checkCollision(ballPolygon, paddlePolygon)) {
        // Calculate the angle of reflection based on the paddle's angle
        float angleOfReflection = (float) Math.toRadians(90 - paddleAngle);
        float newVelocityX = (float) (ball.getVelocityX() * Math.cos(angleOfReflection) + ball.getVelocityY() * Math.sin(angleOfReflection));
        float newVelocityY = (float) (-ball.getVelocityX() * Math.sin(angleOfReflection) + ball.getVelocityY() * Math.cos(angleOfReflection));

        ball.setVelocityX(newVelocityX);
        ball.setVelocityY(newVelocityY);

    }

    // Check for wall collisions
    if (ball.getX() - ball.getRadius() < 0 || ball.getX() + ball.getRadius() > getWidth()) {
        ball.reverseVelocityX();
    }

    if (ball.getY() - ball.getRadius() < 0) {
        ball.reverseVelocityY();
    }
}
```

شکل 5: محاسبه سرعت توپ پس از برخورد با راکت یا دیوار

در کانستراکتور، متغیرهای paint و rect مقداردهی اولیه شده و اطلاعات سنسورها گرفته می‌شوند.

```
public PongGameView(Context context) {
    super(context);
    paint = new Paint();
    paint.setColor(0xFF000000); // Set the color to black
    rect = new RectF();

    lastUpdateTime = System.currentTimeMillis();

    sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
    accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_GAME);
    sensorManager.registerListener(this, gyroscope, SensorManager.SENSOR_DELAY_GAME);

    resetButton = new ResetButton(10, 10, 110, 60); // Adjust the position and size as needed
}
```

شکل 6: تعریف سنسورها در کانستراکتور

در این متد onSizeChanged ، اندازه بازی و شکل را تعیین می‌کند و موقعیت توپ را در ابتدای بازی تنظیم می‌کند.

در onDraw چرخش paddle انجام میشود. همچنین مواردی مانند به روز رسانی موقعیت توپ بر اسا سرعتش انجام میشود که برای به دست آوردن دلتای زمان، تفریق مقدار زمان کنونی و آخرین زمانی کع

داشتیم، تقسیم بر 1000 میشود. این کار را بر اساس این قسمت صورت پروژه انجام داده شده است: برای مثال اگر از 30 فریم بر ثانیه استفاده کنیم(یعنی نرخ آپدیت هرگونه تغییر در صفحه دستگاه 30 فریم در ثانیه باشد) این مقدار حدود 1/30 ثانیه می باشد.

در onTouchEvent، وقتی که کاربر دکمه ریست را لمس می‌کند، توپ به موقعیت اولیه بازی بازمی‌گردد.

متد onSensorChanged موقعیت مستطیل و زاویه بازیکن تغییر می‌کند. همچنین در این متد، متد invalidateفراخوانی می‌شود تا صفحه بازی مجدداً رسم شود. برای بررسی تغییرات دستگاه سنسوری استفاده می‌شود.

در صورتی که دستگاه سنسور از نوع شتاب‌سنج باشد، مقدار شتاب در راستای محور x دریافت می‌شود و با ضریبی به نام movementFactor ضرب شده و موقعیت جدید paddle محاسبه می‌شود. در اینجا، paddle از سمت چپ و راست حرکت می‌کند و براساس شتاب دستگاه، بازیکن به چپ یا راستحرکت می‌کند.

در صورتی که مقدار محاسبه شده برای paddle منفی باشد و خارج از صفحه بازی باشد، با استفاده از شرط‌های if وelse ، مستطیل بازیکن به حداقل یا حداکثر مقدار مجاز برای بازیکن حرکت می‌کند.

در صورتی که دستگاه سنسور از نوع ژیروسکوپ باشد، مقدار زاویه بازیکن از طریق چرخش دستگاه در راستای محور z محاسبه و با ضریبی به نام rotationFactor ضرب شده و ازمقدار زاویه بازیکن کم شده و زاویه‌ی جدید بازیکن محاسبه می‌شود. در صورتی که مقدار زاویه جدید بازیکن از مقدار حداکثر مجازی که در متغیر maxAngleRotation تعریف شده است، بیشتر نباشد، زاویه بازیکن به مقدار جدید تغییر می‌کند.

```java
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float ax = event.values[0];
        float left = rect.left - ax * movementFactor;
        float right = rect.right - ax * movementFactor;

        // Prevent the rectangle from
        if (left < 0) {
            left = 0;
            right = rect.width();
        } else if (right > getWidth()) {
            right = getWidth();
            left = getWidth() - rect.width();
        }
        rect.set(left, rect.top, right, rect.bottom);
    } else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
        float zRotation = event.values[2];
        float newPaddleAngle = paddleAngle - zRotation * rotationFactor;
        if (Math.abs(newPaddleAngle) <= this.maxAngleRotation) {
            paddleAngle = newPaddleAngle;
        }
    }
    invalidate(); // Request a redraw
}
```

شکل 7: نحوه استفاده از سنسورها در بازی

در متد checkCollisions ، بررسی می‌شود که آیا توپ با دیوارهای صفحه بازی و یا paddle برخورد کرده است یا خیر. بعد از آن مقدار سرعت جدید در راستای x و y با توجه به فرمول های گفته شده در پروژه به دست می آوریم.

```
private void checkCollisions() {
    Polygon ballPolygon = ball.getPolygon();
    Polygon paddlePolygon = getPaddlePolygon();

    if (SATCollision.checkCollision(ballPolygon, paddlePolygon)) {
        // Calculate the angle of reflection based on the paddle's angle
        float angleOfReflection = (float) Math.toRadians(90 - paddleAngle);
        float newVelocityX = (float) (ball.getVelocityX() * Math.cos(angleOfReflection) + ball.getVelocityY() * Math.sin(angleOfReflection));
        float newVelocityY = (float) (-ball.getVelocityX() * Math.sin(angleOfReflection) + ball.getVelocityY() * Math.cos(angleOfReflection));

        ball.setVelocityX(newVelocityX);
        ball.setVelocityY(newVelocityY);

    }

    // Check for wall collisions
    if (ball.getX() - ball.getRadius() < 0 || ball.getX() + ball.getRadius() > getWidth()) {
        ball.reverseVelocityX();
    }

    if (ball.getY() - ball.getRadius() < 0) {
        ball.reverseVelocityY();
    }
}
```

شکل 8: تابع مربوط به برخورد توپ به دیواره ها و **paddle**

در متد getPaddlePolygon، مختصات مستطیل بازیکن را بر اساس زاویه بازیکن و مختصات گوشه‌های آن را به شکل یک چند ضلعی تبدیل می‌کند تا بتواند محل برخورد با توپ را دقیق تر به دست آورده و بازتاب درستی از توپ هنگام بازگشت نشان داده شود.

متد drawBorder مربوط به رسم حاشیه بازی است.

همچنین در کنار کار امتیازی، بخش حرکت paddle به سمت راست و چپ در اواخر کار اضافه شد و همچنین با حرکت در راستای z دادن شتاب به توپ اضافه شد.

کتابخانه های مورد استفاده

کتابخانه:android.content.Context

به عنوان یک شیء محیطی برای دسترسی به منابعی مانند فایل‌ها، دیتابیس‌ها و غیره در برنامه‌های اندروید استفاده می‌شود.

کتابخانه:android.graphics.Canvas

برای رسم شکل‌ها و نمودارها در نمایش گرافیکی استفاده می‌شود

کتابخانه:android.graphics.Paint

برای تنظیم ویژگی‌های قلم در نمایش گرافیکی استفاده می‌شود. در این مثال، با استفاده از این کتابخانه، یک شیء از کلاس Paint برای تنظیم ویژگی‌های قلم مورد استفاده قرار گرفته است.

کتابخانه:android.graphics.RectF

برای تعریف و کنترل مستطیل‌ها در نمایش گرافیکی استفاده می‌شود.

کتابخانه:android.hardware.Sensor

برای دسترسی به سنسورهای گوشی موبایل مانند شتاب‌سنج و ژیروسکوپ استفاده می‌شود.

کتابخانه:android.hardware.SensorEventListener

برای گوش دادن به تغییرات سنسورها و دریافت داده‌های آن‌ها استفاده می‌شود.

کتابخانه:android.hardware.SensorManager

برای مدیریت سنسورهای گوشی موبایل مانند ثبت و لغو گوش دادن به تغییرات سنسورها، استفاده می‌شود.

کتابخانه:android.view.MotionEvent

برای تعامل کاربر با تاچ‌اسکرین در برنامه‌های اندروید استفاده می‌شود. در این مثال، با استفاده از این کتابخانه، تابعی برای پردازش رویدادهای تاچ در View تعریف شده است.

سوالات

1-

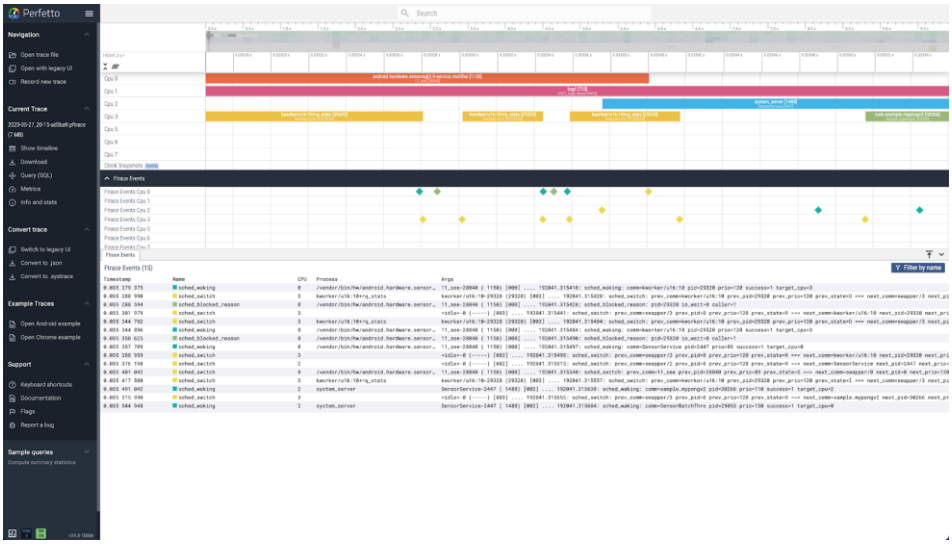
ب) بله، در فراخوانی های سیستمی ممکن است تعارضی بین استفاده از کتابخانه مربوط به گرافیک و بروزرسانی سنسورها وجود داشته باشد. این امر ممکن است به دلیل این باشد که اجزای مختلف بازی، نیاز به پردازش های سنگین دارند و این پردازش ها ممکن است باعث تعویق در اجرای دیگر فراخوانی های سیستمی شوند.

در واقع، با استفاده از Perfetto به عنوان یک ابزار مانیتورینگ، می توان تعارضات بین فعالیت های مختلف را کاهش داد و بهبود عملکرد بازی پونگ را تسهیل کرد. از طریق مانیتورینگ عملکرد سیستم با استفاده ازPerfetto ، می توانناقط ضعف سیستم را تشخیص داد و بهینه سازی آنها را انجام داد. به عبارت دیگر، با استفاده از این ابزار، می توان تعارضات بین عملکرد گرافیکی و بروزرسانی سنسورها را کاهش داد و برای بهبود عملکرد بازی پونگ اقدامات لازم را انجام داد.

ج) همانطور که در شکل زیر مشخص است، سنسورها cpu را خیلی بیشتر نسبت به پردازش های گرافیکی به کار گرفته اند.

الف و د)

می‌توان میزان استفاده از CPU و I/O را در این بازه زمانی مانیتور کرد. با استفاده ازPerfetto ، می‌توان تعیین کرد که چه مراحلی در فرآیند درخواست خواندن داده از سنسور انجام می‌شود و در کدام مرحله از فرآیند، کدام منبع سیستم مانند CPU یا I/O بیشترین بار را دارد.



شکل 9: کارهای انجام شده در طول فعالیت سنسور

2– در بازی‌ها به خواندن مکرر و با دقت سنسورها نیاز است تا وضعیت بازیکن و عناصر بازی مانند توپ به درستی به روز رسانی شود. اگر دوره تناوب خواندن سنسورها کم باشد، حرکات بازیکن و عناصر بازی به صورت نامنظم و پرشی خواهد بود. اگر دوره تناوب خواندن سنسورها زیاد باشد، واکنش‌های بازی به حرکات بازیکن با تأخیر انجام خواهد شد و تجربه بازی را خراب می‌کند.

بنابراین دوره تناوب Game که حدود 20-30 msاست، بهترین گزینه برای خواندن مکرر سنسورها در یک بازی است. برای این امر دوره تناوب خود بازی با استفاده از کد زیر درنظر گرفته شده است که طبق سندها برابر 20 میلی ثانیه است.

```
sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_GAME);
sensorManager.registerListener(this, gyroscope, SensorManager.SENSOR_DELAY_GAME);
```

شکل 10: دوره تناوب خواندن مقادیر سنسورها

3–

مزایا:

عملکرد بهتر: استفاده از NDK به برنامه نویسان اجازه می دهد تا بخش هایی از برنامه را که به زبان سی یا سی++ نوشته شده اند، به صورت native اجرا کنند. این کار می تواند عملکرد بازی را بهبود بخشد.

قابلیت دسترسی به سطوح پایین تر: با استفاده ازNDK ، برنامه نویسان می توانندبه سطوح پایین تر سیستم عامل Android دسترسی پیدا کنند و از ویژگی هایی مانند مدیریت حافظه و تنظیمات سیستم استفاده کنند که امکانات مشابهی در SDK وجود ندارد.

قابلیت استفاده از کتابخانه های سیستمی: با استفاده ازNDK ، برنامه نویسان می توانند از کتابخانه های سیستمی مانندOpenCV ، OpenGLو ... استفاده کنند که قابلیت های بسیار بیشتری نسبت به کتابخانه های Java در SDK دارند.

معایب:

پیچیدگی بیشتر: برای برنامه نویسانی که زبان سی یا سی++ را به خوبی نمی‌شناسند، استفاده از NDK ممکن است پیچیده و سخت باشد.

بهبود عملکرد محدود: استفاده از NDK ممکن است تنها در برخی از موارد به بهبود عملکرد برنامه بیانجامد و در برخی موارد ممکن است عملکرد برنامه را بهبود ندهد.

ابزارهایی که به صورت native نیستند: برخی از ابزارهایی که برای توسعه بازی های Android استفاده می‌شوند، مانند موتور بازیUnity ، به صورت native نیستند و برای استفاده از آن‌ها نیاز به SDK دارید.

4- حسگرهای مبتنی بر سخت افزار اجزای فیزیکی هستند که در طراحی دستگاه ادغام می شوند و اندازه گیری مستقیم عوامل محیطی را ارائه می دهند. نمونه هایی از حسگرهای مبتنی بر سخت افزار عبارتند از شتاب سنج و ژیروسکوپ.

شتاب سنج ها و ژیروسکوپ ها هر دو نمونه هایی از حسگرهای مبتنی بر سخت افزار هستند. شتاب‌سنج‌ها شتاب خطی را اندازه‌گیری می‌کنند و تغییرات در سرعت و جهت حرکت مانند کج شدن یا تکان را تشخیص می‌دهند. از سوی دیگر، ژیروسکوپ ها، سرعت زاویه ای را اندازه گیری می کنند و تغییرات در جهت یا سرعت چرخش را تشخیص می دهند.

از سوی دیگر، حسگرهای مبتنی بر نرم‌افزار، حسگرهای مجازی هستند که از داده‌های منابع دیگر مانند حسگرهای سخت‌افزاری برای استخراج اطلاعات در مورد عوامل محیطی استفاده می‌کنند. نمونه هایی از حسگرهای مبتنی بر نرم افزار شامل سنسورهای GPS، حسگرهای مجاورت و حسگرهای نور هستند. این حسگرها برای تفسیر داده ها و به دست آوردن بینش معنادار به الگوریتم ها و مدل های ریاضی متکی هستند.

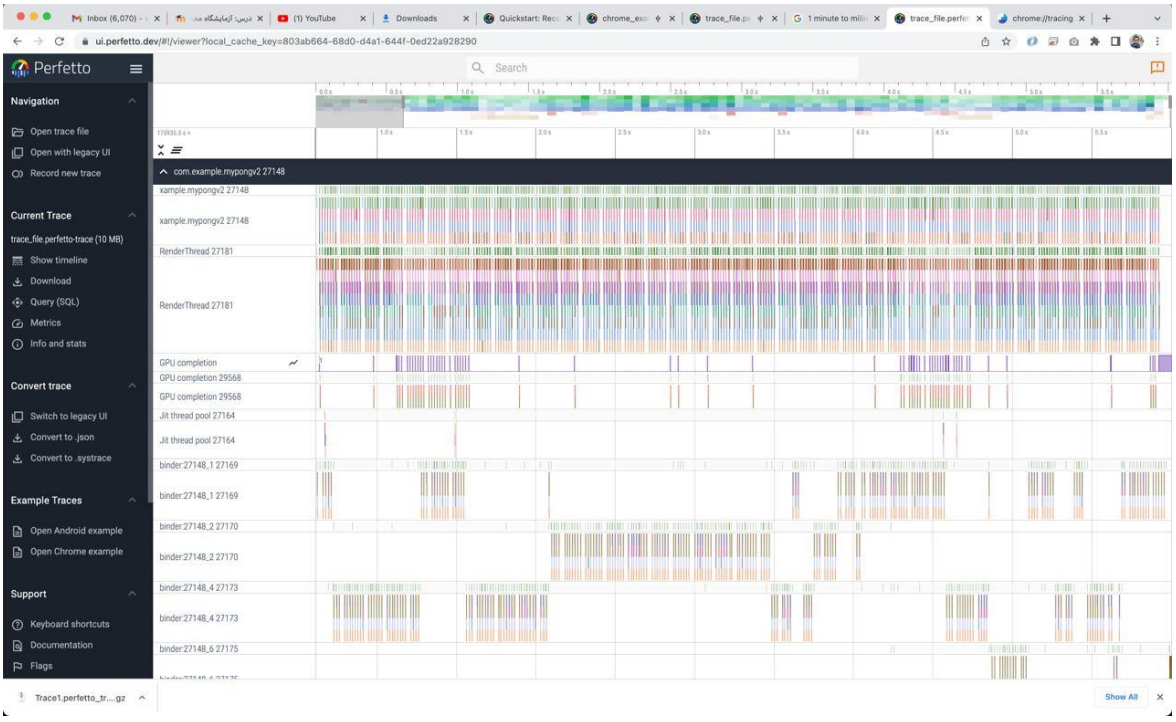
5- در زمینه توسعه بازی، "up-wake" و "non-up-wake" به حالت‌های حسگر اشاره می‌کنند که تعیین می‌کنند یک حسگر چقدر به‌روزرسانی‌ها را از محیط خود دریافت می‌کند.

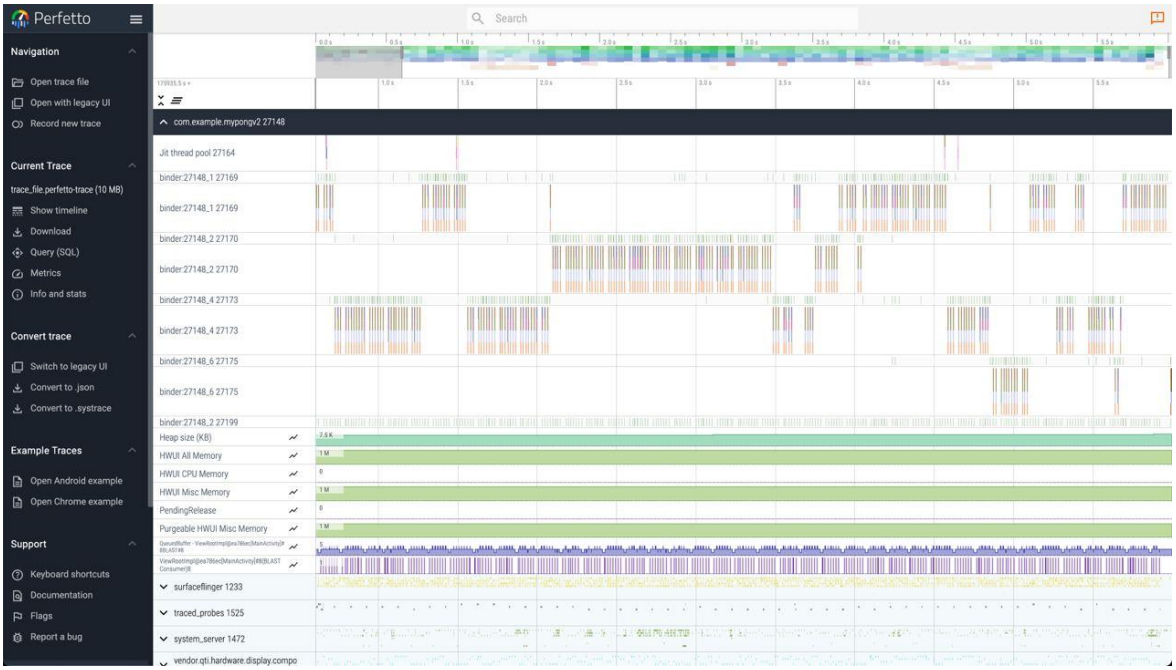
در حالت up-wake هر بار که موتور بازی به‌روزرسانی می‌شود، یک حسگر به‌روزرسانی می‌شود، که معمولاً در یک بازه زمانی ثابت (مثلاً 60 بار در ثانیه) اتفاق می‌افتد. این بدان معناست که سنسور همیشه با آخرین وضعیت بازی به‌روز خواهد بود، اما ممکن است به‌روزرسانی‌های بیشتری از حد لازم دریافت کند که منجر به مشکلات بالقوه عملکرد شود.

در حالت non-up-wake یک حسگر تنها زمانی به روز می شود که به طور فعال توسط موتور بازی یا یک اسکریپت پرس و جو شود. این می‌تواند منجر به عملکرد بهتر شود، زیرا سنسور دائماً خود را به‌روزرسانی نمی‌کند، اما ممکن است منجر به قدیمی شدن سنسور نیز شود، اگر مکرراً از آن سؤال نشود.

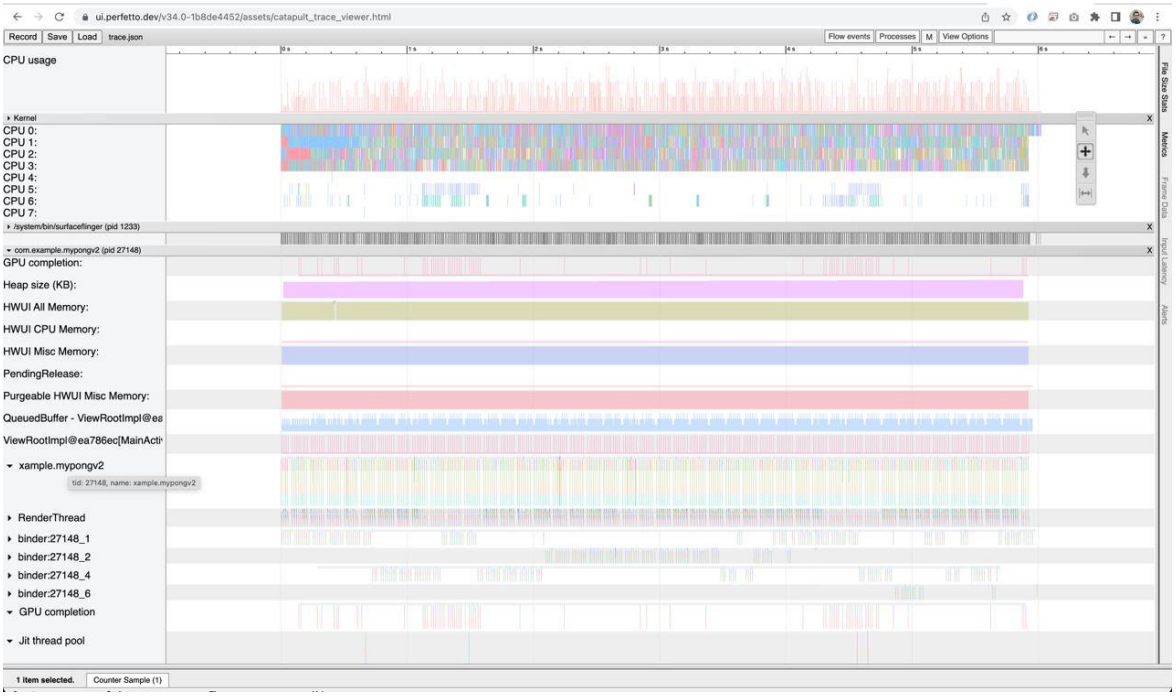
انتخاب بین این دو حالت بستگی به نیازهای خاص بازی دارد. برای بازی‌هایی مانند Pong، که در آن‌ها پاسخگویی در زمان واقعی مهم است، حالت wake-up ممکن است ترجیح داده شود، زیرا تضمین می‌کند که موقعیت راکت همیشه به‌روز و دقیق است. با این حال، برای بازی‌های پیچیده‌تر با سنسورهای زیاد که به‌روزرسانی آنها پرهزینه است، حالت non-wake-up ممکن است برای حفظ عملکرد خوب لازم باشد.

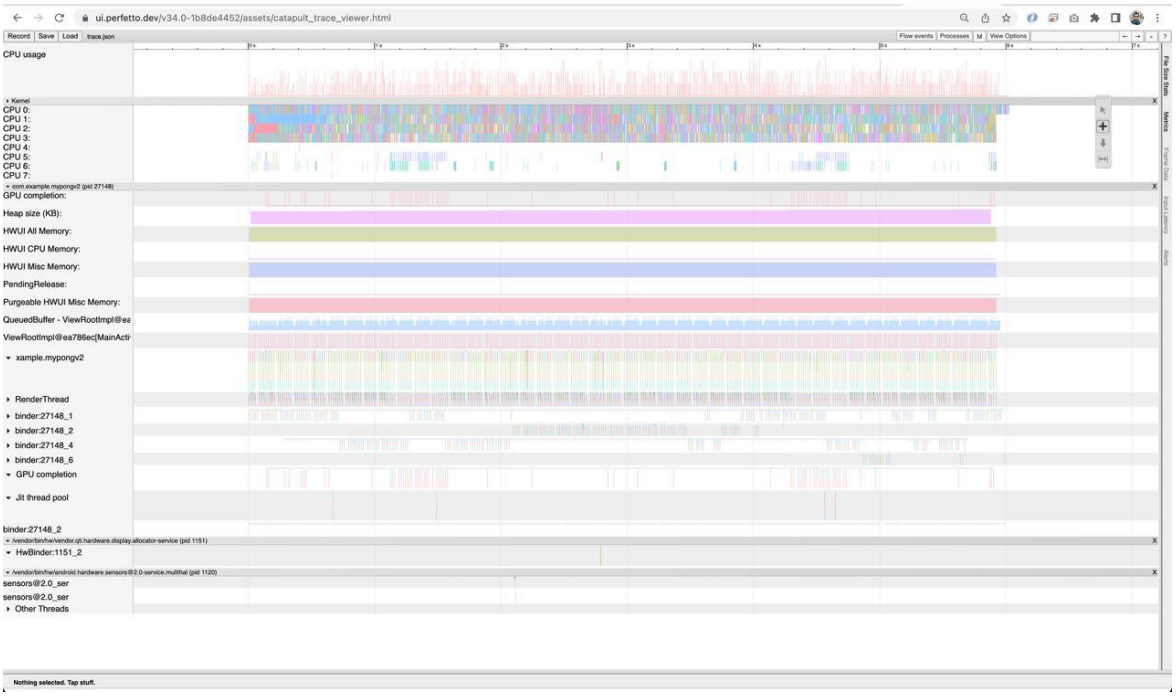
ابزار Profile and Trace




Perfetto
Search
Navigation
Open trace file
Open with legacy UI
Record new trace
Current Trace
trace_file.perfetto-trace (10 MB)
Show timeline
Download
Query (SQL)
Metrics
Info and stats
Convert trace
Switch to legacy UI
Convert to .json
Convert to .systrace
Example Traces
Open Android example
Open Chrome example
Support
Keyboard shortcuts
Documentation
Flags
Report a bug
com.example.mypongv2 27148
Jit thread pool 27164
binder:27148_1 27169
binder:27148_1 27169
binder:27148_2 27170
binder:27148_2 27170
binder:27148_4 27173
binder:27148_4 27173
binder:27148_6 27175
binder:27148_6 27175
binder:27148_2 27199
Heap size (KB)
HWUI All Memory
HWUI CPU Memory
HWUI Misc Memory
PendingRelease
Purgeable HWUI Misc Memory
surfaceflinger 1233
traced_probes 1525
system_server 1472
vendor.qti.hardware.display.compo


Info and stats
Convert trace
Switch to legacy UI
Convert to .json
Convert to .systrace
Example Traces
Open Android example
android.hardware.sensors@2.0-ser
vice.multihal 1120
sensors@2.0-ser 1992
sensors@2.0-ser 1992
sensors@2.0-ser 2438
sensors@2.0-ser 2438
11_see 29516
11_see 29516
41_see 29528
41_see 29528
271_see 31248
1651_see 31251
sensors@2.0-ser 31520
sensors@2.0-ser 31521
sensors@2.0-ser 31522


ui.perfetto.dev/v34.0-1b8de4452/assets/catapult_trace_viewer.html
Record
Save
Load
trace.json
Flow events
Processes
M
View Options
CPU usage
Kernel
CPU 0:
CPU 1:
CPU 2:
CPU 3:
CPU 4:
CPU 5:
CPU 6:
CPU 7:
/system/bin/surfaceflinger (pid 1233)
com.example.mypongv2 (pid 27148)
GPU completion:
Heap size (KB):
HWUI All Memory:
HWUI CPU Memory:
HWUI Misc Memory:
PendingRelease:
Purgeable HWUI Misc Memory:
QueuedBuffer - ViewRootImpl@ea
ViewRootImpl@ea786ec[MainActi
xample.mypongv2
tid: 27148, name: xample.mypongv2
RenderThread
binder:27148_1
binder:27148_2
binder:27148_4
binder:27148_6
GPU completion
Jit thread pool
1 item selected.
Counter Sample (1)
File Size Stats
Metrics
Frame Data
Input Latency
Alerts



شکست کار

دانیال و محمد ابتدای کار را شروع کردند و کارهایی از پروژه مانند استفاده از سنسورها، مشخص کردن کلاس ها، قسمتی از برخورد به دیواره ها و paddle، حرکت paddle به چپ و راست با چپ و راست کردن گوشی، چرخش در حول محور z و... انجام دادند. پس از مدتی کد همراه با توضیحات برای محمدحسین و کیمیا برای کارهای باقی مانده مانند بهبود برخورد بین paddle و دیوار با توپ، خارج نشدن paddle و توپ از صفحه، اضافه کردن کار امتیازی، دکمه ریست و رنگ دیواره ها و... توسط فرستاده شد.