חלליות

רקע למשחק-

<נכתוב כאן רקע למשחק היריות>
במסמך הקרוב ננסה לבנות בסיס למשחק יריות בין גלקטי.

יצירת דמות-

בתור התחלה נבנה בסיס לדמות- ניצור אובייקט משחק חדש(GameObject) שיהווה בסיס לשחקן הראשי ולאויבים.
לצורך הדוגמא ניצור אובייקט מסוג קובייה, כמובן שבהמשך יהיה ניתן להחליף את הקובייה בדגם/תמונה של חללית, אך כיוון שעכשיו אנחנו עובדים רק על התשתית של המשחק די לנו באובייקט משחק פרימיטיבי.
נקרא לקובייה בשם, לשם הפשטות נקרא לה player. נחליט על גודל שנראה לנו מתאים לשחקן הראשי, ונמקם אותו במקום שהכי קל יהיה לחשב ממנו שהוא הווקטור (0,0,0), כלומר בחלק של ה-position ב-inspector נשים את הערכים 0,0 ו-0 במקום ה-x, y ו- z בהתאמה.
במידה ולא מרוצים מהצבע של השחקן הראשי שלנו נוכל להוסיף לו חומר(מומלץ להציץ שוב במסמך 'מה זה unity').
לפני שניכנס לקוד ולהפעלת הדמות נצטרך לשנות את הרקע למשהו שיהיה לנו יותר קל לראות דרכו את המשחק בנתיים.
בשביל לשנות את רקע נכנס לאובייקט המצלמה, וב- inspector נבחר clear flags -> Solid Color, ,ומתחת נבחר ב- background את הצבע שנראה לנו הכי מתאים.

הזזת השחקן-

לאחר שסיימנו לבנות את גוף השחקן נתעסק בלבנות את השכל שמנחה אותו- הקוד.
ניצור סקריפט חדש לשחקן ונקרא לו בשם זהה לאובייקט. כמו שכבר הזכרנו בשיעורים קודמים מומלץ לשמור את הסקריפטים שלנו בתיקייה ייעודית לסקריפטים.
בכדי להתאים את הסקריפט לאובייקט עליו הוא מפעל פשוט נגרור אותו ל-inspector של האובייקט עצמו.
נרצה שלשחקן שלנו יהיו הדברים הבאים: 1)יכולת הזזה- שנוכל להזיז את השחק בעזרת לחיצה על החצים במקלדת, או לחיצה על העכבר. 2) מערכת חיים- כלומר כמה חיים יש לשחקן. בד"כ המערכת חיים נעה בין חמש לשלוש, וכל פסילה מורידה לו נקודת חיים אחת מהמשחק, וכאשר השחקן גומר את חייו המשחק נגמר. 3)נקודות- כדי שנוכל לדעת איפה אנחנו עומדים ,האם השתפרנו בין משחק למשחק.
בשביל להזיז את השחקן שלנו נצטרך להתעמק קצת בפונקציית ()update ובשתי מחלקות חדשות: input ו-transform.
ניזכר בפונקציה ()update, הפונקציה נקראת כל פריים של המשחק כלומר בכל פעם שהמסך מתעדכן אנחנו מבצעים מחדש את המתודה(כמין לולאה שעובדת לכל אורך חיי האובייקט).
אם נרצה שהאובייקט שלנו יזוז במסך נצטרך בעצם לעדכן את הפוזיציה שלו(של ה-x או ה-y) בכל פעם.
כאן בדיוק נכנסת מחלקת transform: לכל אובייקט משחק(בין אם אובייקט ממשי, מצלמה, תיאורה וכו') יש transform , אשר משמשת לביצוע מניפולציות על האובייקט בין אם שינוי גודל , מיקום או סיבוב האובייקט. למעשה כבר נפגשנו עם המחלקה לפני בinspector של האובייקט אך עדיין לא ראינו כיצד ניתן לשנות בזמן ריצת המשחק.
אחת המתודות של המחלקה היא translate שהיא מתודה שמקבלת איזשהו וקטור של שלושה ממדים (`Vector3`) או של שני ממדים(`Vector2`), **ומחברת** בין המיקום הנוכחי של האובייקט לווקטור ששלחנו כפרמטר.

למשל אם נרצה לקדם את האובייקט שלנו ביחידה אחת כלפי מעלה כל פריים נצטרך להוסיף בפונקצייה ()update את קטע הקוד הבא:

```
transform.Translate(new Vector3(0,1, 0));
```

למעשה החברה של unity ידעו כבר שהזזה ביחידה אחת לכיוון מסוים היא שכיחה מאוד ביצירת משחקים, ולכן הם גם יצרו משתנים שמקצרים את כתיבת קוד במעט: למחלקת Vector3 יש את המשתנה up שהוא למעשה הווקטור (0,1,0) והמשתנה down שהוא הווקטור (0,-1,0) כלומר ה-y יורד באחד. ובאותו אופן יש אתת המשתנים right ו- left שבהם רק ה-x משתנה בהתאמה. הרי שניתן לכתוב את אותו הקוד גם כך: `transform.Translate(Vector3.up);`

אם נשמור ונריץ את המשחק נראה שאובייקט שלנו טס כלפי מעלה במהירות עצומה, אם בכלל הצלחנו לראות אותו מרוב שהוא מהיר, במקום לנוע במהירות של יחידה לשנייה למשל, זה משום שהמתודה מתעדכנת במהירות עצומה, פי כמה וכמה ממה שרצינו.

אז מה צריך לעשות כדי שהמהירות תתאים למהירות אחידה של שנייה לדוגמא? גם על זה unity כבר חשבו ויצרו מחלקה מיוחדת שקוראים לה Timeשאחראית על מידע לגבי הזמן. למחלקה יש משתנה מיוחד שקוראים לו deltaTime שהוא משלים את הפריים למהירות של שניה, כלומר אם נכפיל את הווקטור שלנו באותו הdeltaTime הוא יתקדם בהרבה פחות מיחידה אחת לכל פריים, וכך מתי שתעבור שניה הוא יתקדם בעצם יחידה אחת:

```
transform.Translate(Vector3.up * Time.deltaTime);
```

אם נרצה להגדיל את המהירות נוסיף פשוט משתנה speed מסוג float ונכפיל את הווקטור גם בו:

```
transform.Translate(Vector3.down * _speed * Time.deltaTime);
```

זה המקום להוסיף כי ניתן לראות את כל המשתני העצם הציבוריים של המחלקה ב-inspector, ובכדי לראות משתנים פרטיים יש להוסיף מעל למשנה את התווית SerializeField כך:

```
[SerializeField]
private float _speed = 5f;
```

אז הצלחנו להזיז את השחקן שלנו, אך עדיין זה לא קורה תחת השליטה שלנו, מה גם שמהרגע שהוא יוצא מהמסגרת של המשחק הוא נעלם.

נתחיל מהסוף להתחלה- השחקן שלנו נעלם משום הוא לא מפסיק לנוע כלפי מעלה, אנחנו רוצים שאם הוא זז מעבר למסגרת של המשחק שהוא יחזור מהצד השני ,כלומר אם הוא עלה הוא יחזור מלמטה, ואם הוא זז ימינה מידי אז שיופיע מצד שמאל, וכנ"ל הפוך.

לשם כך נצטרך להכיר את transform.position שהוא למעשה משתנה מסוג ווקטור גם כן שמייצג את המיקום הנוכחי של השחקן. משום שposition הוא ווקטור ניתן גם לקבל משתנה ספציפי ממנו (את ה-x ,y או z), וניתן לשנות אותו, את כולו אבל לא חלק ממנו, למשל אי אפשר לשנות רק את האיקס של הווקטור.

עתה נצטרך למצוא את גבולות המסגרת- מאיזה פוזיציה השחקן שלנו יוצא מהמסגרת.

בשביל זה נצטרך להריץ את המשחק וללחוץ על pause בדיוק בשלב שכבר לא רואים את השחקן על המגרש.

אנחנו צריכים למתוא את מיקום ה-y של השחקן(כי הוא מתקדם כלפי מעלה) במצבו הנוכחי, ניתן לראות את המיקום של השחקן ב-inspector, ומה שמוצג שם מייצג את הגבול שממנו לא ניתן לראות את השחקן, כלומר הנקודה שממנה נרצה להפוך את המיקום של השחקן לכיוון השני.

נשים לב שהיות והתחלנו מהמיקום (0,0,0),אז גבול המסגרת העליון הוא בדיוק כמו הגבול התחתון של המסגרת רק כפול מינוס אחת ,כלומר אם גילינו שמהנקודה y=7.0f (הposition ב-float) כבר לא רואים את השחקן, אז מהנקודה y=-7.0f גם השחקן יצא מהתחום רק למטה. לכן אם השחק שלנו עבר בנקודת ה- y (בערך מוחלט) את 7.0f נכפיל את ה-y שלו במינוס אחת (להפוך צדדים), היות ולא ניתן לשנות רק איבר אחד בווקטור אלא את כולם ,נשתמש באופציה לקבלת משתנה ספציפי מהווקטור, זה יראה בקוד כך:

```
if (Mathf.Abs(transform.position.y) > 7f)

   transform.position = new Vector3(transform.position.x, transform.position.y*-1, transform.position.z);
```

בשביל למצוא את המסגרת האופקית נשנה את הפונקציית translate ל-vector3.right במקום up ונעשה את אותו התהליך.
ועכשיו לשאלת השאלות כיצד ניתן לשלוט בדמות- שהדמות תזוז לאיזה כיוון שאני מכוון אותה לזוז במקום שהיא תנוע רק בקו ישר. לשם כך נצטרך להכיר מחלקה חדשה ,מחלקת input.
מחלקת input משמשת בכדי לקרוא קלט מהמשתמש באמצעות axes (צירים) עליהם היא עובדת. לכל סוג axes יש שם מיוחד משלו, למשל הקלט של תזוזה לכיוון ציר ה-x מהעכבר נקרא Mouse X, ואילו מהמקלדת נקרא Horizontal .
לunity יש כ-18 קלטים דיפולטיביים וניתן להוסיף עוד. בשביל לראות את כל הקלטים האפשריים שמגיעים עם unity ניכנס ל- edit-> project setting-> input, בחלון ה-input ניתן לראות את שמות כל ה-axes ומאיזה כפתורים הם קולטים, למשל ה-axe 'Horizontal',שאחראי לתזוזה אופקית, מקבל קלט מהכפתורים: חץ ימינה (או המקש D במקלדת)- למקרה שהתקדמנו ימינה ,וחץ שמאלה(או המקש A במקלדת)- למקרה שאנחנו מתקדמים שמאלה.
בכדי שמחלקת input תוכל לקבל את הקלטים נצטרך להשתמש במתודה ;Input.GetAxis(string AxeName), שמקבלת כפרמטר את שם ה-axe ומחזיר את הערך אחד אם קיבלנו התקדמות לכיוון החיובי (לדוגמא ב-Horizontal אם התקדמנו ימינה), או מינוס אחד אם התקדמנו לכיוון השלילי של הצירים. במידה ולא קיבלנו קלט בכלל הפונקציה מחזירה את הערך 0.
אם כך כיצד נוכל לשלב את המידע החדש עם הקוד שלנו כך שההתקדמות של השחקן תהיה בשליטתנו? פשוט ניצור משתנה חדש שמקבל את הערך שתיתן הפונקציה ונשתמש בו בפונקציה translate כך שהשחקן יתקדם בהתאם לקלט אותו קיבלנו- אם קיבלנו ערך חיובי ,למשל פנינו ימינה, אז השחקן יתקדם יחידת מרחק אחת חיובי מהמיקום הנוכחי שלו, ואם נגיד לא לחצנו על שום כפתור, אז הפונקציה תחזיר 0 כך שאם נחבר את הערך מהפונקציה עם הווקטור של השחקן אז השחקן יישאר במקום.
מבחינת סינטקס זה יראה כך:

```
float horizontal = Input.GetAxis("Horizontal");
float vertical = Input.GetAxis("Vertical");
transform.Translate(new Vector3(horizontal, vertical, 0) * _speed * Time.deltaTime);
```

במקרה לעיל השחקן שלנו יזוז למעלה או למטה בהתאם לחצים או למקשים D,S,A ו-W במקלדת.

-Prefabs

נח לבנות אובייקט משחק חדש(GameObject) בסצנה ע"י הוספת רכיב ועריכת המאפיינים שלו לערכים המתאימים.
אולם זה יכול ליצור בעיות כאשר אנחנו מתעסקים עם אובייקטים כמו NPC- non-playe character כלומר דמות שנשלטת באמצעות המכונה ולא ע"י השחקן, חלק מנוף- עץ למשל או סלעים, או סתם עזרים שיש לשחקן, שהמשותף לכולם שכולם אובייקטים שיכולים להופיע יותר מפעם אחת במהלך המשחק וחולקים מאפיינים דומים.
לשכפל את האובייקטים אומנם יצור העתק שלהם, אך השכפול יגרום לכל עותק לעמוד בפני עצמו , כך שאם נרצה לשנות את המבנה של האובייקטים נצטרך לעבור כל העתק בנפרד ולשנות אותו, במקום שיהיה לנו איזשהו אובייקט אב שכל שינוי שיתבצע בו יתבטא בכל העתקים שלו ישירות.
למרבה המזל ל- unity יש את ה-asset "prefab" שמאפשר לאחסן אובייקט משחק שלם עם רכיבים ומאפיינים כ'תבנית' לכל העותקים שלו. בדומה למחלקות וממשקים בשפות תכנות- כל שינוי שיתחולל במחלקת (או ממשק) האב יתבטא גם באינסטנסים שלו. בנוסף ניתן לדרוס(override) רכיבים ולשנות מאפיינים של prefab אב, דומה מאוד לירושה.
אז כיצד יוצרים prefab? נהוג ליצור תיקיה ייעודית לכל ה-prefabs במשחק. בכדי ליצור אובייקט prefab חדש נצטרך לגרור את האובייקט מתוך הסצנה, כלומר מה-hierarchy view או מחלון הסצנה (scene view) , לתיקיה הייעודית בחלון הפרויקט (project view). אם האובייקט מופיע בחלון הפרויקט והוא צבוע כחול ב- hierarchy view סימן שהפעולה הצליחה, ועכשיו כל פעם שנרצה להוסיף עוד העתק של אותו ה-prefab נצטרך פשוט לגרור את האובייקט מחלון הפרויקט לחלון הסצנה או ל-hierarchy view.
כמו שהוזכר קודם עריכה של ה- asset עצמו, כלומר אותו אובייקט prefab שניתן למצוא בחלון הפרויקט, ישתקף על כל העתקים שלו, אך ניתן גם לשנות כל העתק אינדיבידואלית, זה שימושי כאשר אנחנו רוצים ליצור כמה NPC דומים אבל בווריאציות שונות כדי שהמשחק יראה יותר ראליסטי. כדי להבהיר מתי אינסטנס של prefab דורס את אובייקט האב שלו הוא מוצג ב-inspector עם תווית שם **מודגשת** (וכאשר רכיב חדש נוסף לאובייקט שהוא אינסטנס של prefab, כל המאפיינים שלו מודגשים). בהמשך נראה דוגמא לשני prefabs מהמשחק שלנו- לייזר ואויבים.

לייזר-

יריות או לייזר אלה דוגמאות טובות לסוג של prefab היות וכל היריות צריכות להיות זהות, אנחנו נקרא להן דיי הרבה במהלך המשחק ואין לשחקן שליטה על מנגנון הפעולה שלהן למעט לכוון אותן.
ראשית נעמוד על כמה תכונות של היריות: 1)כל היריות מתנהגות באופן זהה עם אותה מטרה. 2) כל ירייה מגיעה רק אם התרחש איזשהו מאורע (השחקן הראשי ירה ברובה למשל). 3)לכל ירייה יש טווח, היריות נעלמות מהמשחק אם הן פגעו במטרה או אם הן יצאו מהמסגרת של המשחק. 4) סוגי לייזרים שונים הם פשוט העתק של אובייקט לייזר ראשוני עם דריסה של כמה רכיבים. בתור התחלה ניצור אובייקט פרימיטיבי שיהווה דוגמת ירי, אישית אני הייתי בוחר בצילינדר כי יש לו דמיון לקליע.
נשנה את הגודל של האובייקט כך שיראה בגודל ראליסטי יחסית לשחקן שלנו, נגיד השחקן ב-scale של (1,1,1) נעשה את הלייזר ב-scale של 20% ממנו כלומר (0.2, 0.2, 0.2). ניתן ואף מומלץ להוסיף ללייזר חומר בכדי להבליט את הלייזר ברקע.
בדיוק כמו שאמרנו כבר קודם בכדי להפוך את הלייזר שלנו ל-prefab נצטרך לגרור אותו מהחלון בסצנה (או הhierarchy view) לתיקייה הייעודית ל-prefabs בחלון הפרויקט.

סקריפט-

עתה נתעסק בלוגיקה של האובייקט. ניצור סקריפט 'Laser' ונחבר אותו לאובייקט שנמצא **בחלון הפרויקט**(כי אחרת זה דריסה).
נרצה שהלייזר שלנו פשוט יתקדם קדימה מהרגע שהתרחש המאורע שקרא לו. איך להזיז אובייקט בקו ישר כבר ראינו כאשר התעסקנו בתזוזה של הדמות הראשית. כל מה שנצטרך זה להשתמש בפונקציה transform.translate עם ווקטור3 שמתקדם כלפי מעלה כפול הדלתא טיים ומהירות(כי הרי הלייזר אמור להתקדם מהר בהרבה מהשחקן) ומבחינת סינטקס:

```
void Update()
    {

        transform.Translate(Vector3.up * Time.deltaTime*_speed);
       . . .
    }
```

כמובן שאנחנו רוצים לתת ללייזר שלנו טווח, הרי לא נרצה שהוא יתקדם עד אין סוף, כי זה סתם תופס משאבים.
לשם כך נצטרך להכיר את המתודה `Destroy` שהיא משמידה את האובייקט משחק אותו היא מקבלת כפרמטר, במקרה שלנו נרצה להשמיד את this.gameObject במקרה שיצאנו מגבולות המשחק:

```
if(transform.position.y>8.0f)
        {
            Destroy(this.gameObject);
        }
```

אם נריץ את המשחק בינתיים נראה שאותו לייזר שיצרנו מתקדם כלפי מעלה בקו ישר. עתה אנחנו יכולים למחוק אותו מהסצנה (אבל לא מחלון הפרויקט) היות ולא נצטרך אותו אלא אם כן יתרחש מאורע שקרא לו(למשל לחיצה על רווח במקלדת).
בשביל לעשות את זה נצטרך לחזור לסקריפט של השחקן הראשי.
נרצה שהשחקן הראשי יפעיל ,או יותר נכון ייצור אינסטנס של הלייזר שלנו כאשר הוא מקבל קלט מהמקלדת למשל רווח.
כבר נפגשנו עם מחלקת input שמתעסקת עם קלטים מהמשתמש ובמיוחד במתודה "GetAxies" שמחזירה ערך בהתאם להתקדמות לכיוון החיובי או השלילי של הצירים, כרגע נשתמש במתודה אחרת של המחלקה- 'GetKeyDown' שמקבלת כפרמטר שם של מקש (כמחרוזת) ומחזירה ערך בוליאני אם הקשנו על אותו מקש או לא. במילים אחרות נרצה ליצור תנאי: אם קיבלנו ערך חיובי מהפונקציה(כלומר לחצנו על המקש) אז שיוצר אינסטנס של לייזר.
בשביל ליצור אינסטנס של prefab נוכל להשתמש במתודה instantiate(gameObject, transform, Quaternion rotation)
הסבר: המתודה מקבלת כפרמטר איזשהו אובייקט משחק כלשהו, vector3 עם המיקום של האובייקט, ואיזשהו וקטור לסיבוב האובייקט אם נרצה ליצור אותו עם סיבוב כלשהו. לרוב לא נרצה שהאובייקט שאותו אנחנו מפעילים יגיע מסיבוב, לכן נוכל להשתמש במשתנה `Quaternion.identity` שיוצר את האובייקט החדש באותה זווית סיבובית של האובייקט שקרא לו, כלומר הלייזר שלנו יהיה באותו כיוון כמו השחקן הראשי.
מבחינת מיקום נרצה שהלייזר יתחיל קצת מעל לשחקן אבל באותו קו שלו, לשם כך נצטרך להשתמש במיקום של השחקן ולשנות רק את הנקודה y של השחקן כך שתהיה גבוהה יותר במעט:

```
Vector3 laser_position = new Vector3(transform.position.x , transform.position.y+1, 0);
```

בשביל לשלוח אובייקט מסוג לייזר כפרמטר נצטרך לשמור gameObject מסוג לייזר כמשתנה עצם של המחלקה.
זה המקום להזכיר שמשתני עצם של המחלקה שאנחנו יוצרים כפרטיים אנחנו עדיין יכולים לראות ב-inspector אם הגדרנו מעליהם [serializeFiled]. עתה ניצור אובייקט כזה, לצורך הפשטות נקרא לו laser ונחזור למנוע הגרפי.
נשים לב שבחלון ה-inspector של השחקן הראשי, מתחת לסקריפט מופיעים המשתני עצם שלו .
בכדי להגדיר שהאובייקט עצם של player הוא מסוג לייזר נצטרך לגרור מחלון הפרויקט את ה- prefab 'לייזר' לאיפה שמופיע המשתנה laser ב-inspector של השחקן הראשי. עתה שהגדרנו את האובייקט אפשר להשתמש בפונקציה instantiate:

```
if(Input.GetKeyDown("space"))
        {
            Vector3 laser_position = new Vector3(transform.position.x , transform.position.y+1, 0);
            Instantiate(laser, laser_position, Quaternion.identity);
        }
```

תרגיל: נסו לחשוב איך ניתן ליצור דייליי בין יריה ליריה כך שנצטרך לחכות קצת זמן בין היריות והן לא יתחילו אוטומטית כל פעם שנלחץ 'רווח'.

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-

ניצור סקריפט חדש ונחבר אותו לאובייקט 'אויב' שיצרנו **בחלון הפרויקט**, כי אם נחבר לאובייקט שמופיע בסצנה החיבור יחשב כ'דריסה' של המאפיינים של ה-prefab ואנחנו רוצים לשנות את אובייקט האב 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 שנפעיל 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:

```
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-

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.

אז מה זה בעצם collider? הוא האחראי על התנגשויות של גופים במרחב. הcolliders הפשוטים(שצורכים פחות זמן עיבוד)הם ה-colliders הפרימיטיביים. בd3 יש את ה-box collider, shape collider ו- capsule collider. ב-2D יש אפשר להשתמש ב- boxcollider2d ו-circlecollider2d .

ישנם שני סוגים של collisions (התנגשויות): 1)hard surface collision- התנגשות בין עצמים מוחשיים במרחב, למשל כדור שפוגע בקיר, או התנגשות בין שתי מכוניות. 2) trigger collision- שנותן תחושה כאילו קרה איזשהו מאורע, למשל לקבל מטבע, לקחת חפצים מהרצפה, ובמקרה שלנו- לייזר שפוגע באובייקט אויב.

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 בעצמך, 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 ברגע 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 זהות 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 מחרוז 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);
        }
```

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 מעבד 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);
        }
```

– Spawn manager

אז לאחר שיש לנו כבר שחקן שמסוגל לירות ואויב שניתן להרוג השלב הבא שמתבקש הוא להכניס עוד כמה אויבים למשחק.
אז איך נעשה את זה? אם נכניס אותם פשוט אחד אחד מחלון הפרויקט אומנם יהיו לנו כמה אויבים במסך, אך כולם יופיעו ישירות על המסך, והשאיפה היא להכניס אותם בסדר מסוים כך שכל פעם יעלו מספר האויבים הפוטנציאלים בקצב אחיד ולא בפעם אחת, מה גם שזאת סתם עבודה קשה להכניס עכשיו עשרים ומעלה אויבים.
למעשה אנחנו כבר מכירים שיטה ליצור אינסטנס של prefab במהלך המשחק, אם ניזכר בשלב יצירת הלייזר השתמשנו בפונקציה instantiate בכדי ליצור את היריות, למה שלא נשתמש בה גם כאן? למה שלא ניצור אינסטנסים של אויבים בתדירות של כל חמש שניות למשל? ברמת העיקרון המתזמן הוא לא אובייקט, לא נראה אותו על המסך עם סטופר מזניק תור של אויבים כל אחד בתורו. למזלנו unity דאגו גם לזה ונתנו לנו את האפשרות ליצור אובייקט ריק (Empty object).
כדי ליצור אובייקט ריק , נלחץ על מקש ימני-> create empty. ניתן לאובייקט שם, בהגה המקצועית נהוג לקרוא למתזמן spawn manager, ונחבר לאובייקט שיצרנו סקריפט עם אותו שם.
יצירת thread היא בעייתית במקצת במיוחד במשחקים מורכבים, היות והיא דורשת מהאובייקטים להיות מסונכרנים לתרד הראשי, פעולה שצורכת יותר מידי משאבים יחסית למה שהיא מועילה לנו, למרבה המזל ישנן דרכים נוספות שנוכל להשתמש בהן בכדי לקבל את האפקט של התזמון, דרך נוספת היא באמצעות מתודות coroutines.
כשאנו קוראים לפונקציה היא רצה עד שהיא משלימה את עצמה, ורק אז מחזירה ערך אם בכלל. למעשה זה אומר שכל פעולה שמתרחשת בפונקציה קורת בתוך פריים אחד של המתודה update(). קריאה לפונקציה לא יכולה לשמש בכדי להכיל תהליך של אנימציה או רצף אירועים התלויים בזמן.
מתודות coroutines הן מתודות שמאפרות להחזיר ערך 'זמני' מתוך הפונקציה עד הקריאה הבא למתודה שמשם היא ממשיכה מאותו מקום שבו עצרה בפעם הקודמת. למשל ספירה עד עשר: המתודה תחזיר כל פעם מספר מאחד עד עשר.
בכדי שהמתודה תוכל להחזיר משתנה זמני עליה לקיים שני דברים: 1) המשתנה בחתימת הפונקציה אותו היא מחזירה צריך

להיות מסוג IEnumerator שהוא כמו משתנה מסוג iterable ב-c++, כלומר משתנה שמציג את הערך האחרון בו אנחנו נמצאים שלנו. 2) במקום שאנחנו מחזירים ערך עם return אנחנו מחזירים עם yield return.
כברירת מחדל מתודות coroutine נקראות מחדש כל פריים, אך ניתן להשהות את זמן הקריאה שלהן, כך שנקרא להן אחת לפרק זמן מסוים ע"י שנחזיר משתנה waitforseconds עם כמה זמן להשהות עד הקריאה הבאה:

```
yield return new WaitForSeconds(float time);
```

חזרה לענייננו: בדיוק כמו שעשינו עם השחקן נצטרך לעשות גם כאן- כדי לייצר אינסטנסים של אובייקט 'אויב', ניצור משתנה עצם חדש למחלקה spawnManager מסוג GameObject (לצורך הדוגמא נקרא לו enemyPrefab), נגדיר אותו כ-[serializefield] ונגרור את האובייקט 'אויב' מחלון הפרויקט להיכן שמופיע האובייקט שיצרנו ב-inspector של ה spawn manager. אח"כ נוסיף את פונקציה coroutine שתיצור אויב כל פרק זמן שאותו נבחר ותגריל לו מיקום חדש להתחיל ממנו(על ציר ה-x כי אנחנו מתחילים בראש המסגרת בציר ה-y). בכדי שהמתודה תמשיך עד לסוף המשחק נכניס את האיטרציות של המתודה ללולאה, כך בפעם הבאה שנקרא לה היא תתחיל מתחילת הלולאה, ולא תמשיך מyield return האחרון עד לסוף המתודה, ואז לא יהיה ניתן להשתמש בה יותר בתזמון רציף:

```
IEnumerator SpawnRoutine()
    {
        while (true)
        {

            Vector3 postospawn = new Vector3(Random.Range(-8f, 8f), 7, 0);
            GameObject new_enemy =Instantiate(_enemyPrefabs, postospawn, Quaternion.identity);
            yield return new WaitForSeconds(time);
        }
    }
```

עוד לא סיימנו. כדי לקרוא למתודה שעשינו צריך להשתמש במתודה המיוחדת StartCoroutine שמפעילה מתודות מסוג coroutine, המתודה מקבלת כפרמטר את הפונקציה עצמה אליה היא קוראת (ניתן גם לשלוח לה מחרוזת עם שם המתודה).
למתודה StartCoroutine נקרא דווקא מהפונקציה ()start היות ואין צורך לקרוא לה כל פריים, אלא היא קוראת לעצמה בכל פרק זמן החל מהפריים הראשון של המשחק:

```
void Start()
    {
        StartCoroutine("SpawnRoutine");// StartCoroutine(SpawnRoutine())is also valid.

    }
```

חלק שני

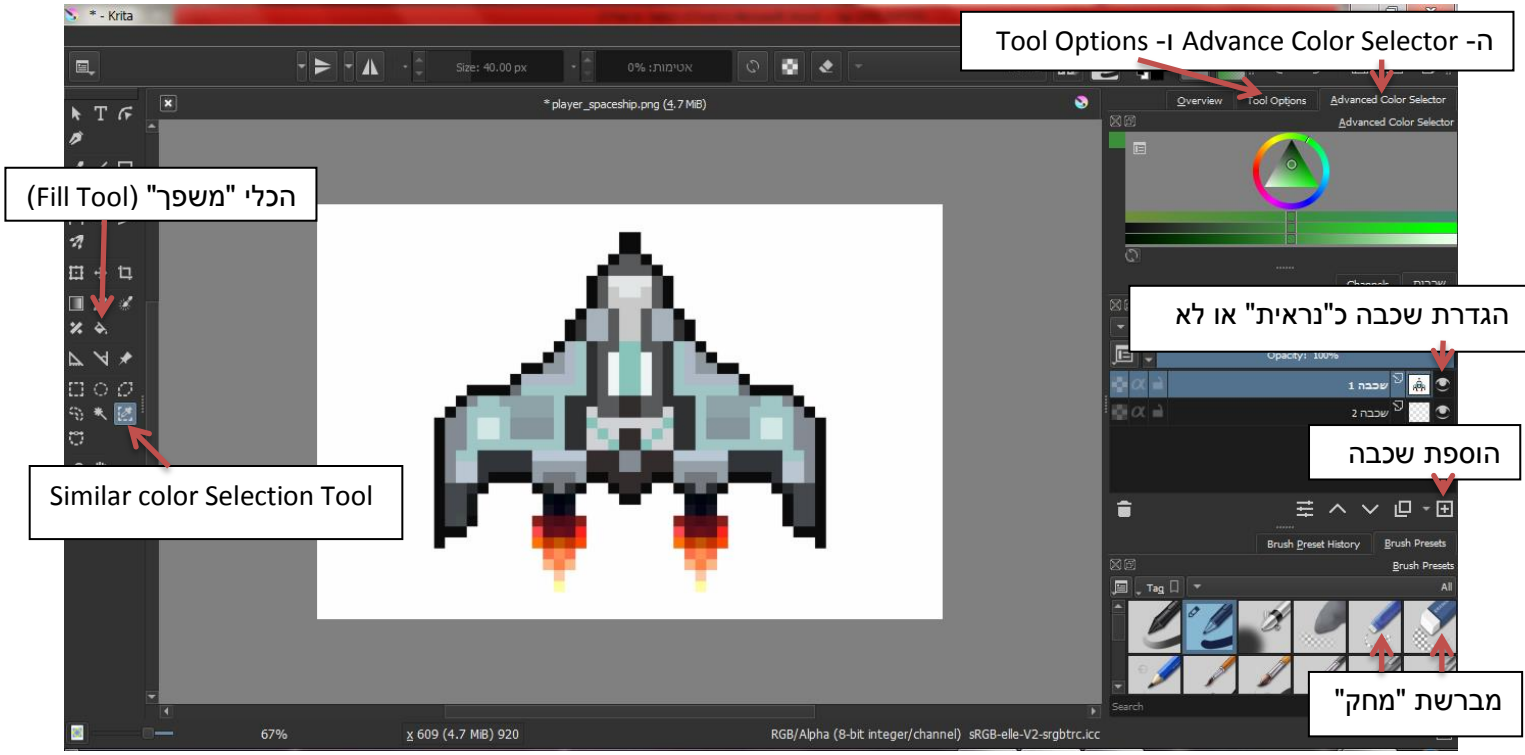
עד עכשיו התעסקנו בבניית שלד למשחק שלנו- תנועה של הדמיות, לוגיקת משחק ועיצוב קל ,כדי לדמות איך המשחק שלנו יתנהל. עכשיו נתעסק בחלק ה"אומנותי" יותר של המשחק- עיצוב דמויות ורקעים, UI ,תפריט ראשי ועוד.
מכאן נוכל לקח את המשחק שלנו לשני כיוונים- או שנמשיך עם אותו כיוון שהתחלנו אתו וניישם משחק תלת-ממדי, שהיתרונות בו ברורים-ממשיך באותו הקוד שהשתמשנו בו קודם(אין שינויים קטנים),מרהיב יותר וראליסטי יותר, אך הוא ישקול בסופו של דבר הרבה יותר. או שנשנה כיוון ונבנה את המשחק שלנו דו-ממדי, שהוא הרבה יותר קל ליישום, שוקל פחות, ומתאים ליותר פלטפורמות.
הבחירה היא בידיים שלכם ושתי הדרכים לגיטימיות, גם ההבדלים מבחינת התהליך לא גדולים במיוחד, אך לשלב הזה עדיף שנתמקד שבמשהו שיותר קל ליישם ולכן אנחנו ממשיכים כדו-ממד, אבל שוב מי שמעדיף ליישם את המשחק כתלת-ממד מומלץ לקפוץ לפרק <שם הפרק> ולהתאים באופן עצמי את החסר למשחק שלו.

מעבר מ3D ל2D-

בשביל להמיר את הדמויות שלנו לדו-ממד נצטרך assets דו-ממדיים בשביל המשחק שלנו. מומלץ ביותר לחפש אם קיימים assets שעונים לנו על רוב הדרישות בחנות של unity (unity asset store), כמעט בטוח שנמצא שם ערכה שכוללת **הכל**, החל מדמויות וכלה בעזרים כמו אודיו או אנימציות במיוחד למשחק שלנו.
חשוב לציין שרוב הasset בחנות בתשלום, אך יש מגוון ענק של asset בחינם שניתן לייבא למשחק ,וניתן למיין לפי מחירים. במידה ולא מצאנו ,או שאנחנו רוצים לייצור בעצמנו אל דאגה גם בזה נטפל. במשחק נצטרך כמה דברים שיחליפו את האובייקטים הפרימיטיביים שהשתמשנו בהם עד כה : 1) תמונת רקע- התמונה לא חייבת להיות בפורמט ספציפי, כל זמן ש-unity יכול לקרוא אותו, לרוב עדיף להשתמש בפורמט png כי הוא שומר על איכות התמונה המקורית. קל למצוא תמונות שמדמות חלל חיצון בגוגל. 2) דמויות או אובייקטים- התמונות של הדמויות\אובייקטים צריכות להיות בפורמט png ספציפית, **ובלי רקע**, לרוב קשה למצוא תמונות כאלה, אם בכלל יש, לכן נצטרך ליצור כאלה בעצמנו ע"י תוכנת גרפיקה כלשהי, המומלצות הן Photoshop או Krita , אומנם לפוטושופ יש הרבה יותר מדריכים שימושיים והיא עם תמיכה טכנית ,אך אישית אני ממליץ על krita משתי סיבות עיקריות : א) היא חינמית- היא תוכנת open source שרצה על כמה מערכות הפעלה(linux בניהן) ואין צורך ברישיון מיוחד כדי לעבוד איתה. ב) יש לה פיצ'רים במיוחד לאנימציה ועיצוב גרפי שמתאימים לבניית משחקים אינדיים (עצמאים). אז איך נפריד בין התמונה לרקע שלה? ברמת העיקרון אם אנחנו בונים בעצמנו את הדמויות אין לנו צורך בפורמט ספציפי (IPJE,PNG,GIF) ובלבד שהרקע יהיה בצבע שונה מהאובייקט שישמש אותנו.
תמונה לדוגמא:

נוריד את התמונה למחשב ונפתח אותה באמצעות קריטה. לקריטה יש כמה כלים שיכולים לעזור לנו להפריד בין הרקע לתמונה עצמה. ראשית נצטרך לדאוג שתהיה לנו שכבה חדשה מתחת לתמונה ,לכן ניצור שכבה חדשה ונגרור אותה שתהיה מתחת לשכבה של התמונה ונגדיר אותה כ"לא נראית" (אייקון שדומה למין עין בצד של האייקון של השכבה). נחזור לשכבה של התמונה ,נבחר בכלי similar color selection שמסמן בתמונה מקורית את אותם המקומות שיש להם את אותו הצבע. צריך לדאוג שה- fuzziness שלו יהיה נמוך (בין 1 ל-5),כך הוא יבדיל כמה שיותר בין גוונים שונים, אפשר לעשות את זה ב- tool option. נלחץ על הרקע עם כלי, הוא אמור לסמן לנו רק את הרקע(זה נראה כמו מלא קווים מסביב לדמות) . נמחק את מה שסימנו, או ע"י delete או ע"י מברשת "מחק" ונמחק פשוט אקטיבית את האזור המסומן. כדי לבדוק שלא התווספו לנו "שאריות מהרקע" נסמן את השכבה של התמונה כ"לא נראית", ונעבור לשכבה מתחת ונסמן אותה כ-"נראית", נבחר בכלי משפך ונתן לו צבע בולט ששונה לגמרי מהצבע של הדמות (לרוב ירוק כמו מסך ירוק בסרטים הוליוודיים). ונחזיר את התמונה להיות "נראית", במידה ויש שאריות פשוט נמחק אותם ע"י המברשת "מחק", אם אין פשוט נסמן את השכבה התחתונה כ"לא נראית" ונשמור

את התמונה כקובץ png ב- File -> save as (או ctrl+shift+s) ובחלון שנפתח נבחר ב- save as type כ-png .
זה אמור לשמור לנו את התמונה ללא הרקע שלה. באיור למטה יש לנו דמונסטרציה של הכלים של קריטה שהשתמשנו בהם.



בס"ד