תכנות ב-c# לרכיבים פיזיקליים של UNITY

דיברנו בגדול על המערכת הפיזיקלית של unity ועל רכיבים שימושיים שיש לה.
אך עדיין לא דיברנו על איך ניתן לנתב את המערכת ע"י קוד: איך לגרום לאובייקט לנוע בצורה אקטיבית, כיצד להפעיל כוח על אובייקט מאובייקט אחר מבלי להשתמש ברכיבים חיצוניים. כמו כן בכלל לא דיברנו על מה הוא כוח פיזיקלי בטבע ואיך ניתן לחשב אותו.
עתה נתמקד בכל הדברים האמורים לעיל, ובסוף נדגים כיצד ניתן ליישם אותם ע"י משחקון של זריקת כדור לסל.

זיהוי התנגשויות - Collision detection

בunity יש שתי קבוצות של פונקציות לזיהוי התנגשות. הראשונה היא "Collisions" ומכילה שלוש מתודות:

- onCollisionEnter- המתודה נקראת כאשר הקוליידר או הגוף קשיח של האובייקט בדיוק נוגע בקוליידר או גוף קשיח אחר.
- OnCollisionStay- נקראת כמו הפונקצייה update אחת לכל פריים, כל זמן שהאובייקט נוגע בגוף קשיח או קוליידר אחר.
- OnCollisionExit- נקראת כאשר גוף קשיח או קוליידר מפסיק לגעת באובייקט שלו יש את המתודה.

כל אחת משלושת הפונקציות הנ"ל מקבלת אובייקט מסוג Collision כפרמטר. המחלקה Collision מכילה מידע על נקודות קשר, מהירות השפעה וכדו'.
אם לא משתמשים בפרמטר שמועבר בפונקציה, אין צורך לכתוב אותו בחתימת הפונקציה כדי למנוע חישובים מיותרים.
את הקבוצה השנייה כבר יצא לנו לראות בשיעורים הקודמים והיא "Triggers " שמכילה את המתודות OnTriggerEnter , OnTriggerExit ו-OnTriggerStay בדומה ל-"Collisions".
לעומת הקבוצה הראשונה, הפרמטר שמועבר לפונקציה בקבוצה זו הוא מסוג Collider ולא Collision. קוליידר כפי שאנחנו יודעים מכיל שדות כמו is trigger , material וכדו', פחות שדות חישוביים.

ההבדל המהותי ביניהם הוא השימוש שלהן. בעוד הקבוצה הראשונה היא יותר חישובית: שימוש באלמנטים של ה-collision שנכנס לקוליידר כדי לבצע פעולות חישוביות כגון הגדרת עוצמת ההדף מהמכה, חישוב הנזק כתוצאה מהתנגשות וכדו'. הקבוצה השנייה משמשת למימוש התנהגות לאחר התנגשות, מבלי להתייחס לגורמים הפיזיקליים העוטפים את התרחיש. למשל לקחת מטבע או power-up נעדיף להשתמש ב- trigger, כי אין כאן באמת צורך חישובי ובכל זאת אנחנו צריכים את הפרמטר המעובר לפונקציה כדי לבצע שינויים באובייקט.

הערה: בשתי הקבוצות יש צורך שבשני האובייקטים המתנגשים יהיו קוליידרים. כמו כן לפחות אחד מהאובייקטים המתנגשים חייב להכיל Rigidbody אם שניהם עובדים עם Trigger .
כמובן שבדו-ממד חתימת הפונקציה שונה OnTriggerEnter הופך לOnTriggerEnter2D והפרמטר הופך לCollider2D או Collision2D וכו'. חייבת להיות התאמה בממדים של האירועים ושל הרכיבים על שני הגופים המתנגשים – אחרת המערכת לא תזהה התנגשות.

Update לעומת FixedUpdate

Update היא אחת מהפונקציות היותר משומשות ב-unity. היא נקראת אחד לפריים בכל סקריפט שמשתמשים בה.
כמעט כל מחלקה או אובייקט שצריך להשתנות או להיות מותאם לסצנה בצורה רגילה נעשה דרך המתודה:

תנועה של אובייקטים לא פיזיקליים, טיימרים פשוטים או זיהוי קלט הם רק חלק מהדוגמאות לדברים שניתן לעשות עם המתודה.
Update לא נקראת בזמן סדור- אם פריים אחד לוקח יותר זמן לעבד מפריים אחר, אז הזמן בין קריאות למתודה יהיה שונה.

FixedUpdate דומה מאוד ל-update אך יש לה כמה שינויים מהותיים:
בעוד ש-update נקראת בזמן לא סדור, FixedUpdate דווקא נקראת בזמן מדויק, ולכן הפער בין קריאה לקריאה של המתודה לא תלוי בזמן עיבוד של הפריים.
בדיוק לאחר שהמתודה נקראת, נעשים החישובים הפיזיים במנוע.
בתוך זה, כל דבר שמכיל rigidbody, כלומר אובייקט פיזיקלי, צריך להתבצע ב-FixedUpdate ולא ב-Update.
זאת אחת הסיבות שהמנוע שומר על חישוביים מדויקים של פעולות פיזיקליות.
הסקריפט הבא מקרין למסך את הזמנים בהם נקראת המתודה FixedUpdate לעומת Update, הצמידו אותו לאיזשהו אובייקט ריק כלשהו והריצו:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

// GameObject.FixedUpdate example.
//
// Measure frame rate comparing FixedUpdate against Update.
// Show the rates every second.

public class ExampleScript : MonoBehaviour
{
    private float updateCount = 0;
    private float fixedUpdateCount = 0;
    private float updateUpdateCountPerSecond;
    private float updateFixedUpdateCountPerSecond;

    void Awake()
    {
        // Uncommenting this will cause framerate to drop to 10 frames per second.
        // This will mean that FixedUpdate is called more often than Update.
        //Application.targetFrameRate = 10;
        StartCoroutine(Loop());
    }

    // Increase the number of calls to Update.
    void Update()
    {
        updateCount += 1;
    }

    // Increase the number of calls to FixedUpdate.
    void FixedUpdate()
    {
        fixedUpdateCount += 1;
    }

    // Show the number of calls to both messages.
    void OnGUI()
    {
        GUIStyle fontSize = new GUIStyle(GUI.skin.GetStyle("label"));
        fontSize.fontSize = 24;
        fontSize.normal.textColor = Color.black;
        GUI.Label(new Rect(100, 0, 200, 50), "Update: " + updateUpdateCountPerSecond.ToString(), fontSize);
        GUI.Label(new Rect(100, 50, 200, 50), "FixedUpdate: " + updateFixedUpdateCountPerSecond.ToString(), fontSize);
    }

    // Update both CountsPerSecond values every second.
    IEnumerator Loop()
    {
        while (true)
        {
            yield return new WaitForSeconds(1);
            updateUpdateCountPerSecond = updateCount;
```

```
            updateFixedUpdateCountPerSecond = fixedUpdateCount;

            updateCount = 0;
            fixedUpdateCount = 0;
        }
    }
}
```