# אויבים והתנגשויות

אז אחרי שיצרנו דמות ראשית ומערכת ירי אנחנו רוצים שהדמות שלנו תוכל לירות על משהו ולא רק באוויר.
אויבים הם גם סוג של prefab- לשחקן אין שליטה עליהם, לרובם יש מאפיינים זהים, וככל הנראה הם מופיעים כמה פעמים במהלך משחק. לרוב האויבים 'יורשים' מאובייקט אב אחד ופשוט משכללים את הנתונים שלו, לכן במשחקים קלאסיים בד"כ נראה קבוצות של שונות אויבים אך עם מערכת פעולה דומה.

תחילה נבנה אב טיפוס לכל האויבים במשחק, ניצור אובייקט משחק פרימיטיבי שישמש כבסיס, היות והשחקן הראשי שבחרנו כדוגמא היה קובייה נראה לי מן הראוי להמשיך באותו קו ונבנה גם את האויב כקובייה.
כדי להבדיל בין השחקן הראשי לאויב כדאי שניתן לו חומר בצבע שונה ונשנה גם את הגודל במקצת.
ניתן לו שם ונגדיר אותו כ-prefab ע"י גרירה לתיקיה הייעודית.

## סקריפט בסיסי

ניצור סקריפט חדש ונחבר אותו לדגם של ה'אויב' **בחלון הפרויקט** (כי אם נחבר לאובייקט שמופיע בסצנה החיבור יחשב כ'דריסה' של המאפיינים של הדגם, ואנחנו רוצים לשנות את הדגם דווקא כדי שיוריש לאינסטנסים שלו).
נרצה שהאויב יקיים את הדברים הבאים: 1) זז כלפי מטה במהירות אחידה. 2)אם הוא הגיע לתחתית העמוד שלא יושמד, אלא יחזור למעלה אך מנקודה אחרת בציר ה-X, כלומר יצוץ באופן רנדומלי מלמעלה.3)ברגע שאובייקט 'יתנגש' בשחקן או בלייזר הוא יושמד. כבר ראינו כיצד ניתן לגרום לאובייקט לנוע בקו ישר ולכן לא נתעכב על זה יותר מידי, רק יש לזכור שלמחלקת vactor3 יש כמשתנה וקטור ייעודי למקרה הספציפי שלנו: Vector3.down.

בשביל לגרום לאובייקט שלנו לצוץ מלמעלה בנקודה אחרת על ציר ה-x לאחר שהוא יצא מהמסגרת של המשחק נכיר מחלקה חדשה- מחלקת Random. אומנם ל-c# יש מחלקה ייעודית להגרלת מספרים באופן רנדומלי עם אותו שם, אך החברה של unity פיתחו גם מחלקה ייעודית בכדי להקל על מפתחי המשחקים. למחלקת Random יש את המתודה range(float start ,float end) שמחזירה מספר מוגרל בין שני מספרים שהיא מקבלת כפרמטרים. אנחנו נשתמש במתודה כדי לקבל איזושהי נקודת x שממנה יצוץ האובייקט שלנו. איך נעשה את זה? ראשית נבדוק מה הגבולות של המסגרת שלנו לצדדים וכך נדע את טווח המספרים שממנו אנחנו יכולים להגריל. ברמת העיקרון יש לנו כבר את הטווח הזה מהסקריפט של השחקן הראשי, כאשר רצינו לתת לו טווח תזוזה לצדדים. אח"כ ניצור תנאי בפונקציה update: אם השחקן שלנו יצא מגבולות המסגרת על ציר ה-y, אז נפעיל את המתודה range ונשמור את הערך שקיבלנו במשתנה שייצג את נקודת ה-x הבאה של האובייקט, ובאותו תנאי גם נשנה את ה-position שלו גם לאותה נקודת x וגם את נקודת ה-y כפול מינוס אחד (כי הוא עובר מתחתית המסגרת לראש המסגרת). מבחינת סינטקס (*** אזהרה – בקוד שלכם אל תשתמשו ב"מספרי קסם, אלא בשדות ***):

```
void Update()
    {
        transform.Translate(Vector3.down * _speed * Time.deltaTime);
        if(transform.position.y<-8.0f)
        {
            float randomized = Random.Range(-8.0f, 8.0f);
            transform.position = new Vector3(randomized, 7, 0);
        }
 }
```

## התנגשויות

**התנגשויות** (collisions) הן אחד המנגנונים המרכזיים ליצירת אירועים במשחקים. המנגנון של התנגשויות ביוניטי הוא מורכב למדי, וכדי להבין אותו אנחנו צריכים ללמוד כמה מושגים.

## א. סוגי גופים קשיחים

מבחינה פיסיקלית, מבחינים ביוניטי ב*שלושה* סוגים עיקריים של גופים (ראו באתר יוניטי https://docs.unity3d.com/Manual/class-Rigidbody2D.html):

1. גוף **דינמי** (dynamic) – גוף שיש לו מיקום ומהירות, ופועלים עליו כוחות פיסיקליים כגון כבידה וכד'.

2. גוף **קינמטי** (kinematic) – גוף שאמור לזוז, יש לו מיקום ויכולה להיות לו מהירות, אבל לא פועלים עליו כוחות.

3. גוף **סטטי** (static) – גוף שאמור להישאר במקום אחד, כמו קיר. יש לו מיקום אבל אין לו מהירות.

כל אחד מהם יכול להיות בדו-ממד או בתלת-ממד, כך שבסה"כ יש *שישה* סוגים של גופים.

איך אנחנו מגדירים את סוג הגוף?

1. כדי להגדיר גוף דינמי, צריך להוסיף לו *רכיב גוף קשיח* (RigidBody או RigidBody2D – הראשון תלת-ממדי והשני דו-ממדי). לרכיב הזה יש שדה בשם Body Type, כברירת מחדל הוא Dynamic – דינמי.

2. כדי להגדיר גוף קינמטי, צריך להוסיף לו רכיב גוף קשיח ולשנות את סוג-הגוף שלו ל Kinematic – קינמטי.

3. כדי להגדיר גוף סטטי, אפשר להוסיף לו רכיב גוף קשיח ולשנות את סוג-הגוף שלו ל Static, אבל אפשר פשוט לא להוסיף לו רכיב גוף קשיח בכלל: עצם בלי רכיב גוף קשיח נחשב סטטי כברירת מחדל.

## ב. מתנגשים

כדי שהמערכת תזהה התנגשות בין שני גופים, צריך שיהיה על כל אחד מהם רכיב מיוחד שנקרא **מתנגש** (collider). יש שני רכיבי מתנגשים: דו-ממדי שנקרא Collider2D, ותלת-ממדי שנקרא Collider.

אירוע של "התנגשות" בין שני גופים קורה רק בתנאים הבאים:

א. לפחות אחד משני הגופים הוא *דינמי*;

ב. על שני הגופים יש רכיב Collider;

ג. הממדים של הגופים והמתנגשים הם תואמים (כולם בדו-ממד או כולם בתלת-ממד).

במצב זה, *שני* הגופים יקבלו אירוע שנקרא **Collision** או **Collision2D** – בהתאם לממד.

ניתן "לתפוס" את האירוע הזה ע"י מימוש הפונקציה OnCollisionEnter או OnCollisionEnter2D באחד הרכיבים של הגוף המתנגש. בתוך הפונקציה, אנחנו כותבים מה אנחנו רוצים שיקרה כשהמערכת מזהה התנגשות.

ניתן לממש את הפונקציה המתאימה בכמה רכיבים שונים, וכל אחד מהם בנפרד יקבל הודעה על האירוע (למשל, רכיב אחד יכול לכתוב את אירוע ההתנגשות ל Log, רכיב אחר יכול לגרום לכך שהגוף ייהרס או יסתובב, וכד'...)

## ג. טריגרים

כאמור למעלה, אירועי Collision קורים רק כאשר אחד משני הגופים הוא *דינמי*. האירועים האלה מורכבים, וכוללים חישובים של כוחות, נקודות-מגע, וכו'.

אבל במקרים רבים אנחנו רוצים לזהות התנגשות עם גוף *קינמטי* – גוף פשוט יותר, שלא פועלים עליו כוחות. האירועים האלה מייצגים התנגשות "וירטואלית" - תחושה כאילו קרה איזשהו מאורע, למשל לקבל מטבע, לקחת חפצים מהרצפה, ובמקרה שלנו- לייזר שפוגע באובייקט אויב.

לשם כך המציאו ביוניטי אירוע פשוט יותר וקל יותר לחישוב, שנקרא טריגר - "**Trigger**". וגם כאן יש שני סוגים - אחד בדו-ממד (Trigger2D) ואחד בתלת-ממד (Trigger). אירוע של "טריגר" בין שני גופים קורה בתנאים הבאים:

א. לפחות אחד משני הגופים הוא *דינמי* או *קינמטי* (כלומר, לפחות אחד מהגופים לא סטטי).

ב. על שני הגופים יש רכיב Collider;

ג. רכיב הCollider על אחד משני הגופים לפחות מסומן כ-Trigger

ד. הממדים של הגופים והמתנגשים הם תואמים (כולם בדו-ממד או כולם בתלת-ממד);

במצב זה, *שני* הגופים יקבלו אירוע שנקרא **Trigger** או **Trigger2D** – בהתאם לממד.

ניתן "לתפוס" את האירוע הזה ע"י מימוש הפונקציה OnTriggerEnter או OnTriggerEnter2D באחד הרכיבים של הגוף המתנגש. בתוך הפונקציה, אנחנו כותבים מה אנחנו רוצים שיקרה כשהמערכת מזהה את ההתנגשות.

**שימו לב**: המערכת בשום מקרה לא מזהה התנגשות/טריגר בין שני גופים סטטיים. זה הגיוני – גופים סטטיים לא אמורים לזוז ובפרט לא להתנגש. לכן, גם אם אנחנו בכל-זאת נזיז אותם, למשל ע"י שינוי ה transform, המערכת עדיין לא תחשב עבורם התנגשות. בקישור למטה ניתן לראות שתי טבלאות המסבירות מתי בדיוק נוצרת התנגשות מכל אחד מהסוגים:

https://docs.unity3d.com/Manual/CollidersOverview.html

שימו לב שהטבלאות מתייחסות לכל ממד בנפרד, כלומר: מתנגש דו-ממדי יכול להתנגש רק עם מתנגש דו-ממדי, והתוצאה תהיה התנגשות (או טריגר) דו-ממדית; וכן מתנגש תלת-ממדי יכול להתנגש רק עם מתנגש תלת -ממדי, והתוצאה תהיה התנגשות (או טריגר) תלת -ממדית. העולמות של דו-ממד ותלת-ממד הם נפרדים לגמרי.

(עוד על התנגשויות ביוניטי: https://www.binpress.com/unity-3d-collisions-basics/ וגם https://gamedev.stackexchange.com/q/181370/18261 ).

## בחזרה למשחק

כאור, הרכיב שאחראי על זיהוי וניהול התנגשויות נקרא **Collider**. נחזור ל-unity. אם נסתכל על חלון ה-inspector של האובייקטים שיצרנו נראה שיש להם box collider שהוא אחראי על היכולת של אובייקט להתנגש במרחב. הcolliders הפשוטים (שצורכים פחות זמן עיבוד) הם ה-colliders הפרימיטיביים. בד3 יש את ה-box collider, shape collider ו- capsule collider. ב-2D אפשר להשתמש ב-boxcollider2d ו-circlecollider2d .

בשביל להפעיל את היכולת להתנגשות של האובייקט שלנו נצטרך להוסיף לו רכיב rigidbody (גוף קשיח). בשביל להוסיף רכיב rigidbody נבחר בחלון ה-inspector של האובייקט שלנו ('אויב') add component -> rigidbody, וב-box collider לסמן את is Trigger . נשים לב שב-rigidbody יש כמה אפשרויות לבחירה, במקרה שלנו אף אחת מהם לא ממש רלוונטי אלינו, אבל נעבור עליהם בקיצור:

1) mass – אחראי על המסה של האובייקט, יותר מסה ההתנגשות יכול לגרום ליותר נזק לגוף המתנגש.
2)drag- משמש כדי להאט את האובייקט, כלומר כמה כוחות מושכים את האובייקט ומונעים ממנו התקדמות, ככל שה- drag גדול יותר ככה קצב ההתקדמות שלו יורד.
3) angular drag- משמש כדי להאט את מהירות הסיבוב של אובייקט, ככל שהוא גבוהה יותר ככה מהירות הסיבוב קטנה יותר.
4)use Gravity – האם מופעל כוח משיכה על האובייקט, כלומר האם הוא ינוע כלפי מטה. אם הוא לא מסומן, כלומר מוגדר כ-false אז האובייקט יתנהג כאילו הוא נע 'בחלל החיצון'.
5) is kinematic- מגדיר את האובייקט האם הוא מושפע מגרומים נוספים, או רק מהסקריפט ואנימציות.

**נסו זאת בעצמכם**, ובדקו כיצד האובייקט מתנהג בהתאם לאפשרויות השונות.

לאחר שהוספנו את ה- rigidbody ושחקנו קצת באפשרויות השונות שלו עתה נתעסק בהתנגשות עצמה של האובייקט. אנחנו רוצים שברגע שהאובייקט שלנו יתנגש באובייקט אחר יקרה איזשהו מאורע, למשל כשהלייזר או השחקן הראשי פוגע

באויב, האויב מושמד. ל-unity יש פונקציה מיוחדת בדיוק למקרה הזה: private void onTriggerEnter(Collider other).
המתודה פועלת באופן עצמי, כלומר לא צריך להפעיל אותה בפונקציית update() כי היא מופעלת אוטומטית במקרה של התנגשות עם גוף זר (אם מאפשרים is trigger ב- box collider) ומבצעת את הפקודה שמגדירים לה בפונקציה.
הפונקציה מקבלת כפרמטר איזשהו collider אחר, collider יכול להיות כל אובייקט משחק אחר שמתנגש עם האובייקט שלנו ובלבד שנוכל לזהות אותו.

אבל כיצד נזהה שהאובייקט שהתנגשנו בו הוא לייזר או השחקן? בשביל זה נצטרך להשתמש בתגיות.
נחזור ל-inspector של השחקן הראשי. אם נסתכל למעלה ,שורה מתחת לשם האובייקט, נראה שמופיע שם tag, התגיות מאפשרות לנו לזהות את שם האובייקט. הן משמשות כמו משתנה name מסוג מחרוזת לאובייקט שלנו. על מנת שנוכל להגדיר לאובייקט שלנו שם ייחודי נבחר איזשהו Tag, במידה ואין תגית שמתאימה לנו, למשל player עבור השחקן הראשי , נוכל להוסיף תגית חדשה ב- Add tag. נוסיף תגיות לכל אחד מהאובייקטים שלנו ונחזור לקוד של האוייב.
נרצה שבפונקציה onTriggerEnter יתרחש הדבר הבא: אם התג של הcollider שנכנס הוא כמו של הלייזר אז שיושמד האוייב:

```
if(other.tag=="Laser" ) {
            Destroy(this.gameObject);
}
```

אפשרות שניה לזהות שהתנגשנו בלייזר היא לבדוק שלגוף שהתנגשנו בו יש רכיב בשם Laser (שיצרנו קודם):

```
if (other.gameObject.GetComponent<Laser>() != null) {
            Destroy(this.gameObject);
}
```

אם נריץ עכשיו את המשחק נראה שכשאנחנו פוגעים באויב הוא נעלם אבל הלייזר ממשיך לנוע קדימה, זה משום שלא הגדרנו לו collider ו-trigger. נעשה את אותו תהליך שעשינו לאויב גם ללייזר, כלומר נוסיף לו rigidbody ונסמן is trigger. ניכנס לקוד של הלייזר ונוסיף לו המתודה onTriggerEnter כך שאם הוא התנגש באובייקט עם התגית 'אויב' הוא יושמד גם:

```
private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Enemy")
        {
            Destroy(this.gameObject);
        }
    }
```

## מערכת חיים

חלק מהמשחק הוא גם לדעת מתי הוא אמור להיגמר. בדר"כ המשחקים בסגנון המשחק שלנו נגמרים כאשר הגענו לסוף המסלול, או כאשר השחקן הראשי מאבד את כל הנקודות חיים שלו.
בשביל ליצור לשחקן הראשי מערכת חיים נצטרך להוסיף לו משתנה עצם חדש. נצטרך להוסיף לאובייקט גם מתודה שמורידה לו מהחיים נקודה אחת, לה נקרא כאשר תהיה התנגשות בין השחקן שלנו לאויב, לצורך העניין נקרא למתודה public void Damage():

```
public void Damage()
    {
        life--;
        if(life<1)
        {
            Destroy(this.gameObject);
        }
    }
```

במשחק שלנו ספציפית אין צורך להוסיף לשחקן הראשי collider ניתן להשתמש ב-collider של ה'אויב'.

בשביל להפעיל מתודה של אובייקט משחק מתוך סקריפט של אובייקט אחר אנחנו צריכים לקבל את אותו אובייקט, ל unity יש מתודה מיוחדת במיוחד בשביל זה: transform.GetComponent<GameObject object Component>().
אנחנו רוצים שבפונקציה onTriggerEnter של האויב ,מתי שקורת התנגשות בין אויב לשחקן אז תופעל המתודה Damage() של השחקן ונהרוג את האויב שלנו. לשם כך נצטרך לבקש אובייקט 'שחקן' והרכיב שלו הרלוונטי לנו (במקרה שלנו הסקריפט נשמר ב-inspector ברכיב- "Player") ולשמור אותו במשתנה מסוג שחקן, אח"כ נפעיל את המתודה שלו ונשתמש במתודה destroy() כדי להשמיד את האובייקט 'אויב':

```
if(other.tag=="Player")
        {
            Player player= other.transform.GetComponent<Player>();
            player.Damage();
            Destroy(this.gameObject);
        }
```

בשלב זה, השחקן לא רואה שום חיווי על כמות החיים שלו או על כך שקרתה התנגשות. בהמשך כמובן נרצה לתת לו חיווי על כך. יש דרכים רבות לעשות זאת, למשל:

- אפקט קולי – קול של התפוצצות;
- אפקט חזותי – חלקיקים צבעוניים המייצגים התפוצצות;
- טקסט או אייקונים המראים על מספר ה"חיים" שנשארו לשחקן, בפינת המסך;
- שינוי כלשהו בדמות של השחקן על המסך, למשל הקטנה או שינוי צבע;

בהמשך נלמד על כך בהרחבה.