	به نام خدا	

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

پروژه دوم درس سیستم‌های نهفته بی‌درنگ

نام و نام‌خانوادگی	**سارا رضایی‌منش-نرگس غلامی-آوا کاظمی‌نژاد-ریحانه احمدپور**
شماره دانشجویی	**810198494-810197675-810198447-810198576**
تاریخ ارسال گزارش	

نحوه شکست کار بین اعضا:

- درست کردن گرافیکی راکت و توپ و بررسی برخورد توپ و چوب صاف
- کار با شتاب سنج و جابجایی راکت
- کار با ژیروسکوپ و چرخاندن راکت
- حرکت توپ و بازخورد توپ به برخورد با راکت
- بررسی اپلیکیشن با استفاده از ابزار و پاسخ به سوالات صورت پروژه

مفروضات:

طراحی مفهومی و ساختار برنامه اندروید:

ساختار برنامه به این صورت است که هر کدام از توپ و راکت کلاس مخصوص به خود را دارند.

روند برنامه به این صورت است که بعد از مقدار دهی اولیه در کلاس پدر که همان کلاس GameView است، همواره در یک چرخه هستیم که ابتدا اجسام آپدیت می‌شود و سپس draw می‌شود. در نتیجه هر دوی این کلاس ها یک فانکشن update و یک فانکشن draw دارند که دائما در این چرخه صدا زده می‌شوند. کلاس velocity نیز برای مدیریت سرعت توپ می‌باشد.

ابزارهای مربوط و کتابخانه های مورد استفاده:

از کتابخانه canvas برای گرافیک برنامه استفاده کردیم. همچنین برای اندازه‌گیری‌های مختلف برنامه من جمله شتاب سنج و ژیروسکوب از کتابخانه‌های Sensor کمک گرفتیم.

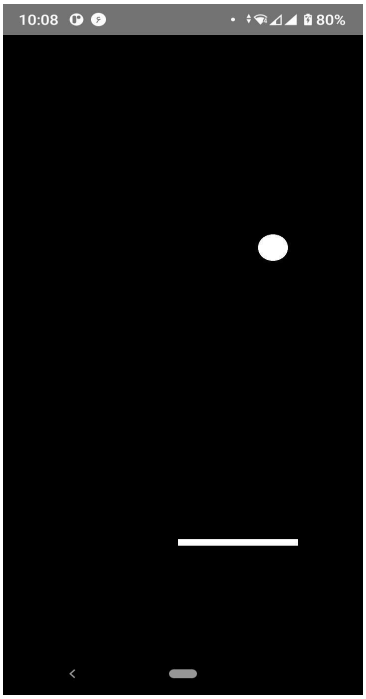
تنظیمات آزمایش ها و سناریوهای تست:

یک سری از تست‌هایی که بر روی اپلکیشن صورت گرفت توسطemulator خود اندروید استادیو بود ولی برای تست نهایی روی گوشی موبایل امتحان میشد.

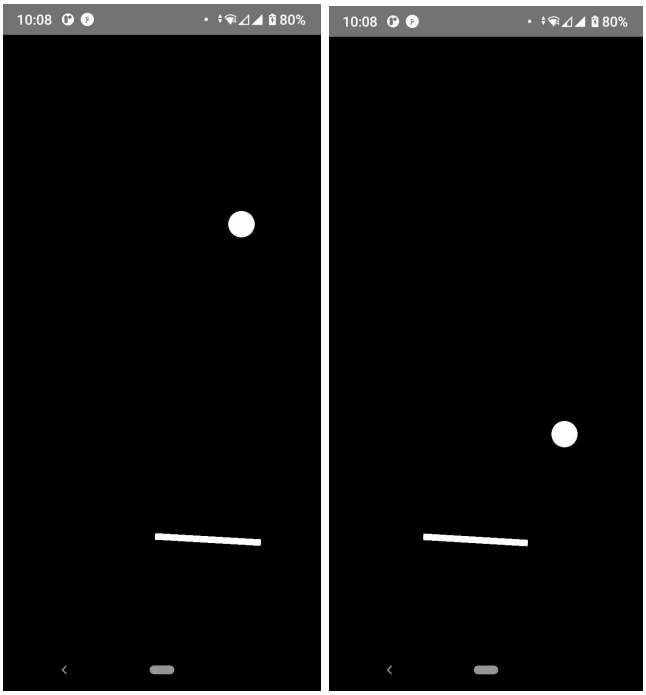
سناریوهای تست شامل گرداندن موبایل، جابجایی موبایل، در حالت سکون نگه داشتن آن بود.

تصاویر مربوط به خروجی برنامه و تصاویر Perfetto :

برنامه در لحظات اولیه، سقوط آزاد توپ:

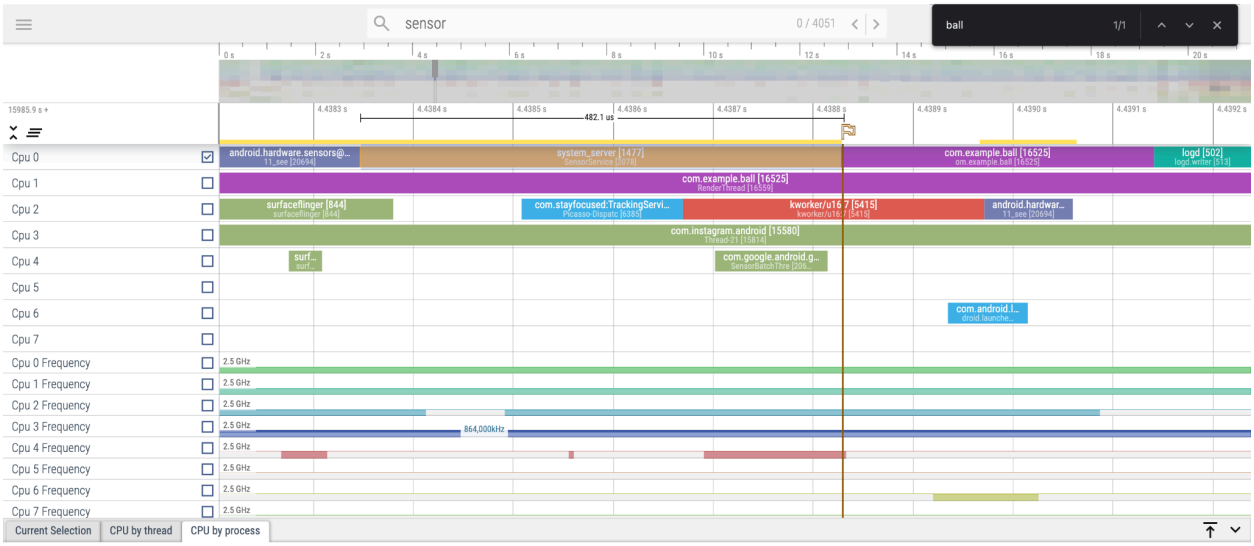


جابجایی راکت و چرخیدن راکت:



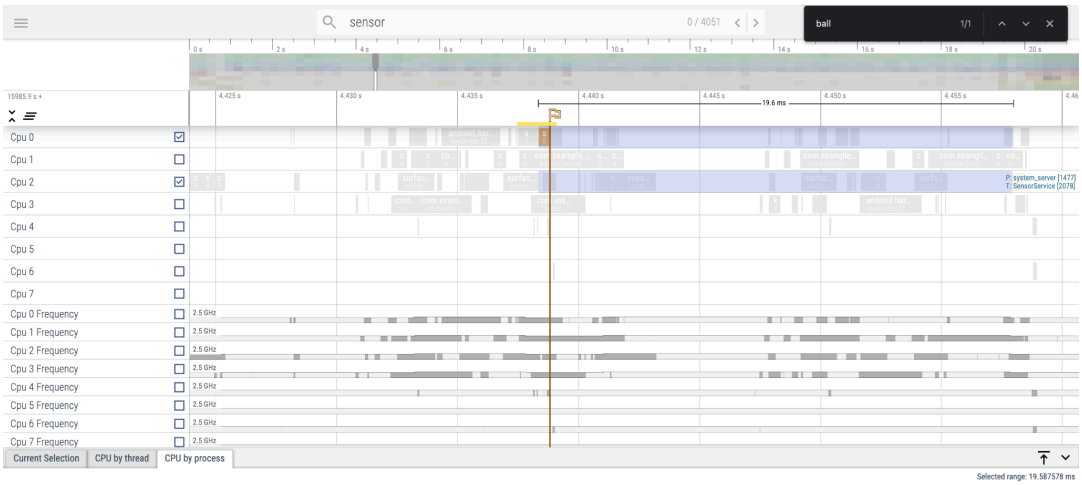
خروجی برنامه preffeto:

طول خواندن سنسور



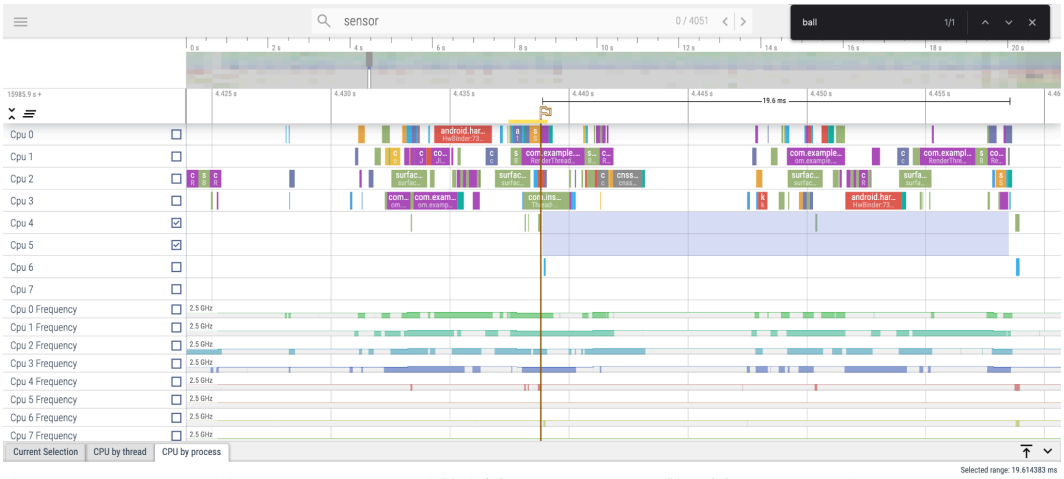
Process	PID	Wall duration (ms)	Avg Wall duration (ms)	Occurrences
		0.795		2
system_server	1477	0.481407	0.481407	1
com.example.ball	16525	0.313593	0.313593	1

زمان بین ۲ تا سنسور کال



Selected range: 19.587578 ms

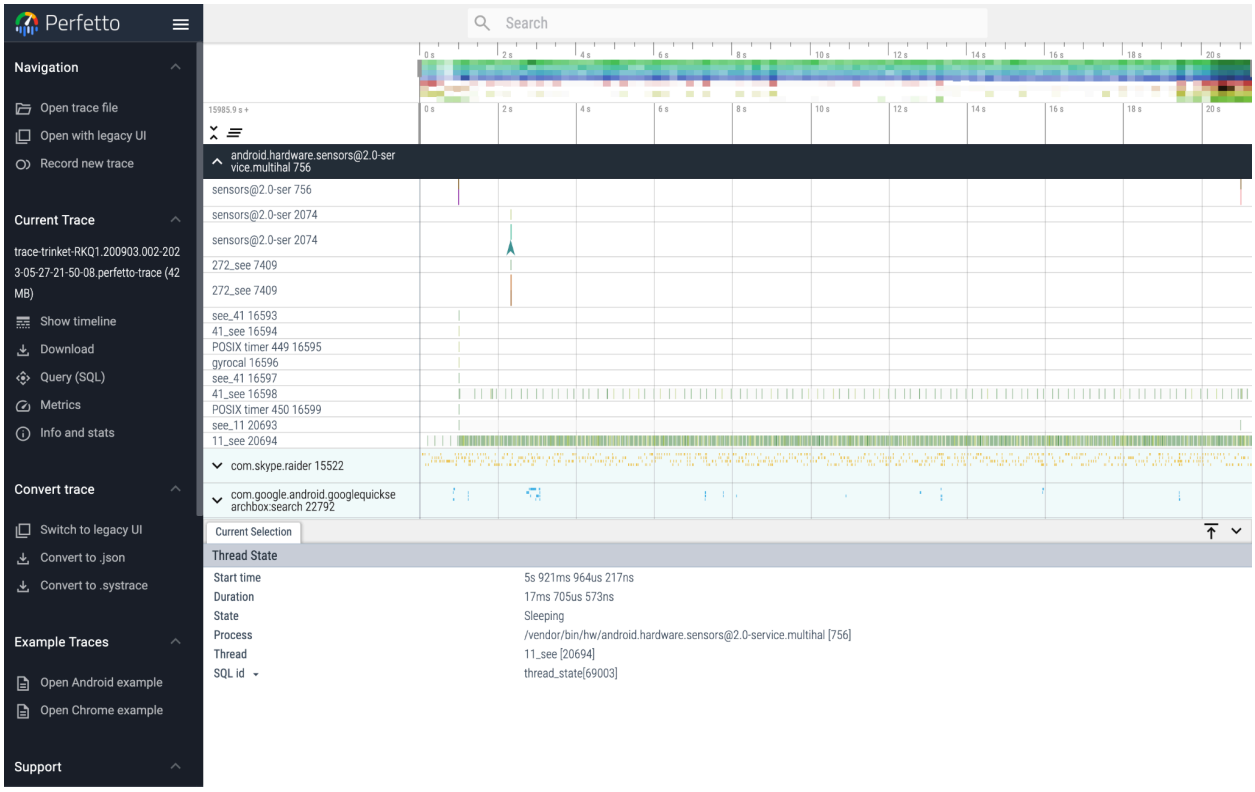
Process	PID	Wall duration (ms)	Avg Wall duration (ms)	Occurrences
		13.236929		58
/system/bin/surfaceflinger	844	5.489375	0.365958	15
/system/vendor/bin/cnss_diag	3578	1.679948	0.839974	2
com.example.ball	16525	1.526716	0.138792	11
kworker/u16:7	5415	0.643022	0.160755	4
/system/bin/logd	502	0.54021	0.18007	3
system_server	1477	0.481407	0.481407	1
/vendor/bin/hw/android.hardware.sensors@2.0-service.multihal	756	0.423802	0.211901	2
/vendor/bin/hw/android.hardware.wifi@1.0-service	762	0.31802	0.31802	1
com.android.systemui	2192	0.316563	0.158281	2

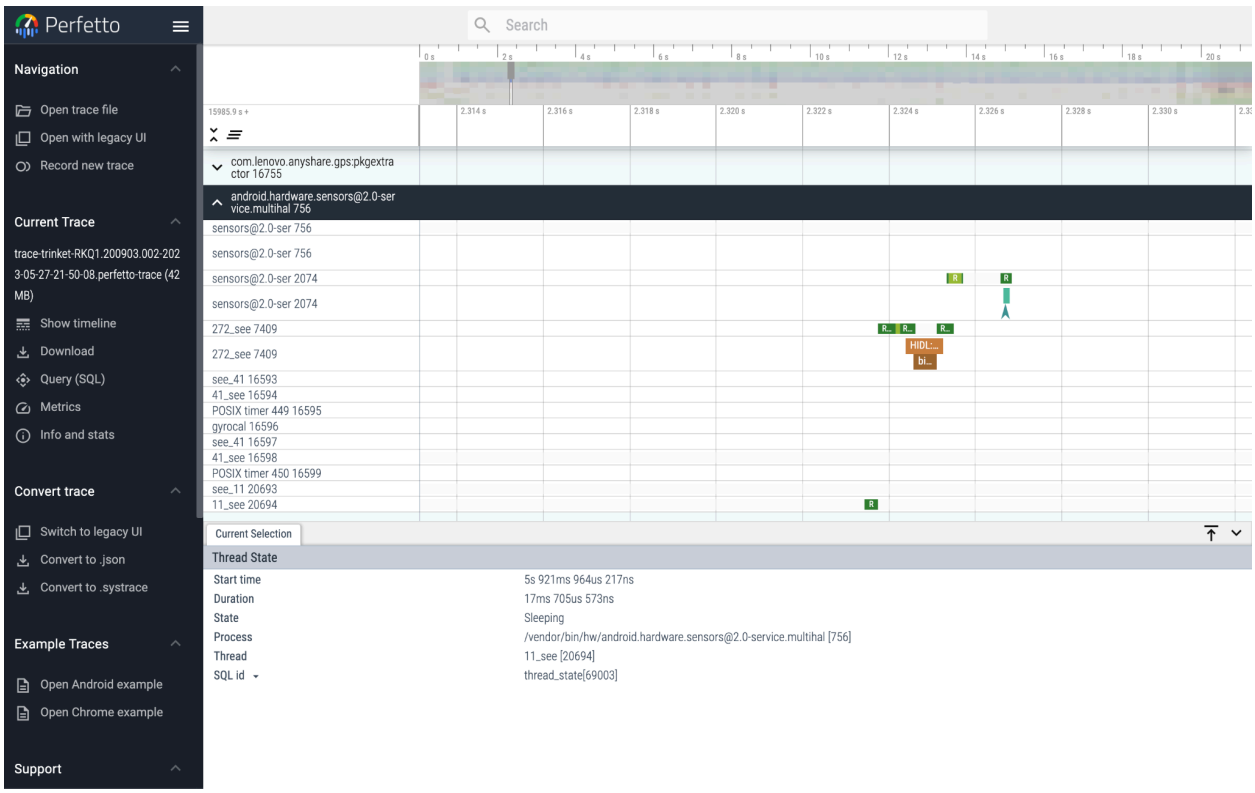


Selected range: 19.614383 ms

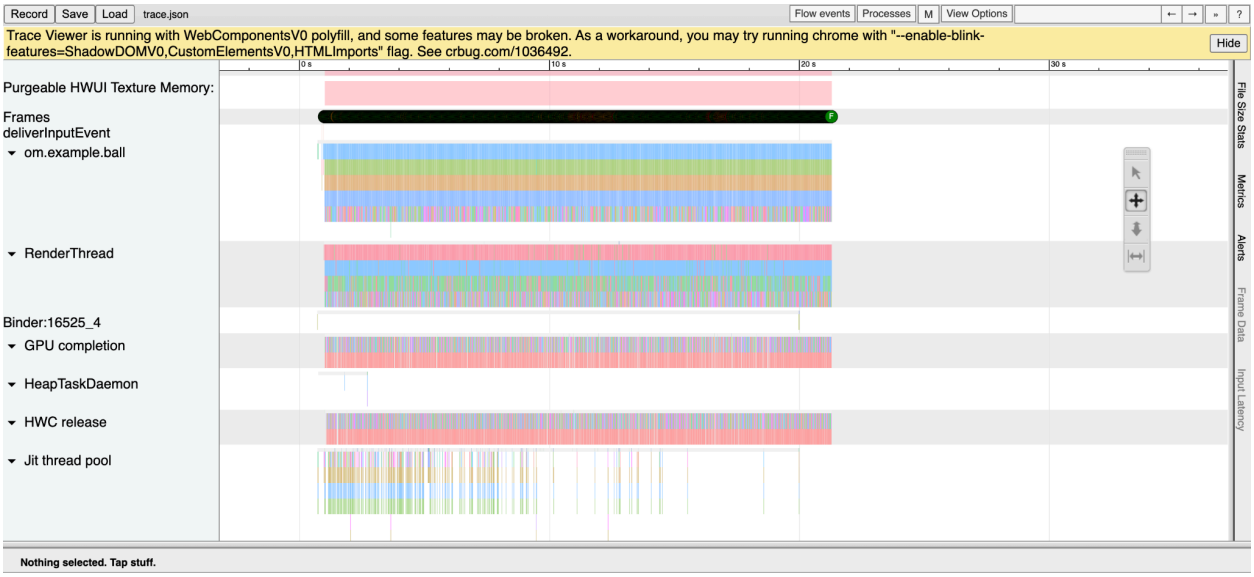
Process	PID	Wall duration (ms)	Avg Wall duration (ms)	Occurrences
		0.075834		1
/system/bin/surfaceflinger	844	0.075834	0.075834	1

سنسور های برنامه





پارت legacy


Record
Save
Load
trace.json
Flow events
Processes
M
View Options
Trace Viewer is running with WebComponentsV0 polyfill, and some features may be broken. As a workaround, you may try running chrome with "--enable-blink-features=ShadowDOMV0,CustomElementsV0,HTMLImports" flag. See crbug.com/1036492.
Hide
0 s
10 s
20 s
30 s
Purgeable HWUI Texture Memory:
Frames
deliverInputEvent
om.example.ball
RenderThread
Binder:16525_4
GPU completion
HeapTaskDaemon
HWC release
Jit thread pool
File Size Stats
Metrics
Alerts
Frame Data
Input Latency
Nothing selected. Tap stuff.


Record
Save
Load
trace.json
Flow events
Processes
M
View Options
Trace Viewer is running with WebComponentsV0 polyfill, and some features may be broken. As a workaround, you may try running chrome with "--enable-blink-features=ShadowDOMV0,CustomElementsV0,HTMLImports" flag. See crbug.com/1036492.
Hide
0 s
10 s
20 s
30 s
Interactions
Alerts
CPU usage
Kernel
CPU 0:
CPU 1:
CPU 2:
CPU 3:
CPU 4:
CPU 5:
CPU 6:
CPU 7:
/system/bin/surfaceflinger (pid 844)
com.example.ball (pid 16525)
aq:pending:com.example.ball/com
GPU completion:
Heap size (KB):
HWC release:
HWUI All Memory:
HWUI CPU Memory:
File Size Stats
Metrics
Alerts
Frame Data
Input Latency
Nothing selected. Tap stuff.

گزارش نتایج و پاسخ به سوالات مطرح شده:

1.در شکل های بالا cpu 0 را مشاهده کنید. (طول زمان خوندن سنسور در آن مشخص شده) زمانی که درخواست خواندن داده از یک سنسور داده می‌شود تا زمانی که داده‌ها دریافت شوند، چند اتفاق در سطح سیستم عامل رخ می‌دهد. در ادامه، مراحل اصلی از این فرآیند را توضیح می‌دهیم:

-ثبت نام سنسور: برنامه شما از طریق سرویس SensorManager در سیستم عامل ثبت نام می‌کند تا از سنسور مورد نظر (مثلاً ژیروسکوپ) خواندن داده را دریافت کند.

-درخواست داده: بعد از ثبت نام سنسور، برنامه شما درخواست خواندن داده را به سیستم عامل می‌دهد. این درخواست ممکن است شامل نرخ نمونه‌برداری مورد نظر و دقت داده‌ها باشد.

- تنظیمات سنسور: سیستم عامل تنظیمات مربوط به سنسور را بررسی می‌کند و سپس تنظیمات لازم را بر روی سنسور اعمال می‌کند. این شامل تنظیم نرخ نمونه‌برداری و حساسیت سنسور است.

- خواندن داده: سیستم عامل در زمان‌های مشخصی داده‌های سنسور را از سنسور دریافت می‌کند. این زمان‌ها معمولاً متناسب با نرخ نمونه‌برداری تنظیم شده است.

- ارسال داده به برنامه: پس از دریافت داده‌های سنسور، سیستم عامل آنها را به برنامه شما ارسال می‌کند. برنامه می‌تواند این داده‌ها را برای تحلیل و استفاده در عملکرد خود استفاده کند.

- تجمیع و پردازش داده: برنامه می‌تواند داده‌های دریافت شده را تجمیع و پردازش کند تا اطلاعاتی مانند زاویه چرخش، شتاب و جهت را محاسبه کند.

-بروزرسانی داده: در هر زمان که داده‌های سنسور به روز شوند، برنامه می‌تواند داده‌های جدید را دریافت و بروزرسانی کند.

دقت کنید که این فرآیند به طور کلی استاندارد است و ممکن است بسته به سیستم عامل، سخت‌افزار و رابط برنامه نویسی (API) مورد استفاده، تفاوت‌هایی داشته باشد. همچنین، عوامل دیگری مانند مصرف انرژی، پایداری سیگنال و زمان پاسخ سیستم نیز می‌تواند بر عملکرد و دقت نمونه‌برداری تأثیر بگذارد.

بله، در برخی موارد، ممکن است تعارضی بین استفاده از کتابخانه مربوط به گرافیک و بروزرسانی سنسورها وجود داشته باشد، به ویژه در صورتی که هر دو فعالیت در یک Thread انجام شوند. این ممکن است به دلیل مسدود شدن یا تأخیر در یکی از فعالیت‌ها، تأخیر در بروزرسانی سنسورها یا کاهش کارایی گرافیکی منجر شود.در برنامه‌نویسی multithread، بهتر است که فعالیت‌های مرتبط با گرافیک را در رشته رابط کاربری (UI Thread) انجام داده و فعالیت‌های مرتبط با بروزرسانی سنسورها را در رشته‌های دیگری انجام دهید. این کار ممکن است از طریق استفاده از روش‌های مانیتورینگ مانند Executor، AsyncTask، Handler یا Coroutines در Java و Kotlin انجام شود.با این روش، شما می‌توانید بروزرسانی سنسورها را در یک thread مستقل انجام داده و به طور همزمان با بروزرسانی سنسورها، فعالیت‌های گرافیکی را در رشته رابط کاربری انجام دهید. این به شما امکان می‌دهد تا تعارض بین این دو فعالیت را کاهش دهید و بهبود عملکرد و کارایی برنامه خود را تضمین کنید.بنابراین، برنامه‌نویسی چند رشته‌ای صحیح و استفاده از روش‌های مناسب برای هر فعالیت، می‌تواند به حل مشکلات تعارض بین استفاده از کتابخانه مربوط به گرافیک و بروزرسانی سنسورها کمک کند.

بیشترین بار پردازشی در پروژه ما بر روی گرفتن و پردازش داده‌های سنسور است زیرا از طرفی از دو سنسور دیتا میگیریم و آن را تحلیل و بررسی می‌کنیم و از آن طرف قسمت گرافیکی اپلیکیشن اصلا پیچیدگی خاصی ندارد.

2. در شکل های بالا cpu 0 و cpu2 را مشاهده کنید.(زمان بین ۲ تا سنسور کال که حدودا ۲۰ میلی ثانیه میشه)بهترین نرخ نمونه‌برداری معمولاً در حدود 50 تا 200 هرتز است. این محدوده نرخ نمونه‌برداری به طور کلی به عنوان یک محدوده معقول برای بسیاری از برنامه‌ها در حالت کلی قابل قبول است. در عمل، بهتر است با نرخ نمونه‌برداری بالا آزمایشات خود را آغاز کنیم و سپس در صورت لزوم نرخ را کاهش دهیم. با این روش، می‌توانیم نرخ نمونه‌برداری را براساس نیازهای خاص برنامه و قابلیت‌های سیستم تنظیم کنیم.

در این برنامه، از آنجایی که لازم است در هر لحظه به حرکات کاربر واکنش نشان دهد و محاسبات جدید انجام دهد، در صورت نمونه برداری با استفاده از تاخیر بیشتر از SENSOR_DELAY_GAME، ممکن است بسیاری از حرکات را از دست بدهیم. همچنین از آنجایی که حرکات کاربر در بازه های زمانی بسیار کوچک ممکن است دچار خطا شود و همچنین سنسور ها نیز نویز دارند، استفاده از تاخیر کمتر هم تاثیر این نویز ها و خطا ها را در محاسبات ما بیشتر می کند و باعث می شود حتی به تغییرات بسیار

جزئی در حرکات کاربر که ممکن است عمدی نبوده باشد واکنش دهیم. در این برنامه با استفاده از آزمون و خطا به دست آوردیم که SENSOR_GAME_DELAY تاخیر نمونه برداری ای است که با استفاده از آن نه ایونت های مهم را از دست می دهیم و نه به ایونت های غیرعمدی و خطاها واکنش نشان می دهیم.

در اندروید استادیو می‌توان از سرویس SensorManager استفاده کرد و نرخ نمونه‌برداری را برای سنسورها تنظیم کرد. به عنوان مثال:

```
8 usages
@Override
protected void onAttachedToWindow() {
    super.onAttachedToWindow();
    if (accelerometerSensor != null) {
        sensorManager.registerListener( listener: this, accelerometerSensor, SensorManager.SENSOR_DELAY_GAME, SensorManager.SENSOR_DELAY_GAME);
    }
    if (gyroscopeSensor != null) {
        sensorManager2.registerListener( listener: this, gyroscopeSensor, SensorManager.SENSOR_DELAY_GAME, SensorManager.SENSOR_DELAY_GAME);
    }
}
```

در برنامه خودمان به این صورت نرخ نمونه برداری ها را ست کرده‌ایم. SENSOR_DELAY_GAME تا حد خوبی دقیق است.

۳.استفاده از Android NDK به جای Android SDK ، در زیر به مزایا و معایب استفاده از آن ها اشاره می شود:

مزایا:

1- اجرای سریع‌تر: یکی از مزایای استفاده از Android NDK، این است که برنامه‌هایی که با استفاده از زبان‌های C و C++ نوشته شده‌اند، سریعتر از برنامه‌هایی هستند که با استفاده از Java نوشته شده‌اند.

2- دسترسی به کتابخانه‌های C/C++: با استفاده از Android NDK، می توانید به هر کتابخانه‌ای که برای C/C++ وجود دارد، دسترسی داشته باشید و از آن در برنامه‌های خود استفاده کنید.

3- قابلیت استفاده در انواع پلتفرم‌ها: برنامه‌هایی که با استفاده از Android NDK نوشته شده‌اند، می‌توانند بر روی انواع پلتفرم‌هایی که از C/C++ پشتیبانی می‌کنند، اجرا شوند.

معایب:

1- پیچیدگی بیشتر در توسعه: برنامه‌نویسی با استفاده از زبان‌های C و C++ پیچیده‌تر از برنامه‌نویسی با استفاده از Java است.

2- عدم سازگاری با همه دستگاه‌ها: برنامه‌هایی که با استفاده از Android NDK نوشته شده‌اند، به علت استفاده از کد native، ممکن است با برخی دستگاه‌ها سازگار نباشند.

3- افزایش حجم برنامه: به دلیل استفاده از کد native، حجم نرم‌افزارهایی که با استفاده از Android NDK نوشته شده‌اند، افزایش خواهد یافت.

در نهایت باید گفت که استفاده از Android NDK به جای Android SDK، بسته به نوع پروژه و نیازهای آن می‌تواند مزایا و معایب خود را داشته باشد، بنابراین انتخاب به عهده برنامه‌نویس است.

.۴

سنسورهای Software-based و Hardware-based

1. سنسورهای Hardware-based:

این سنسورها به صورت فیزیکی و با استفاده از قطعات الکترونیکی و حسگرها ساخته شده‌اند. این سنسورها معمولاً به صورت دستگاه جداگانه‌ای به کامپیوتر شما متصل می‌شوند و برای کاربردهای مختلفی مانند اندازه‌گیری دما، فشار، رطوبت و... استفاده می‌شوند. سنسورهای Hardware-based عموماً دقیق‌تر از سنسورهای Software-based هستند و برای کاربردهایی که دقت بالا را می‌طلبند، مناسب هستند.

2. سنسورهای Software-based:

این سنسورها برای اندازه‌گیری هایی که برایشان پردازش نیاز است، از الگوریتم های نرم‌افزاری استفاده می‌کنند. به این ترتیب ، سنسورهای Software-based قبل از تولید داده‌ها با پردازش داده‌های قبلی و الگوریتم‌های خود، دقت بالاتری را ارائه می‌دهند. این سنسورها برای کاربردهایی که نیاز به پردازش قبلی دارند مناسب هستند. به عنوان مثال، یک سنسور شتاب‌سنج در یک گوشی هوشمند یک سنسور Software-based است که با استفاده از الگوریتم‌های نرم‌افزاری، شتاب را اندازه‌گیری می‌کند.

در کل ، این دو نوع سنسور برای کاربردهای مختلفی طراحی شده‌اند و بر اساس کاربرد و نیاز شما به دقت و توانایی پردازش داده‌ها، سنسور مناسب را انتخاب کنید.

در این پروژه از دو سنسور ژیروسکوپ و Acclemetor استفاده میکنیم. ژیروسکوپ برای اندازه گیری درجه و Acclemetor برای اندازه گیری شتاب است و هر دو Hardware-based اند .

.۵

دو نوع سنسور رایج در صنعت الکترونیک هستند. تفاوت اصلی بین این دو نوع سنسور در Non-wake-up و Wake-up سنسورهای .حالت فعال شدن آن‌ها است

- سنسور wake-up: در این نوع سنسور، فقط زمانی که مقدار مشخصی از داده‌ها به دست آمده باشد یا عملیات خاصی روی دستگاه انجام شود، سنسور فعال شده و اطلاعات جدیدی را به سیستم ارسال می‌کند. این نوع سنسور به دلیل استفاده از منابع کمتری از باتری و انرژی استفاده می‌کند و مزیتی برای طول عمر باتری و دسترسی به انرژی صرفه‌جویی می‌کند.
- سنسور non-wake-up: در این نوع سنسور، به طور پیوسته داده‌ها را تولید و برای سیستم ارسال می‌کند. این نوع سنسور نیاز به منابع بیشتری از باتری و انرژی دارد، زیرا به طور پیوسته داده‌ها را تولید می‌کند.

در آینده نزدیک و با توسعه فناوری های صنعتی مانند اینترنت اشیاء (IoT) به طور گسترده ای از سنسورهای Wake-up استفاده می شود، به عنوان مثال در صنعت خودروسازی که اگر از سنسورهای non-wake-up استفاده شود، سرعت عملکرد کندی خواهد داشت.

درباره نحوه دریافت بروزرسانی سنسورها و نتیجه کنترل راکت در Motion Controlled Pong Game نیز به دقت قابل بررسی است. با توجه به اینکه بازی Pong بسیار حساس است و باید به سرعت بسیار بالا اجرا شود، استفاده از سنسورهای Wake-up به دلیل سرعت عملکرد بالاتر و انرژی صرفه جویی بیشتر توصیه می‌شود. با این حال، انتخاب نوع سنسور باید با توجه به نیازمندی‌های پروژه و شرایط محیطی انجام شود.

هندل کردن برخورد چوب و توپ:

در صورتی که توپ یا بخشی از آن (با احتساب شعاع) بین x1 و x2 (مقدار x دو سر چوب) باشد و فاصله بین مرکز دایره و سطح چوب از شعاع دایره کمتر باشد، یعنی برخورد صورت گرفته است. در این صورت با استفاده از فرمول های ذکر شده در صورت پروژه شتاب جدید در دو جهت x و y محاسبه شده و اعمال می شود.

تشخیص برخورد بین توپ و چوب:

```
if((ball.getCenterX() - ball.getRadius() >= paddle.getLeft()-50 && ball.getCenterX() + ball.getRadius() <= paddle.getRight()+50)
    && (Math.abs(ball.getCenterY() - paddle.getTop()) <= ball.getRadius() ||
       Math.abs(ball.getCenterY() - paddle.getBottom()) <= ball.getRadius()) && !lastRebound) {
     velocity.setY(velocity.getY()*-1);
   ball.percussion(velocity, gyroZ, accelerationVelocity);
   lastRebound = true;
}
else {
   lastRebound = false;
}
```

محاسبه مقادیر جدید:

بخش گرانش:

پس از اینکه gameView تشکیل می‌شود، ابتدا سازنده‌ی سرعت برای شتاب ابتدایی صدا زده می‌شود، که طبق گفته‌ی پروژه این سقوط آزاد است و تنها محور yها تغییر می‌کند و پس از برخورد بازمی‌گردد.

```
21    public class GameView extends View implements SensorEventListener {
          2 usages
22        Velocity velocity = new Velocity( x: 0,  y: 25);
```

طبق گفته‌ی پروژه ایکس در ابتدا صفر است و می‌ماند و ایگرگ است که تغییرات ارتفاع را دارد، از آنجایی که خود گرانش کشش ده‌هایی دارد و ما هم در هر timeStamp باید تقریبا یک سوم صفحه را طی کرده باشیم، بنابراین ۱۵ تا برای محرک اولیه و آزادسازی حالت سقوط به روش آزمون و خطا اضافه کردیم.

همین‌طور نقطه‌ای که توپ ابتدا در آنجا قرار دارد را هم طبق پروژه وسط صفحه و بالاترین نقطه در نظر می‌گیریم:

```
20      private final int radius = 30;
        1 usage   ryhnAp
21      public Ball() {
22          centerX = 540;
23          centerY = 5;
24      }
```

تنظیمات ابتدایی سازنده‌ها به این صورت می‌شود ، حال برای اعمال آن‌ها:

در ابتدا که سنسور شتاب را فعال می‌کنیم که چند قدم ثابت برای فعال کردن سنسورها داریم:

تعریف کنترل کننده‌ی سنسور و خود سنسور شتاب:

```
        4 usages
57      private SensorManager sensorManager;
        2 usages
58      private SensorManager sensorManager2;
59
        3 usages
60      private Sensor accelerometerSensor;
```

مقداردهی سنسورها به سیستم مورد تست:

```
75      sensorManager = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
76      sensorManager2 = (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
77      accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

تنظیم زمان‌بندی سنسورها:

```
118     protected void onAttachedToWindow() {
119         super.onAttachedToWindow();
120         if (accelerometerSensor != null) {
121             sensorManager.registerListener( listener: this, accelerometerSensor,
122                     SensorManager.SENSOR_DELAY_NORMAL, SensorManager.SENSOR_DELAY_NORMAL);
123         }
```

حال سیستم setup شده و به گرانش و مطالب مربوطه می‌پردازیم:

```
147         float deltaT = (event.timestamp - acceleratorSpan) * NS2S;
148         float acceleration = (float) Math.sqrt(rx*rx + ry*ry + rz*rz);
149         accelerationVelocity = acceleration * deltaT * 10;
150
151         if((event.values[0] < low_pass && event.values[0] > 0)
152                 || (event.values[0] < 0 && event.values[0] > -low_pass)
153                 || event.values[0] > high_pass
154                 || event.values[0] < -high_pass
155                 || lastax*ax < 0)
156             ax = 0;
157         dist = (float) (0.5*ax*Math.pow(delta_t, 2)+velox*delta_t);
158         last_x = dist;
159         velox = ax*delta_t;
160         lastax = ax;
161         acceleratorSpan = event.timestamp;
```

سه خط اول مقادیری هستند که به گرانش ربط دارن، از آنجایی که گفته شد طبق چند فریم یک سوم یک سوم حرکت کند و فریم در واقع میشه زمان میان ایونت‌ها، deltaT را بدست می‌آوریم. فقط نکته‌ی مهمی که دارد این است که این زمان سیستمی ممکن است استاندراد نباشد، به همین علت NS2S ضریب استاندارد با توجه به سیستم تعریف می‌شود.

خط ۱۴۸ به توان دوی بردارهای فضا اشاره می‌کند که مجموع اینها میشه r دو که رادیکالشان میشه شتاب و از آنجایی که شتاب در زمان میشه سرعت، بنابراین سرعت شتابی که توپ می‌خواهد استفاده کند را در اینجا داریم.

باقی بخش‌های این تصویر به گرانش ربطی ندارد و acceleratorSpan را به تایم جدید برای بازه‌ی جدید آپدیت می‌کنیم.

دو جا گرانش ماهیتش را نشان می‌دهد و باید بررسی شود هنگام عوض شدن سنسور و نیازش به آپدیت و هنگام کشیدنش و بررسی برخورد، از آنجایی که در بخش ابتدایی فقط گفته شده در ابتدا سقوط آزاد داریم و سپس با توجه به چوب فرمول‌ها برخورد اعمال می‌شود پس نیازی نیست این حالت همواره در بروز رسانی بررسی شود، زیرا تنها یک بار است بروزی ندارد، بنابراین فقط هنگام برخورد firstFallen ترو می‌شود، و دیگر نیازی نیست که گرانش سقوط داشته باشیم از این پس برخورد داریم:

```
38      public void percussion(Velocity velocity, float angle, float v){
39          float yy = (float) Math.sin(v);
40          float xx = (float) Math.cos(v);
41
42          if (!firstFallen) {
43              firstFallen = true;
44              velocity.setY(velocity.getY()*-1*ground);
45          }
46          double sinThetaOverTwo = Math.sin(2*angle);
47          double cosThetaOverTwo = Math.cos(2*angle);
48          Log.d(TAG,  msg: "#1 sinThetaOverTwo : " + sinThetaOverTwo);
49          Log.d(TAG,  msg: "#1 cosThetaOverTwo : " + cosThetaOverTwo);
50          double x_ = sinThetaOverTwo * yy;
51          x_ += cosThetaOverTwo * xx;
52          double y_ = -1*sinThetaOverTwo * xx;
53          y_ -= cosThetaOverTwo * yy;
54          Log.d(TAG,  msg: "#1 x_ : " + x_);
55          Log.d(TAG,  msg: "#1 y_ : " + y_);
56          vxNew = -1*(float) x_*10;
57          vyNew = -1*(float) y_*10;
58
59          velocity.setX(vxNew);
60          velocity.setY(vyNew);
61          centerX += velocity.getX();
62          centerY += velocity.getY();
63          Log.d(TAG,  msg: "#1 yy : " + yy);
64          Log.d(TAG,  msg: "#1 xx : " + xx);
65          Log.d(TAG,  msg: "#1 vxnew : " + vxNew);
66          Log.d(TAG,  msg: "#1 vynew : " + vyNew);
67
68      }
```

خط ۴۱ تا ۴۴ مطلب گفته شده است، برای اینکه اگر چوب در حالت نرمالش بود توپ همان بالا برود و برگردد.

خط ۳۹و۴۰ به ترتیب برای این است که سرعت را در راستای ایگرگ و ایکس بشکند و طبق فرمول به کمک خط ۴۶ و ۴۷ فرمول گفته شده در پروژه را پیاده می‌کنن:

$$V_{x2} = V_{x1}cos2\theta + V_{y1}sin2\theta$$
$$V_{y2} = -V_{x1}sin2\theta - V_{y1}cos2\theta$$

از آنجایی که گفته شده می‌توان فرمول خودمان را داشته باشیم ما برای قرینه شدن و آیینه‌ای شدن، خط ۵۶و ۵۷ در منفی ضرب می‌کنیم، سپس در خطوط بعد مقادیر جدید را برای سرعت و موقعیت ست می‌کنیم.

در بروزرسانی تنها آپدیت موقعیت داریم و اگر در این آپدیت به دیواره‌های صفحه خوردیم آیینه‌ای قرینه می‌کنیم.

برای حالت مختلف اولین برخورد می‌توانیم خط ۷۴ تا ۷۸ را به نیت تاثیر گرانش در بازگشت و دایره‌ای شدن داشته باشیم:

```
70  @       public void update(Velocity velocity, int w, int h) {
71              centerX += velocity.getX();
72              centerY += velocity.getY();
73
74              if (!firstFallen){
75                  centerX += velocity.getX();
76                  centerY += velocity.getY()*ground;
77                  velocity.setY(centerY);
78              }
79
80              if ((centerX + radius > w) || (centerX - radius < 0)) {
81                  velocity.setX(velocity.getX()*-1);
82      //            velocity.setX(vxNew);
83              }
84              if ((centerY + radius > h) || (centerY - radius < 0)) {
85                  velocity.setY(velocity.getY()*-1);
86      //            velocity.setY(vyNew);
87              }
88          }
```

بخش Accelerometer:

برای حرکت دادن چوب با حرکت افقی پدال با از accelemetor استفاده می کنیم. Sampling rate را برابر با SENSOR_DELAY_GAME قرار می دهیم که برای بازی مناسب است و بهترین نتیجه را می دهد.

با چاپ کردن مقدار شتاب در جهت x، مشخص شد که مقادیری که کمتر از 0,2 هستند معمولا نویز هستند. همچنین در صورتی که گوشی به صورت ناگهانی بایستد، مقدار شتاب ناگهان بسیار منفی یا بسیار مثبت می شود.

این مقادیر را با استفاده از یک فیلتر low pass و یک فیلتر high pass فیلتر می کنیم تا در محاسبه میزان جا به جایی چوب مشکلی ایجاد نکنند. قطعه کد زیر نحوه محاسبه میزان جا به جایی به متر را نمایش می دهد.

از آنجایی که در حرکت با سرعت تقریبا ثابت شتاب در هر دو جهت مثبت ثبت می شود، با استفاده از ژیروسکوپ جهت حرکت چوب را تشخیص می دهیم. برای اینکار از یک queue استفاده می کنیم که همیشه 10 مقدار از مقدار ثبت شده توسط ژیروسکوپ در محور z را دارد. در صورتی که مقدار مثبت ها از تعداد منفی ها در این queue بیشتر بود، چوب به سمت چپ و در غیر اینصورت به سمت راست حرکت می کند. همچنین برای تشخیص ایستادن گوشی از یک queue دیگر استفاده می کنیم که این queue تعداد 50 مقدار آخر شتاب در جهت x را نگهداری می کند و در صورتی که تعداد صفر ها از یک تعداد مشخصی بیشتر شد یعنی دستگاه ایستاده است و در نتیجه چوب هم می ایستد و جا به جایی و سرعت صفر می شوند.

میزان جا به جایی با استفاده از فرمول زیر انجام می شود:

$$x = 0.5\ a\ dt \wedge 2 + v\ dt + x0$$

$$v = a\ dt$$

قطعه کد زیر محاسبات بالا را انجام داده و با استفاده از آنها میزان جا به جایی را محاسبه می کند.

```
if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
    float alpha = 0.7f;
    float low_pass = 0.15f;
    float high_pass = 20f;

    ax = event.values[0]*2;
    if((ax < low_pass && ax > 0)
            || (ax < 0 && ax > -low_pass)
            || (ax > high_pass)
            || (ax < -high_pass))
        ax = 0;

    if(queue2.size() <= 50) {
        queue2.add(ax);
    } else {
        queue2.add(ax);
        queue2.remove();
    }

    dist = (float) (0.5*ax*Math.pow(delta_t, 2)+velox*delta_t);
    velox = 0.5f;
    if(ax <= 0) {
        dist = 0;
    }
    if(queue.stream().filter(element -> element > 0).count() > queue.stream().filter(element -> element < 0).count()) {
        dist = -dist;
    }
    if(queue2.stream().filter(element -> element <= 0).count() >= 45) {
        dist = 0;
    }
}
```

در کلاس paddle مقدار بدست آمده در این بخش را بر 0.50 تقسیم می کنیم و نسبت مسافت طی شده را به 50 سانتی متر بدست می آوریم. سپس این نسبت را در عرض صفحه موبایل ضرب می کنیم و به این ترتیب میزان جا به جایی به پیکسل به دست می آید. در صورتی که چوب از سمت چپ صفحه بیرون رفته باشد و مقدار جا به جایی هم کوچکتر از صفر باشد و یا چوب از سمت راست صفحه بیرون رفته باشد و جا به جایی مثبت باشد، جا به جایی را اعمال نمی کنیم. قطعه کد زیر، محاسبات ذکر شده را انجام داده و بر روی موقعیت چوب اعمال می کند.

بخش ژیروسکوپ:

از سنسور ژیروسکوپ برای پیدا کردن زاویه چرخش چوب استفاده می‌کنیم. این سنسور سرعت زاویه‌ای را به ما می‌دهد و ما با استفاده از تایم استمپ اختلاف زمانی را محاسبه می‌کنیم و سپس ضربدر مقدار متغیری که ژیروسکوپ به ما می‌دهد می‌کنیم. برای آن یک

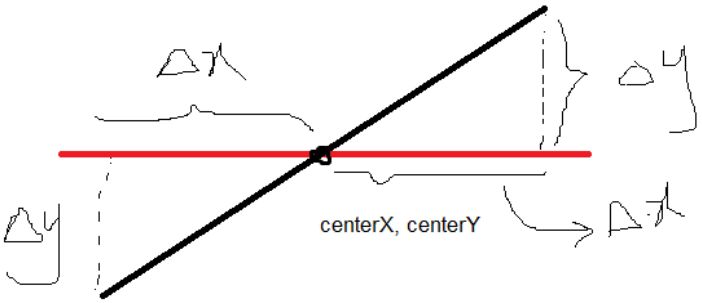
فیلتر تعیین می‌کنیم که اگر کمتر از یک مقداری شد نویز ژیروسکوپ را بگیرد. سپس این مقادیر را به عنوان ورودی به قسمت آپدیت می‌دهیم.

```
if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
    if (timestamp != 0) {
        final float dT = (event.timestamp - timestamp) * NS2S;
        rx = event.values[0];
        ry = event.values[1];
        rz = event.values[2];

        gyroX = (float) (rx*dT);
        gyroY = (float) (ry*dT);
        gyroZ = (float) (rz*dT);

        if (Math.abs(gyroZ) > EPSILON) {
            double sinThetaOverTwo = Math.sin(gyroZ);
            sin = (float) sinThetaOverTwo;
            double cosThetaOverTwo = Math.cos(gyroZ);
            cos = (float) cosThetaOverTwo;
        }
        else {
            gyroZ = 0;
            sin = 0;
            cos = 0;
        }
    }
    timestamp = event.timestamp;
}
```

سپس بعد از این که این زاویه را به رادیان پیدا کردیم سینوس و کسینوس آن را محاسبه می‌کنیم. با استفاده از فرمول‌های زیر می‌توان میزان جابجایی چوب را محاسبه کرد و این جابجایی را اعمال کرد.




```
delta_x = cos*sideX/2;
delta_y = sin*sideX/2;

float centerX, centerY;
centerX = (x1+x2)/2;
centerY = (y1+y2)/2;

if (gyroZ != 0) {
    x1 = centerX - delta_x;
    y1 = centerY - delta_y;
    x2 = centerX + delta_x;
    y2 = centerY + delta_y;
}
```

برای x1 از آنجایی که نهایت تغییرات زاویه‌ای +۹۰ تا -۹۰ است و چرخش در این زوایا صورت می‌گیرد، برای انعطاف پذیری می‌گوییم که همواره چرخش نسبت به نقطه مرکزی چوبمان باشد، و کلا از سمت چپ بالا تصویر شکل می‌گیرد و مبدا آنجاست به همین علت چپ رفتن و با رفتن یعنی کم شدن و غیر آن عکس است. به همین علت x1, y1 کم می‌شوند و دو متغیر دیگر خط اضافه می‌شوند.