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 הצגת מיני משחק זריקות לסל.

-Collision detection

לunity 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 הדף מהמכה, חישוב נזק כתוצאה מהתנגשות וכדו'. הקבוצה השנייה יותר 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 לשניהם עובדים עם Trigger .
כמובן שבדו-ממד חתימת הפונקציה שונה OnTriggerEnter הופך לOnTriggerEnter2D והפרמטר הופך לCollider2D וכו'.

-Update & 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;
        }
    }
}
```

– Forces

עתה נדבר קצת על כוחות(forces). כשמדברים על כוחות ב-unity מתכוונים בד"כ לדחיפה או משיכה של אובייקט. לדוגמא אדם ואבן גדולה. כאשר האדם דוחף את האבן הוא בעצם מפעיל כוח שגורם לה לנוע. כמה רחוק האבן תנוע תלוי בכמה חזק אותו אדם דחף את האבן, או בunity-ית: מה ה-magnitude של הכוח, וכמה המסה(mass) של האבן (יש הבדל בין מסה לסתם משקל, מסה בהגדרה היא המידה בה הגוף מתנגד לשינוי במהירות).

למעשה הראשון שהגדיר את זה בצורה פורמלית היה איזייק ניוטון. ניוטון הגדיר שלושה חוקי תנועה המהווים הבסיס למכניקה הקלאסית. החוק הראשון קובע שכל גוף יתמיד במצבו, כל עוד אין כוחות חיצוניים שפועלים עליו. כלומר בהיעדר כוחות חיצוניים, גוף השרוי במנוחה יישאר במנוחה, וגוף נע יתמיד בתנועתו במהירות קבועה בקו ישר.

החוק השני של ניוטון קובע שהאצה של אובייקט הוא תולדה של יחס הכוח מופעל עליו ומשקלו, ככל שהאובייקט כבד יותר ככה צריך להפעיל יותר כוח כדי להזיז אותו, או בניסוח פורמלי: $a = \frac{F}{M}$, כאשר a זה acceleration, F-force ו-M מייצג את המסה.

כמובן שהמשוואה גם לכיוון השני, אם נרצה לגלות כמה כוח הופעל על האובייקט נצטרך לגלות כמה רחוק הוא הגיע ולהכפיל במסה שלו. למשל נניח שהאבן נעה במהירות של $8\frac{m}{s^2}$ (ה-m כאן מייצג מטר) כתוצאה מהדחיפה של האדם, ונניח שמשקלה של האבן הוא 50kg, כמה כוח הופעל על האבן? התשובה $(8 * 50)\mathrm{kg}\frac{\mathrm{m}}{\mathrm{s}^2} = 40kg\frac{m}{s^2}$

נהוג לכתוב את הביטוי $kg\frac{m}{s^2}$ באות N או כ-f_n, השם המקצועי של הכוח הוא כוח ניוטון.

אם אותו אדם ימשיך לדחוף את האבן באותו הכוח, היא תעלה את מהירות שלה בהדרגה בהתאם לכוח המופעל עליה, המהירות היא בעצם המהירות ממנה התחיל האובייקט לנוע ועוד התאוצה של האובייקט כפול הזמן של התאוצה, או בניסוח פורמלי יותר: $v = v_0 + a * t$. במקרה שלנו האדם התחיל לדחוף את האבן כאשר היא נחה במקום, לכן המהירות ההתחלתית שלה היה 0, כלומר אם אותו אדם דוחף את האבן במשך שניה (באותה תאוצה), יוצא שהאבן תנוע במהירות: $0 + 8\frac{m}{s^2} * 1s = 8\frac{m}{s}$.

בשביל למצוא את המרחק אותו עברה האבן נצטרך להיזכר בנוסחה שלמדנו בתיכון: מהירות כפול זמן שווה לדרך, היות והמהירות של האובייקט משתנה מv_0 ל-a*t(המהירות המקסימלית) נחפש את המהירות הממוצעת שעבר האובייקט בתהליך וכך נקבל שהמרחק אותו עבר האובייקט הוא מכפלת ממוצע המהירויות של האובייקט כפול הזמן, כאשר האובייקט עומד במקום בתחילת הדרך נקבל שהדרך היא: $d = \frac{at+0}{2} * t = \frac{at^2}{2}$

ניקח את המקרה שלנו כדוגמא ונקבל: $\frac{1}{2} * 8\frac{m}{s^2} * s^2 = 4m$.

לא קשה לנחש מה יקרה במקרה של שתי כוחות נגדיים שפועלים על אותו אובייקט, למשל נניח שכנגד האדם שדוחף את האבן מהדוגמא הקודמת יש אדם שדוחף אותה בכוח של 600N, הכוח שיופעל על האבן יהיה ההפרש בין שני הכוחות: 400N-600N=-200N והתאוצה של האובייקט תהיה $a = \frac{F}{M} = -\frac{200N}{50kg} = -4\frac{m}{s^2}$

מה שמוביל אותנו לחוק השלישי של ניוטון שקובע שכל גוף המפעיל כוח על גוף אחר, הגוף האחר יחזיר כוח שווה בעוצמתו לגוף הראשון. זה מסביר מדוע כאשר אנחנו דוחפים קיר או קופצים במקום אנחנו נהדפים אחורה- מיצרים כוח שחוזר אלינו, למרות שהגוף שאותו אנחנו דוחפים הוא סטטי ולא יכול להחזיר לנו כוח.

אם כך כיצד בכלל אותו אדם מצליח להזיז את האבן אם היא מחזירה לו את אותו הכוח? התשובה לכך היא משום שמופעלים כוחות נוספים על אותו אובייקט חוץ מהכוח שאותו אדם מפעיל על אבן (כמו חיכוך למשל).

לעומת זאת אם הוא היה דוחף את האבן בחלל כנראה שההדף כתוצאה מהדחיפה היה שווה, והוא היה עף אחורה.

<אפשר להתעמק יותר בכל הנוסחאות האחרות($v^2 = u^2 + 2ad$; $d = vt - \frac{at^2}{2}$; $d = v_0t + \frac{at^2}{2}$) שהן נגזרות של הנוסחאות שלמדנו ולהראות איך הם בהם לידי ביטוי אח"כ בתוכנה>

–Rigidbody properties

תנועה של גופים-

עכשיו שאנחנו יודעים קצת מהתיאוריה שמאחורי הפיזיקה של תנועת גופים במרחב, נראה כיצד ניתן ליישם את זה בקוד. כפי שראינו בשיעורים הקודמים, כאשר אנחנו רוצים לגרום לאובייקט רגיל לנוע אנחנו משתמשים ברכיב translate ובמחלקה transform כדי לשנות את ווקטור הכיוון של האובייקט:

```
transform.Translate(new Vector3(x, y, z));
```

את העדכון בתזוזה נעשה במתודה Update ,ואם נרצה שהתנועה תהיה מבוקרת נָתנֶה את התנועה בקלטים.
הבעיה בתנועה כזאת שהיא לא פיזיקלית, האובייקט אומנם ינוע אבל לא יפעלו עליו חוקי הפיזיקה שלמדנו לעיל, זה מתבטא בכך שהוא לא יוכל לדחוף אובייקט אחר עם תאוצה וכוח. אם האובייקט שייצג את האדם מנסה לדחוף אובייקט אחר המייצג את האבן, אם האבן שוקלת הרבה יותר מהאדם אז לא משנה כמה חזק השחקן ילחץ על המקש שמזיז את הדמות היא לא תצליח להזיז את האבן.
לעומת זאת אם הכוחות פיזיקליים, כלומר האדם ינוע תחת אילוצים פיזיקליים, יכול להיות שאם הדמות תיקח תאוצה מספיק גדולה הוא יוכל להניע את האבן במידה מסוימת
תנועה פיזיקלית בקוד מעט שונה אך עם זאת מאוד דומה ברעיון. בתנועה פיזיקלית אנחנו נפעיל כוח דרך הרכיב rigidbody שנע בכיוון ווקטור מוגדר כלשהו. את העדכון בתנועה נבצע במתודה הייעודית לעדכונים פיזיקליים- FixedUpdate ,ואם נרצה שהתנועה תהיה מבוקרת נשתמש בקלט כדי לנתב את וקטור התנועה.
נסתכל על תנועה פיזיקלית בunity. אם נרצה לשנות את הפוזיציה של האובייקט נשנה את השדה הנקרא velocity, נוכל להשתמש במתודות כמו AddForce, AddExplosionForce ועוד.
אם נרצה לשנות את הסיבוביות של האובייקט, נצטרך לשנות שדה הנקרא AngularVelocity, ונוכל להשתמש במתודות כמו AddTorque או AddRelativeTorque.
המתודות המשומשות ביותר הן AddForce (ו-AddTorque למקרה של סיבוב האובייקט).
נראה הדגמה: פתחו פרויקט חדש ב-unity בגרסת התלת ממד. צרו אובייקט קובייה והוסיפו לו Rigidbody.
הוסיפו לאובייקט סקריפט חדש באיזה שם שתבחרו.
בשביל שנוכל להפעיל את המתודות לעיל נצטרך להשתמש ב-rigidbody של האובייקט לשם כך נשמור אובייקט עצם במחלקה מסוג rigidbody ובמתודה start נבקש את אותו רכיב באמצעות המתודה GetComponent.
במתודה Update שנו את הפונקציה כפי שהיא מופיעה בחתימת הפונקציה ל- FixedUpdate .
כדי לגרום לאובייקט לנוע בכיוון אחד נשתמש בפונקציה של הrigidbody: AddForce.
לפונקציה יש פרמטר אחד שנדרש ואחד אופציונלי. הפרמטר הראשון והנדרש הוא ווקטור הכיוון שאליו הכוח מכוון.
לצורך הנוחות נניח שהאובייקט נע קדימה(רק על ציר ה-z) ולכן נשתמש ב Vector3.forward, אם נרצה שהאובייקט גם ינוע במהירות מסוימת נכפיל את הווקטור בסקלר שייצג את המהירות, מומלץ לעשות משתנה מסוג float שייצג את המהירות ולהגדיר אותו כ- SerializeFiled, כדי שאח"כ נוכל לשנות את המהירות בזמן ריצת הסצנה.
הפרמטר השני והאופציונלי הוא להגדיר את סוג הכוח שמפעילים, בשביל כך נצטרך להגדיר איזה ForceMode מפעילים. ForceMode היא מחלקה שמכילה כמה שדות:
Acceleration- משמש לשינויים מתמשכים בפונקציונליות, אך לא מושפעים מהמסה של הגוף הקשיח.
Force- דומה לAcceleration רק שהשינויים מושפעים מהמסה של האובייקט.
Impulse- שינוי מידי במהירות הגוף(הגוף לא צובר תאוצה). כמו ב-Force השינוי מושפע מהמסה
VelocityChange- כמו Impulse רק שהמסה לא משפיעה על התנועה.
כברירת מחדל, אם לא הוספנו ForceMode כשלהו, המתודה תתבצע כ-ForceMode.Force .
אם נרצה שהאובייקט גם יסתובב נשתמש באותה הדרך בדיוק במתודה AddTorque כמו שהשתמשנו בAddForce
מבחינת קוד זה יראה כך:

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

public class AddForceScript : MonoBehaviour
{
    Rigidbody rb;
    [SerializeField]
    private float _speed = 10f;
    [SerializeField]
    private float _angularSpeed = 10f;

    private float _Velocity;
    private float AngularVelocity;



    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    private void FixedUpdate()
    {
       _Velocity = rb.velocity.magnitude;
        AngularVelocity = rb.angularVelocity.magnitude;
        rb.AddForce(Vector3.forward * _speed, ForceMode.Force);
        rb.AddTorque(Vector3.forward*_angularSpeed,ForceMode.Force);
    }

    void OnGUI()
    {
        GUIStyle fontSize = new GUIStyle(GUI.skin.GetStyle("label"));
        fontSize.fontSize = 22;
        fontSize.normal.textColor = Color.black;
        GUI.Label(new Rect(100, 0, 200, 50), "Speed: " +_Velocity, fontSize);
        GUI.Label(new Rect(100, 50, 200, 50), "AngularSpeed: " +AngularVelocity.ToString("F2"), fontSize);
    }

}
```

הפונקציה OnGUI היא פונקציה שמדפיסה ערכים שהיא מקבלת על המסך, ברעיון היא חוסכת את הצורך ליצור UI במיוחד בשביל הסצנה ,במקרה שלנו היא נועדה כדי להציג לנו על המסך את ההבדלים בין המהירות הסיבובית של האובייקט למהירות התנועה שלו.

שמרו וחזרו לunity , היכנסו לאינספקטור שך הקובייה ובטלו את ה-Gravity כדי שהקובייה לא תיפול בזמן ריצת הסצנה, והריצו את המשחק.

שחקו באובייקט- שנו את המהירות של הקובייה, המסה שלה ה-Drag וכו', כמו כן שחקו בForceMode בקוד, וראו כיצד השינויים הפיזיקליים מתבטאים בהרצת המשחק.

הערה: לדו ממד יש ForceMode אחר:ForceMode2D ולו יש רק שני מצבים אפשריים: Force ו-Impulse

עוד מאפיינים של גוף קשיח

אם נסתכל על המאפיינים של הגוף הקשיח נראה את המרכיבים הבאים:

use gravity, is Kinematic ... אלו שדות בrigidbody שניתן שלנות אותן אינטראקטיבית תוך כדי הרצת הסצנה, לעומתם ישנם שדות (או מתודות) שניתן לגשת אליהם רק ע"י קוד למשל IsSleeping ו-Velocity.

נסתכל על **RigidBody.IsKinematic**. כפי שכבר למדנו, כאשר גוף הוא קינמאטי לא משפעים עליו כוחות חיצוניים. במילים אחרות, אם נגדיר את הגוף כקינמאטי אזי שהמנוע יתעלם מ-collision, joints וכוחות פיזיקאליים שמופעלים על אותו הגוף והוא ינוהל ע"י קוד במקום על ידי המערכת הפיזיקלית של unity.

לפעמים נרצה לשנות אובייקט שהתחיל כאובייקט פיזיקלי לאובייקט קינמאטי, למשל אויב שנפגע מלייזר ועכשיו הוא

"מת", יכול להיות שנרצה שאויב המחוסל יישאר על המסך אך לא יפריע למהלך המשחק, לכן ונעביר אותו למצב isKinematic.

RigidBody.IsSleeping()- היא מתודה שמחזירה ערך Boolean אם הגוף הקשיח במצב שינה.

אז מה זה בעצם מצב שינה? כאשר גוף קשיח נע מאוד לאט המנוע הפיזיקלי מניח שהוא עומד לעצור, ואז מצב "שינה" מופעל. האופטימיזציה הזאת חוסכת זמן עיבוד.

בד"כ נשתמש בזה כדי לזהות מתי אובייקט משחק הפסיק לנוע. למשל נניח שאנחנו רוצים שדמות תזרוק כדורים, אבל שהכדור השני לא ייזרק לפני שהכדור הראשון עצר. נשתמש בisSleeping של הrigidbody של הכדור הראשון, ואם הוא true נזרוק את השני.

Rigidbody.Velocity- ווקטור המייצג מהירות וכיוון תנועה. ברוב המקרים לא מומלץ לשנות את הווקטור כי זה עלול להביא לתוצאות לא ריאליסטיות.

אך יש גם מקרים יוצאי דופן, למשל במשחקי יריות מגוף ראשון כאשר הדמות קופצת נרצה שינוי חד במהירות.

<נשאר נושא אחרון: הצגת משחק בסגנון Angry Bird >