دانشگاه تهران
دانشکده‌ی مهندسی برق و کامپیوتر

گزارش نهایی پروژه سیستم‌های سایبر-فیزیکی

بازی Runner مبتنی بر سنسورهای android

اعضای گروه:
سید پارسا حسینی نژاد
شکیبا بلبلیان خواه
سارا توکلی
غزل نیسی مینایی

شماره دانشجویی:
۸۱۰۱۹۶۶۰۴
۸۱۰۱۹۶۴۲۶
۸۱۰۱۹۶۶۸۴
۸۱۰۱۹۶۶۸۳

اساتید:
دکتر مهدی کارگهی، دکتر مهدی مدرسی

ترم بهار ۱۴۰۰

فهرست مطالب

1- مقدمه

- **محدوده پروژه**

تلفن‌های هوشمند دارای سنسورهای متفاوتی هستند که جهت رفع نیازهای متفاوت، مورد کاربرد قرار می‌گیرند. با این وجود، بسیاری از بازی‌های مبتنی بر اندروید تنها بر سنسور لمسی تلفن همراه و گرافیک تاکید دارند و از این سنسورها استفاده‌ای نمی‌کنند. حال آنکه سنسورها می‌توانند نقش‌های قابل توجه و هیجان‌انگیزی را در بازی‌ها ایفا کنند. در این پروژه، قصد داریم این ایده را عملی سازیم.

- **اهداف پروژه**

هدف از انجام این پروژه طراحی یک بازی مبتنی بر اندروید برای سرگرمی و وادار کردن کاربر به انجام فعالیت‌های مختصر فیزیکی در حین بازی برای سرگرمی بیشتر و متمرکز کردن کاربر بر روی کنترل حرکات فیزیکی خود می‌باشد.

2- معرفی پلتفرم و ابزارهای استفاده شده در پروژه

در این پروژه از ابزار یونیتی و زبان C# برای توسعه بازی مبتنی بر اندروید استفاده کردیم. ابزار Unity3D مورد استفاده نسخه individual بود که برای توسعه و ساخت بازی اندروید نیاز به SDK و JDK و NDK داشت. این موتور بازی سازی در کنار زبان C# ابزار اصلی مورد استفاده در این پروژه بودند.

ابزار دیگر که در ابتدا برای شبیه سازی و بررسی سنسورها استفاده کردیم، ابزار Android Studio بود. این نرم افزار برای نوشتن برنامه‌های مبتنی بر اندروید بسیار کاربردی است و برای درک بهتر عملکرد سنسورهای گوشی همراه از آن استفاده کردیم.

3- بیان چالش ها

در این پروژه به چالش‌های گوناگونی اعم از نصب و راه اندازی محیط توسعه و مشکلات فنی برخوردیم که در ادامه بررسی می‌کنیم:

- نصب و راه اندازی محیط توسعه برای بازی اندروید: ابزار یونیتی برای اندروید نیاز به پکیج‌های مختلف و SDK و JDK مناسب دارد. به دلیل تحریم بودن ایران برای دانلود و نصب این پکیج‌ها چون مجبوریم بودیم از VPN

استفاده کنیم به مشکلاتی از جمله کند بودن اینترنت و قطعی اینترنت برخوردیم. همچنین پیدا کردن پکیج‌های متناسب با نسخه Unity نیز با دشواری همراه بود.

- اتصال به سنسورها: برای شروع، همانطور که آموخته بودیم با استفاده از سنسور ژیروسکوپ در محیط اندروید استودیو جهت حرکت را تشخیص دادیم. اما در ادامه متوجه شدیم در بازی‌ها برای تشخیص حرکت گوشی در حرکات دو بعدی به سمت چپ و راست بهتر است از سنسور accelerator استفاده شود و سنسور ژیروسکوپ برای تشخیص چرخش مناسب است.
- استفاده از سنسور صدا: برای این کار باید طوری عمل می‌کردیم که فقط تغییر صدا در نظر گرفته شود، نه همه نویزها. برای حل این مشکل peak ها را در صدای ورودی در نظر گرفتیم.
- هماهنگی نسخه PC و گوشی: محیط توسعه این بازی PC است اما در زمان اجرا باید روی اندروید اجرا شود. برای این کار تعدادی تابع داریم که حرکت دست را با کلیدهای کیبورد شبیه سازی کردیم. همچنین به دلیل کیفیت و حساسیت متفاوت میکروفون گوشی و PC و همچنین وجود صدای فن در pc که خود نوعی ورودی صدا محسوب می‌شود، تست صدا روی PC ممکن است دقت کافی نداشته باشد.

4- معرفی نزدیکترین نمونه‌های مشابه

- بازی subway surfers: یکی از معروف ترین بازی های runner در حال حاضر این بازی است. این بازی با سنسور نیست اما مکانیک پریدن و دویدن و تغییر سرعت را دارد.
- بازی Car Runner - mobile game: این بازی با swipe چپ و راست میرود و می‌پرد.
- بازی Doodle jump: این بازی با سنسور accelerator گوشی به راست و چپ می‌رود.

5- مبانی فنی پروژه

o ارائه راه‌حل پیشنهادی بصورت کلی

در این پروژه به طراحی یک زمین بازی شامل مسیری باریک با موانع متعدد پرداختیم که کاراکتری در زمین بازی باید موانع را جهت رسیدن به انتهای راه پشت سر بگذارد و کاربر وظیفه‌ی هدایت او را بر عهده دارد. مراحل مختلف شامل تسک‌های متفاوت هستند که شامل موارد زیر می‌باشد:

- کاراکتر در ابتدا ثابت است و کاربر باید با زدن ضربه‌های متعدد روی صفحه‌ی گوشی (سنسور Touch تلفن همراه)، کاراکتر را وادار به حرکت کند. هر چه فرکانس ضربه روی صفحه بیشتر باشد، کاراکتر با سرعت بیشتری حرکت خواهد کرد. با رد کردن هر مانع کاربر امتیاز دریافت می‌کند.
- این حالت مشابه حالت پیش می‌باشد با این تفاوت که سرعت کاراکتر ثابت است و با داد زدن (سنسور صدا) می‌پرد.

- **ارائه راه‌حل با جزییات**

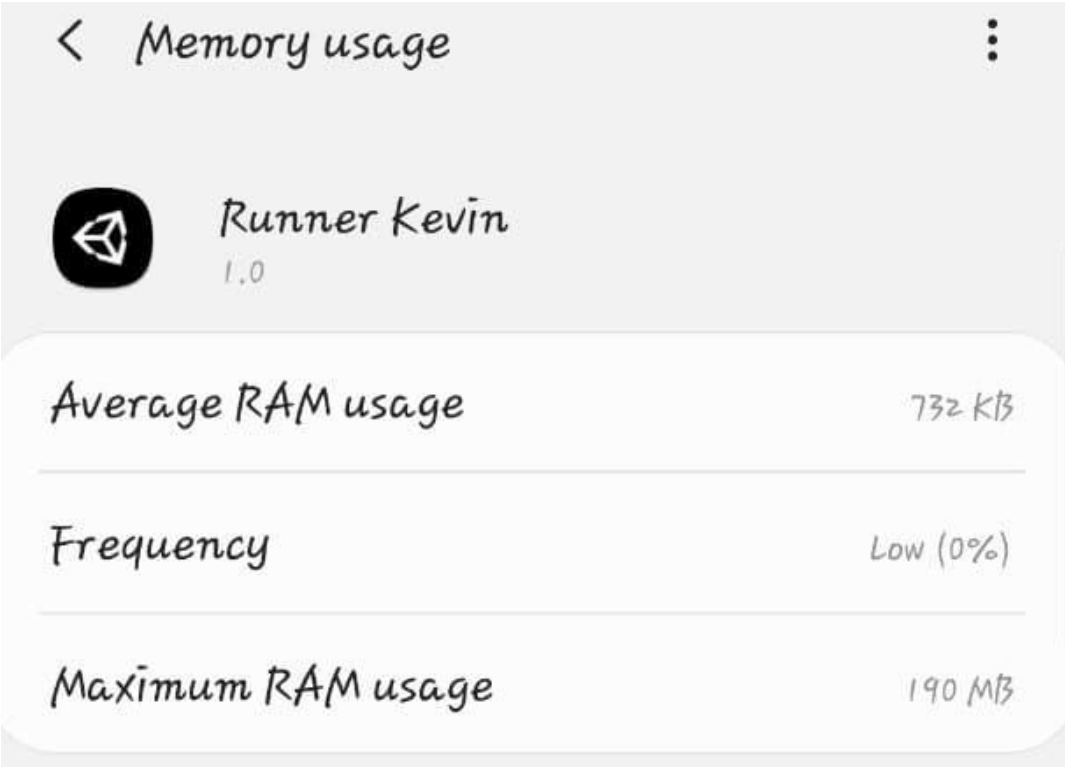
برای ساخت این بازی یک زمین و موجود ثابت (کاراکتر) در نظر گرفتیم. دوربین (جهت دید کاربر) پشت سر کاراکتر است. موانع (دیوارهای آجری) از یک نقطه دوری ایجاد می‌شوند و با سرعت به سمت کاراکتر میایند. با استفاده از سنسور accelerator حرکت گوشی (دست کاربر) تشخیص داده می‌شود و کاراکتر به چپ و راست می‌رود که از موانع فرار کند.

با استفاده نقطه فعلی کاراکتر مکان آن را روی محور x تشخیص می‌دهیم و با استفاده از عرض زمین مانع از افتادن کاراکتر می‌شویم.

با استفاده از سنسور صدا شدت صدا را تشخیص می‌دهیم و آن را تبدیل به عمل پریدن می‌کنیم. پریدن فقط زمانی می‌تواند اتفاق بیفتد که کاراکتر روی زمین باشد. با استفاده از مکان فعلی کاراکتر رو زمین بودن یا نبودن را تشخیص می‌دهیم.

- **نحوه ی تحلیل راه‌حل و اثبات کارایی**

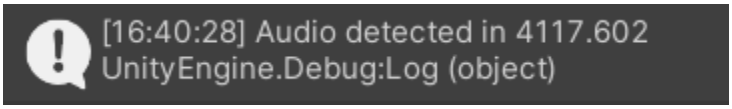
میانگین مصرف حافظه را از developer options گوشی به دست میاوریم. همانطور که مشاهده می‌شود، میزان حافظه بسیار بهینه است.



تصویر ۱ - مصرف حافظه

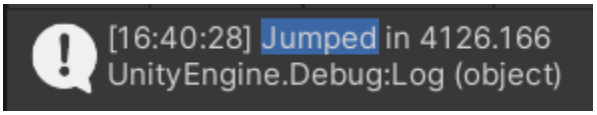
زمان تشخیص توسط سنسور و پریدن کاراکتر را تحلیل می‌کنیم:

زمان تشخیص صدا به میلی ثانیه:



تصویر ۲ - زمان تشخیص صدا

زمان پریدن به میلی ثانیه:



تصویر ۳ - زمان پریدن

تاخیر از دریافت صدا تا پریدن حدودا ۸.۵ میلی ثانیه است که کمتر از زمانی است که کاربر بتواند تشخیص دهد.

6- پیاده‌سازی‌های انجام شده

o شکست کار بین اعضای تیم

شکیبا بلبلیان خواه: توسعه بدنه اصلی اپلیکیشن

سید پارسا حسینی نژاد: توسعه اتصال به سنسور صدا

سارا توکلی: توسعه اتصال به سنسور تشخیص حرکت

غزل نیسی مینایی: اتصال ماژول‌ها به یکدیگر و تهیه مستندات

o مشخصات محیط توسعه

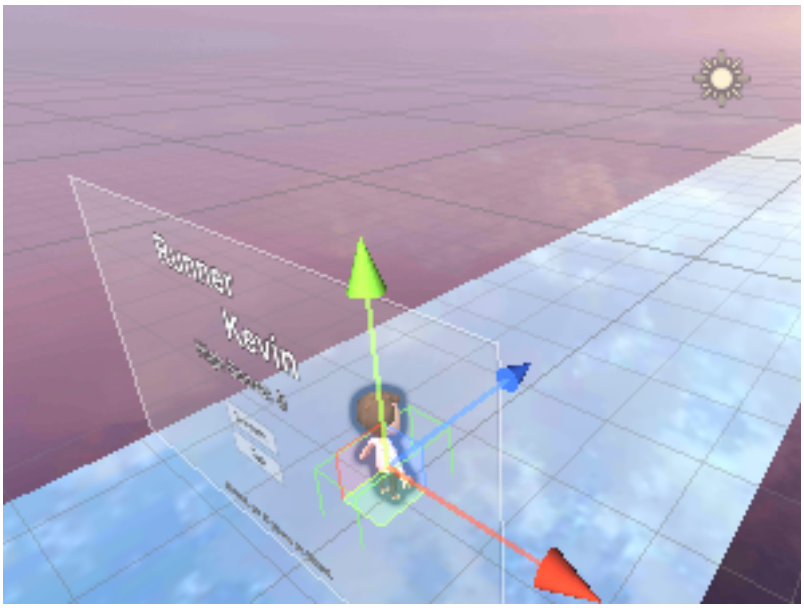
- نرم افزار Unity 3D Individual نسخه 2020.3.12f1
- سیستم عامل Ubuntu و یا Windows 10
- ابزار SDK
- ابزار JDK
- ابزار NDK

o تشریح پیاده‌سازی

تصویر ۳ - محیط بازی

این پروژه از چندین بخش که هرکدام attribute ها و اسکریپت خود را دارد تشکیل شده است که در ادامه هرکدام را با جزییات بررسی می‌کنیم:

- دوربین اصلی (Main Camera):



تصویر ۴ - مکان ثابت کاراکتر و دوربین

این دوربین همواره پشت سر کاراکتر اصلی قرار دارد و همراه با آن جابجا می‌شود.

این دوربین اول کار در نقطه شروع کاراکتر قرار دارد:

```
transform.position = new Vector3(0,2.0f,-5.0f);
```

و برای اینکه موقع پریدن جهت مناسب بماند:

```
transform.position =
new Vector3(0, 2.0f, player.position.z -5.0f);
```

- نور (Directional Light):

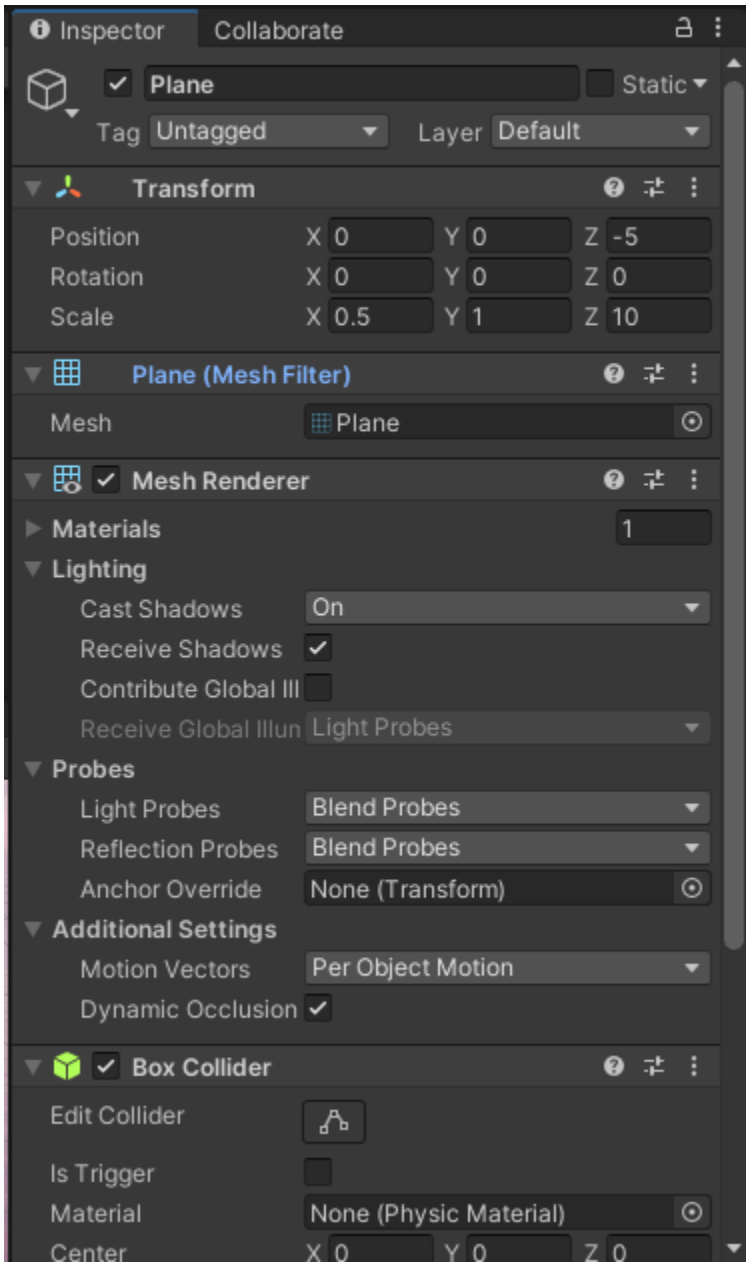
نور محیط که از بالا می‌تابد و مسئول ایجاد سایه است.

- زمین (Plane):

زمین یک plane طولانی با عرض مشخص و ثابت است که موانع بر روی آن حرکت می‌کنند و ارتفاع کاراکتر نیز بر اساس آن تعیین می‌گردد و همچنین جهت جاذبه را مشخص می‌کند.

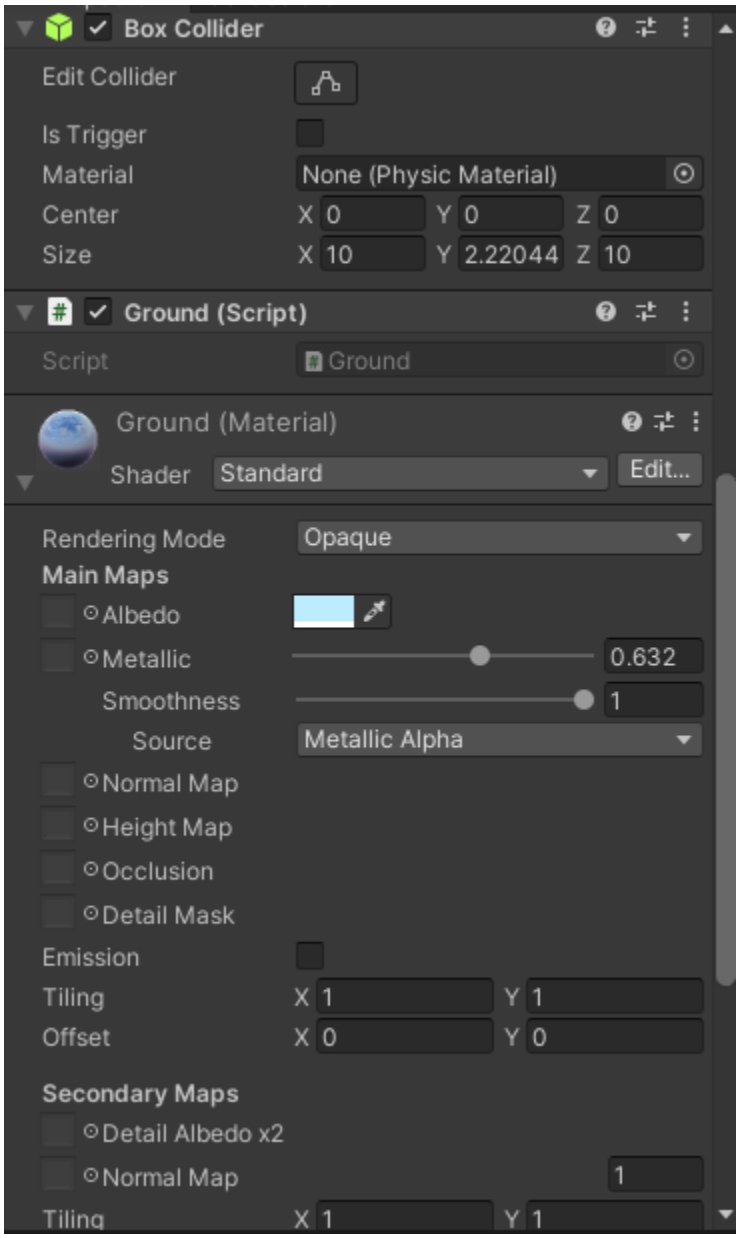
ویژگی های مهم:

برای اینکه بازیکن از زمین رد نشود، از box collider استفاده می‌کنیم و Is trigger را false می‌کنیم:



تصویر ۵ - ویژگی های plane - برخورد

و متریال استفاده شده در زمین:



تصویر ۶ - ویژگی های plane - متریال

- نقطه تولید موانع (Spawn Point):

نقطه تولید موانع، نقطه‌ای دور نسبت به کاراکتر است که مبدا تولید موانع به حساب می‌آید. موانع (دیوارها) از آنجا در مکان‌های تصادفی با نرخ مشخص و ثابت تولید (spawn) می‌شوند و به سمت کاراکتر حرکت می‌کنند. نرخ تولید موانع در هر دو mode ثابت است اما با یکدیگر فرق دارد.

- مدیر بازی (Game Manager):

مدیر بازی، ماژول اصلی بازی است و وظیفه مدیریت کارهای مختلف از جمله نمایش نوشته‌ها روی صفحه، کنترل نرخ تولید دشمنان و امتیاز را برعهده دارد.

نمایش امتیاز: حالت ۱- به معنای عدم شروع بازی است و در این حالت high score نمایش داده می‌شود:

```
void Start()
    {
     mode = -1;
          if(PlayerPrefs.HasKey("highScore"))
          {
               highScore =
PlayerPrefs.GetInt("highScore");
               highScoreText.text = "High Score: " +
highScore.ToString();
          }
    }
```

وقتی بازی شروع شده باشد باید منوی اصلی مخفی شود و امتیاز لحظه‌ای جای آن را بگیرد. همچنین تولید موانع شروع می‌گردد.

```
   void Update()
    {
     if (mode != -1 && !gameStarted)
         {
          menuPanel.gameObject.SetActive(false);
          scoreText.gameObject.SetActive(true);
          StartCoroutine("SpawnEnemies");
          gameStarted = true;

         }
    }
```

تولید موانع:

با استفاده از یک روتین، دشمنان را با نرخ ثابتی ایجاد می‌کنیم، این نرخ وابسته به mode بازی است. در mode دوم برای آنکه کاربر با چالش بیشتری روبرو شود، موانع با سرعت ابتدایی بسیار کم و فاصله زمانی کمی بیشتر از حالت اول تولید می‌شوند. بدین ترتیب اگر کاربر با ضربه (tap) زدن‌های متعدد بر صفحه به کاراکتر سرعت ندهد، مجبور است از میان موانع بسیار نزدیک به هم عبور کند. حال آنکه اگر سرعت موانع زیاد شود، با توجه به نرخ تولیدشان، فاصله بیشتری از هم خواهند گرفت و بازی برای کاربر راحت‌تر خواهد شد:

```
IEnumerator SpawnEnemies()
    {
    while(true)
    {
        float delay = 1.0f;
        if(mode == 1){
            delay = 0.8f;
        }
        else if(mode ==2){
            delay = 1.5f;
        }
            Debug.Log(delay);
        yield return new WaitForSeconds(delay);
        Spawn();
        }
    }
```

روتین بالا تابع spawn را صدا می‌زند. این تابع یک مانع به صورت تصادفی ایجاد می‌کند.

```
public void Spawn()
    {
    float randomSpawnX = Random.Range(-maxSpawnPointX,
maxSpawnPointX);
    Vector3 enemySpawnPos = spawnPoint.position;
```

```
    enemySpawnPos.x = randomSpawnX;

    Instantiate(enemy, enemySpawnPos,
Quaternion.identity);
    }
```

وقتی بازی از اول می‌شود باید امتیاز جدید در صورت بالاتر بودن از high score جایگزین آن شود:

```
public void Restart()
    {
    if(score > highScore)
    {
        highScore = score;
        PlayerPrefs.SetInt("highScore", highScore);
    }

    SceneManager.LoadScene(0);
    }
```

به ازای هر مانعی که بدون برخورد رد شود امتیاز فعلی یکی زیاد می‌شود:

```
public void ScoreUp()
    {
    score++;
    scoreText.text = score.ToString();
    }
```

بازی دارای دو mode اصلی است، حالت پرش با صدا و حالت حرکت با ضربه، این mode ابتدای کار توسط بازیکن انتخاب می‌شود و ذخیره می‌شود و در منطق بازی تاثیر می‌گذارد:

```
public void LoadMode(int modeNum){
    if(modeNum == 1)
    {
        mode = 1;
        Debug.Log(mode);
```

```
        }
        if(modeNum == 2)
        {
            mode = 2;
            Debug.Log(mode);
        }
    }
```

- مانع (Enemy):

یکی از بخش های مهم بازی موانع (دشمنان) هستند. این موانع به شکل دیوارهای کوچک هستند و همانطور که بالاتر توضیح داده شد با نرخ ثابتی تولید می‌شوند و به سمت کاراکتر ثابت حرکت می‌کنند. این کار باعث می‌شود کاربر حس کند خود در حال حرکت به سمت موانع است.

در حالت ۱ (حالتی که کاراکتر با صدا می‌پرد) سرعت ثابت است و در حالت ۲ سرعت کاراکتر وابسته به سرعت ضربه به صفحه است (touchCount). متغیر click که کلیک ماوس را در نظر می‌گیرد، برای راحت تر دیباگ کردن بازی بر روی PC مورد استفاده در نظر گرفته شده و معادل touch است. تابع `Input.touchConut` تعداد برخوردهای همزمان با صفحه گوشی را برمی‌گرداند که شرط `touchConut > 0` به معنای حداقل یک ضربه است. وقتی برخوردی نباشد، سرعت رفته رفته کم می‌شود و وقتی برخورد باشد کم کم زیاد می‌شود.

```
if(mode == -1){
    mode = GameManager.instance.mode;
}
else if(mode == 1){
    speed = maxSpeed;
}
else if(mode == 2){
    int touchCount = Input.touchCount;
    bool click = Input.GetMouseButton(0);

    if( (touchCount > 0 || click) && speed > maxSpeed){
        speed -= 1.0f;
```

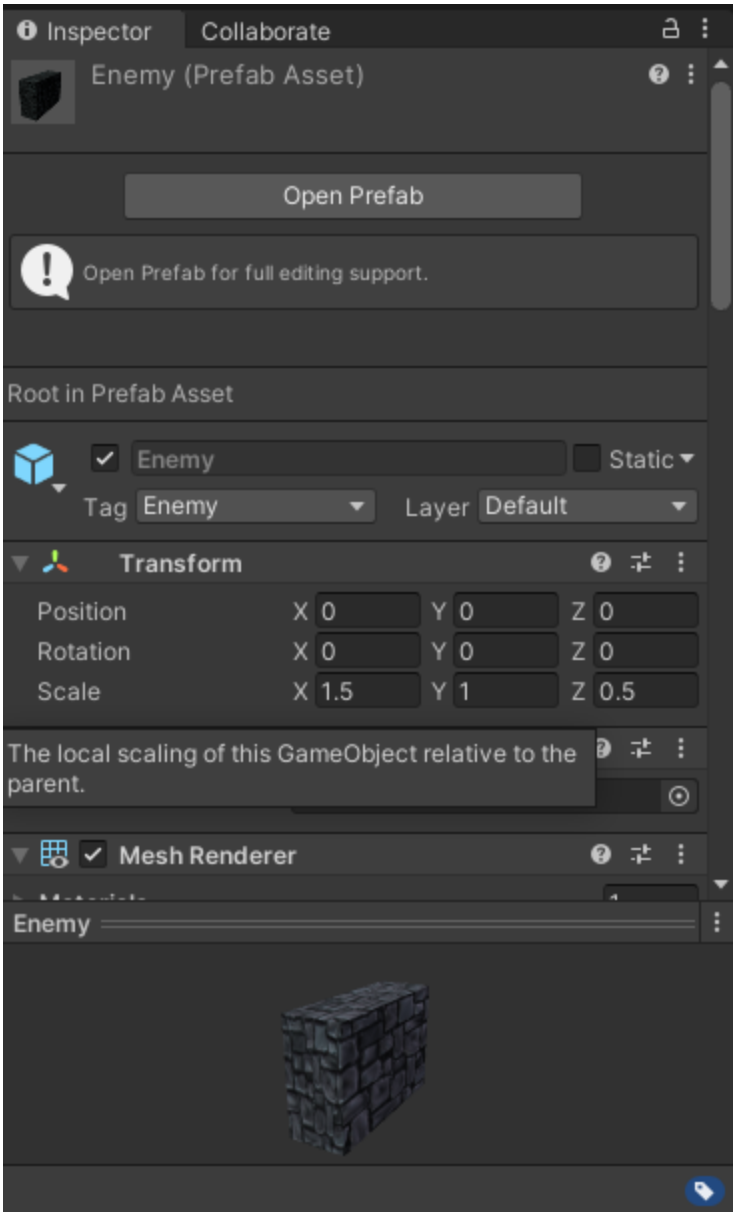
```
    }
    else if( (touchCount == 0 || !click) && speed < -2)
        speed += 0.5f;
}
transform.Translate(0, 0, speed * Time.deltaTime);
```

وقتی مانعی بدون برخورد از کاراکتر بگذرد، امتیاز کاربر باید زیاد شود و همچنین از آنجا که دیگر نیازی به مانع رد شده نداریم باید پاک شود:

```
if(transform.position.z < -10f)
{
    GameManager.instance.ScoreUp();
    Destroy(gameObject);
}
```

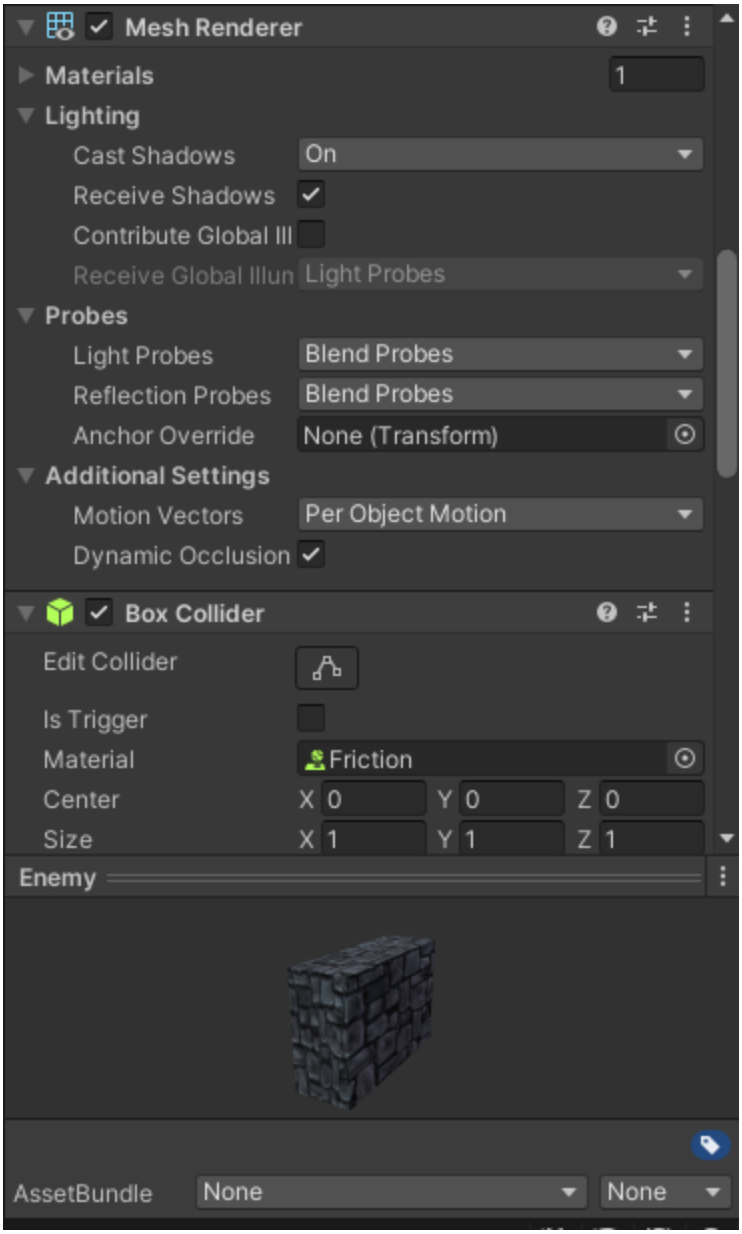
ویژگی های مهم موانع:

سایز و ظاهر موانع:



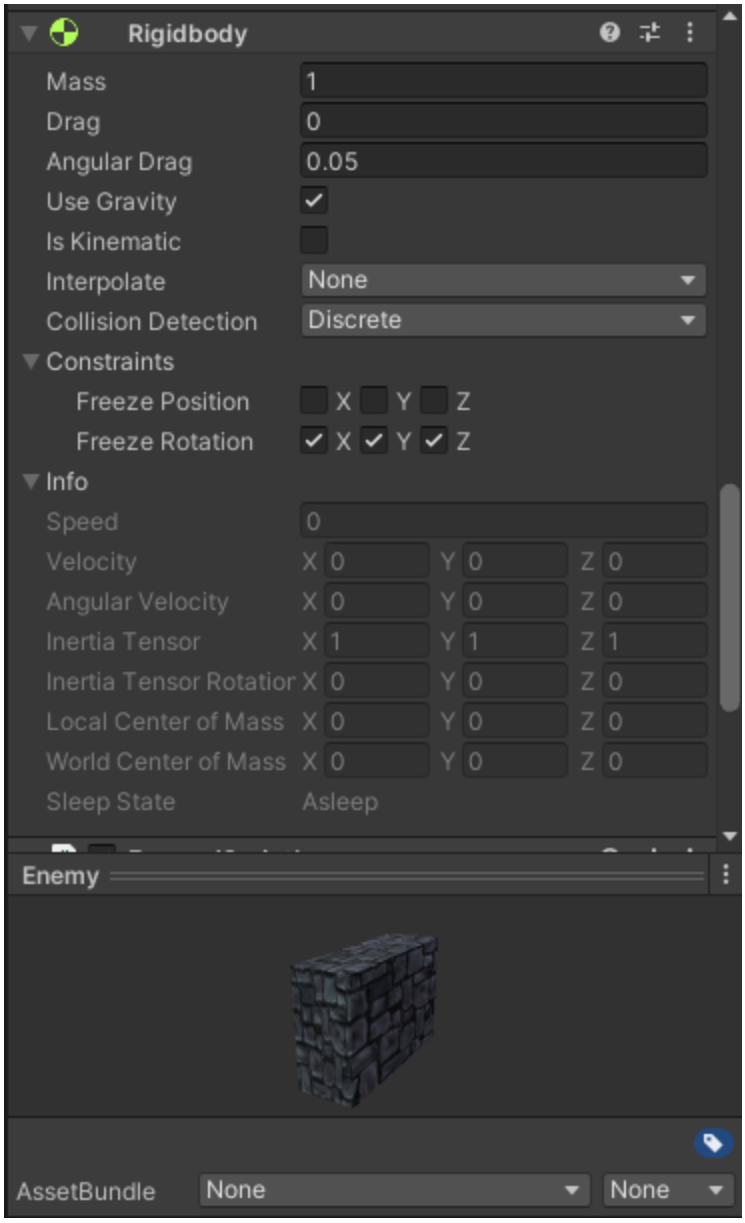
تصویر ۷ - ویژگی های enemy - سایز و ظاهر

و برای تشخیص برخورد از Box Collider استفاده می‌کنیم و متریال آن friction است که اصطکاک را در آن صفر کردیم:



تصویر ۸ - ویژگی های enemy - برخورد

و برای داشتن بدنه سخت از Rigid Body استفاده می‌کنیم:



تصویر ۹ - ویژگی های enemy - بدنه

- بازیکن (Player):

این ماژول وظیفه کنترل کاراکتر اصلی را برعهده دارد، منطق حرکت بر اساس سنسور accelerator تعیین می‌شود و منطق پرش بر اساس صدا و شکست در بازی نیز براساس برخورد با دشمنان تعیین می‌شود.

حرکت به چپ و راست:

تابع AccelInput (که در ادامه بررسی می‌کنیم) وظیفه خواندن اطلاعات از سنسور و تبدیل آن به واکنش مناسب از سوی کاراکتر بر عهده دارد. تابع KeyboardInput با استفاده از ورودی کیبورد کاراکتر را جابه جا می‌کند. این تابع برای دیباگ راحت تر در محیط PC نوشته شده است.

زمانی که در حالت ۱ باشیم تابع SoundInput با استفاده از سنسور صدا پریدن را کنترل می‌کند.

برای اینکه کاراکتر از لبه ها پایین نیفتد مکان آن بین دو عدد ثابت برای محور x باقی می‌ماند:

```
void Update() {
    if(mode == -1){
        mode = GameManager.instance.mode;
    }

    xInput = Input.GetAxis("Horizontal");
    AccelInput();
    KeyboardInput();
    if(mode == 1){
        SoundInput();
    }

    transform.Translate(xInput * dodgeSpeed *
Time.deltaTime, 0, 0);
    float limitedX = Mathf.Clamp(transform.position.x,
-maxX, maxX);
    transform.position = new Vector3(limitedX,
transform.position.y, transform.position.z);
    }
```

تابع AccelInput وظیفه خواندن داده از سنسور را برعهده دارد:

```
void AccelInput() {
    float dirX = 0;
    dirX = Input.acceleration.x;
```

```
    if (dirX != 0)
        xInput = dirX;
    }
```

تابع KeyboardInput وظیفه ورودی گرفتن و اعمال آن بر جهت حرکت را برعهده دارد. این تابع برای تست در محیط PC نوشته شده است:

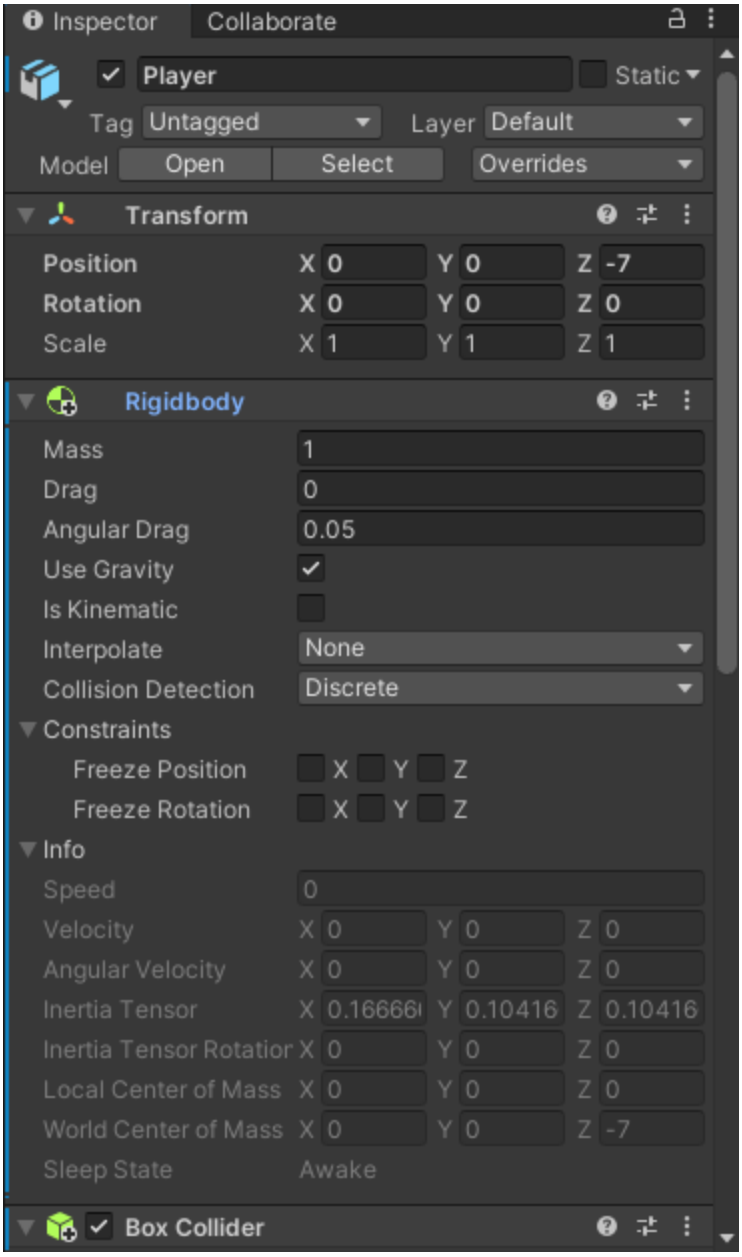
```
void KeyboardInput() {
    if (Input.GetKey("d"))
        rb.AddForce(700 * Time.deltaTime, 0, 0);
    else if (Input.GetKey("a"))
        rb.AddForce(-700 * Time.deltaTime, 0, 0);
    else if (Input.GetKey("w"))
        rb.AddForce(0, 0, 700 * Time.deltaTime);
    else if (Input.GetKey("s"))
        rb.AddForce(0, 0, -700 * Time.deltaTime);
    else if (Input.GetKey("c"))
        rb.AddForce(0, 1000 * Time.deltaTime, 0);
}
```

زمانی که کاراکتر به دشمنان برخورد کند، باید بازی ریستارت شود. برای تشخیص موانع به همه آنان تگ Enemy اختصاص داده ایم:

```
private void OnCollisionEnter(Collision col) {
if (col.gameObject.tag == "Enemy")
        GameManager.instance.Restart();
    else
        touchedGround = true;
}
```

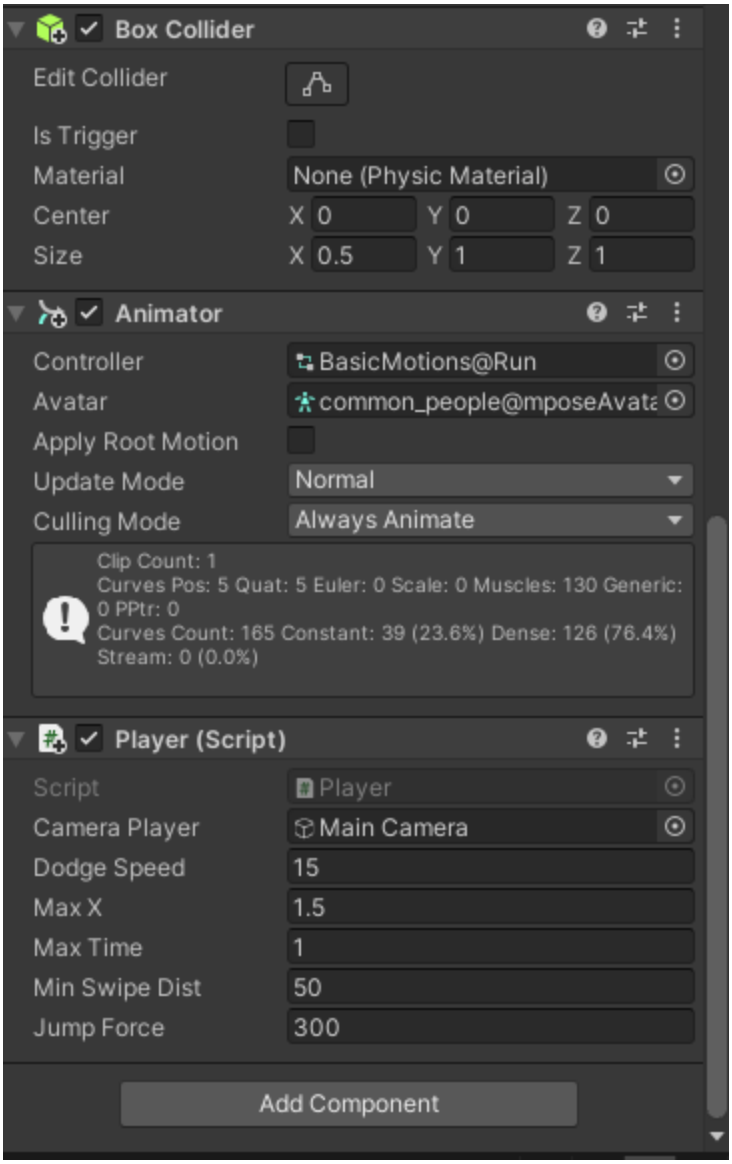
ویژگی های مهم بازیکن:

سایز و مکان اولیه بازیکن را تعیین می‌کنیم:



تصویر ۱۰ - ویژگی های Player - سایز و مکان

و با Box collider برخورد را تشخیص می‌دهیم، همچنین انیمیشن آن و ظاهرش را تعیین می‌کنیم:



تصویر ۱۱ - ویژگی های Player - سایز و مکان

- بوم (Canvas):

این ماژول نوشته ها را در خود نگه می‌دارد. بطور خاص منوی اصلی و امتیاز در حین بازی در Canvas است. وقتی بازی اجرا می‌شود منوی اصلی مخفی می‌شود و امتیاز نمایش داده می‌شود و بالعکس.

- تشخیص صدا (Audio Detection):

این ماژول مسئولیت تشخیص و تحلیل صدا را برعهده دارد.

برای تشخیص peak ابتدا صدا را در یک AudioClip ذخیره می‌کنیم، سپس ۱۲۸ نمونه آخر را بررسی می‌کنیم و peak را تشخیص می‌دهیم.

```
float LevelMax()
{
    float levelMax = 0;
    float[] waveData = new float[_sampleWindow];
    int micPosition =
          Microphone.GetPosition (null) -
(_sampleWindow + 1);
    if (micPosition < 0) {
        return 0;
    }
    _clipRecord.GetData (waveData, micPosition);
    for (int i = 0; i < _sampleWindow; ++i) {
        float wavePeak = waveData [i] * waveData [i];
            if (levelMax < wavePeak) {
                levelMax = wavePeak;
            }
        }
        return levelMax;
    }
```

مقدار testSound را در هر آپدیت تغییر می‌دهیم و همانطور که اشاره شد از آن برای تشخیص پرش استفاده می‌کنیم:

```
void Update()
{
    MicLoudness = LevelMax ();
    testSound = MicLoudness;
}
```

وقتی برنامه در حالت focus نیست نباید صدا را دریافت کنیم برای این کار از روش زیر استفاده می‌کنیم:

```
void OnApplicationFocus(bool focus)
{
    if (focus) {
        if (!_isInitialized) {
            InitMic ();
            _isInitialized = true;
        }
    }

    if (!focus) {
        StopMicrophone();
        _isInitialized = false;
     }
}
```

در این بخش ماژول‌ها و کارهای آن‌ها با جزییات بررسی شد.

7- تست عملکرد

- **طرح تست**

برای تست عملکرد بازی ما از چهار دستگاه گوشی همراه استفاده کردیم و بازی را بر روی آنها نصب کرده و بررسی کردیم.

برای بررسی تشخیص صدا نیز از لاگ نوشتن در زمان تشخیص و هنگام واکنش استفاده کردیم.

- **نحوه اجرای تست (پیاده سازی)**

نسخه apk برنامه را بر روی دستگاه‌ها با نسخه اندروید مختلف نصب می‌کنیم و به بررسی نحوه حرکت و تشخیص ضربه، حرکت و صدای آن می‌پردازیم.

همچنین با نوشتن لاگ در نسخه PC زمان تشخیص صدا و واکنش به آن را اندازه می‌گیریم.

- **نتایج تست‌های انجام شده**

در همه دستگاه‌ها بازی به خوبی و بدون تاخیر و لگ اجرا شد و همه سنسورها به خوبی پاسخ دادند.

زمان تشخیص صدا نیز به این صورت بود:

زمان تشخیص صدا	۴۱۱۷.۶۰۲
زمان پریدن	۴۱۲۶.۱۶۶
تاخیر سنسور تا عملکرد	۸.۵۶۴

- **تحلیل نتایج**

مشخصات دستگاه‌ها:

۱- دستگاه Samsung Galaxy S8 با اندروید نسخه ۹

۲- دستگاه HUAWEI honor 6x با اندروید نسخه ۷

۳- دستگاه Sumsung Galaxy A51 با اندروید نسخه ۱۱

۴- دستگاه Xiaomi Mi9t با اندروید نسخه ۱۰

این بازی بر روی دستگاه‌Xها با نسخه اندروید مختلف و همچنین سخت افزارهای متفاوت تست شد که به خوبی در همه آنها جواب داد و بدون لگ و تاخیر اجرا شد. همچنین همه سنسورها به درستی کار کردند.

8- پاسخ به سوالات طراحی و تایید شده در پروپوزال

- میزان تاخیر از زمان اجرای منطق بازی تا زمان اعمال شدن آن روی بازی، چه مقدار است؟

این موضوع تاثیر مستقیمی رو تجربه کاربر از بازی دارد و یکی از مهم ترین موارد در پیاده سازی بازی هاست. تاخیر دریافت اطلاعات از سنسور، پردازش آنها و تصمیم گیری نباید به گونه ای باشد که کاربر تاخیری در بازی حس کند. در این پروژه تاخیر همه این محاسبات بسیار کم بوده و تاثیری رو کیفیت بازی ندارد. تشخیص دقیق این عدد به خاطر تاثیر مکانیزم تشخیص بر نتیجه، ممکن نیست.

- میزان تاخیر از زمان دریافت ورودی توسط سنسورها تا پردازش آن و به دست آوردن داده‌ی مناسب جهت انتقال به بخش منطق چه مقدار است؟

همانطور که در بخش‌های قبلی گفته شد، این تاخیر میتواند تاثیر مستقیمی رو کیفیت بازی بگذارد و این پروژه بر روی گوشی‌های مختلف تست شد و سرعت دریافت و پردازش اطلاعات به گونه ای بود که روی تجربه کاربر تاثیر محسوسی نداشت. این عدد در بخش‌های قبلی بررسی شد و در حدود ۸.۵ میلی ثانیه است.

- بیشترین نرخ نمونه‌گیری دیجیتال ممکن از داده‌های ورودی (صوت و حرکت) به چه میزان است؟

به ازای هر فریم نمونه گیری اتفاق میفتد، در بیشتر دستگاه‌ها این نرخ ۶۰ فریم بر ثانیه است. بنابراین نرخ نمونه گیری هر ۱۶ میلی ثانیه یک بار است.

9- پیوست‌های فنی

مرجع اندروید SDK

مرجع Unity3d

10-مراجع*

“Car Runner - mobile game” Car Runner. [Online]. Available: https://github.com/mwalasz/Endless-runner. [Accessed: 6-Jun-2021].

“Scripting api” Microphone.devices. [Online]. Available: https://docs.unity3d.com/ScriptReference/Microphone-devices.html [Accessed: 10-Jun-2021].

“video game unity” Brackeys. [Online]. Available: https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA [Accessed: 1-Jun-2021].