

# תכנות ב-c# לרכיבים פיזיקליים של UNITY

<מילות פתיחה לנושא>

## 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](https://he.wikipedia.org/wiki/%D7%97%D7%95%D7%A7%D7%99_%D7%94%D7%AA%D7%A0%D7%95%D7%A2%D7%94_%D7%A9%D7%9C_%D7%A0%D7%99%D7%95%D7%98%D7%95%D7%9F) 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: , 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 (ה-m כאן מייצג מטר) כתוצאה מהדחיפה של האדם, ונניח שמשקלה של האבן הוא 50kg, כמה כוח הופעל על האבן? התשובה   
נהוג לכתוב את הביטוי באות N או כ-, 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: . 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: .   
בשביל למצוא את המרחק אותו עברה האבן נצטרך להיזכר בנוסחה שלמדנו בתיכון: מהירות כפול זמן שווה לדרך, היות והמהירות של האובייקט משתנה מv0 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:   
ניקח את המקרה שלנו כדוגמא ונקבל: .  
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   
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 מיצרים כוח שחוזר אלינו, למרות שהגוף שאותו אנחנו דוחפים הוא סטטי ולא יכול להחזיר לנו כוח.  
אם כך כיצד איך 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).  
  
<אפשר להתעמק יותר בכל הנוסחאות האחרות() שהן נגזרות של הנוסחאות שלמדנו ולהראות איך הם בהם לידי ביטוי אח"כ בתוכנה>

עכשיו שאנחנו יודעים קצת מהתיאוריה מאחורי הפיזיקה של האובייקטים, כיצד נוכל ליישם את זה בקוד.

### 