הכנת power-up

Power ups-

הם אלמנטים שמוסיפים למשחק כדי לשפר את החוויה. למשל במהלך המשחק ניתן לקחת אייטמים שמחזקים את השחקן, או מוסיפים לו חיים וכדו'. אנחנו נראה שני סוגים של power-ups,כדאי מאוד להוסיף סוגים שונים כדי להעצים את חווית המשחק.

Triple shot-

ה-power up הראשון שנעשה הוא יריה משולשת, כלומר בכל פעם שהשחקן שלנו מקבל את ה Power up(או יותר נכון מתנגש אתו) הוא יוכל לירות שלוש יריות במקביל במקום שתיים.
דבר ראשון שנעשה הוא למצוא איזושהי תמונה שתייצג את ה power up עם כל השלבים הנלווים. שלב הבא יהיה לבנות את היריה המשולשת עצמה, נגרור שלושה לייזרים לחלון הסצנה ונסדר אותם יחסית ל-player. ניצור אובייקט ריק בשם triple shot ונגרור את הלייזרים אליו שייהפך להיות אובייקט האב שלהם. נגדיר את האובייקט שיצרנו כ-prefab ונמחק אותו מהיררכיה (hierarchy view).
לפני שניכנס לקוד כדי שנבין מה התהליך שנרצה שיקרה עם קבלת ה power up- אנחנו רוצים שהאובייקט 'יילקח' ע"י השחקן, הדבר שמדמה לנו לקיחה, כפי שראינו בפרקים הקודמים, הוא בעצם "התנגשות" של האובייקטים, כלומר מהרגע שהופעל הטריגר של אחד האובייקטים (שקלט שהייתה כאן התנגשות)האובייקט יפעיל איזשהי מתודה או משנה של השחקן שיאותת לו שהחל מעכשיו הוא ישתמש ביריה המשולשת במקום ביריה רגילה. ברמת השחקן נצטרך את הדברים הבאים: 1) משתנה בוליאני שמסמן אם עכשיו יורים יריה משולשת. 2)משתנה עצם מסוג יריה משולשת שאותו הוא יאתחל בכל פעם שנלחץ על מקש ספציפי במקלדת. 3)מתודה שתפעיל מתודת IEnumerator למשך זמן שבו הערך של המשתנה הבוליאני יהיה אמת, עד שייגמר הזמן והמשתנה יחזור להיות שקר.
לא קשה להבין אם ככה איך ליישם את זה מבחינת סינטקס:

```
[SerializeField]
    private GameObject _tripleShot;
    private bool _tripleShotActive = false;
```

ובמתודה shoot() או ב update() (תלוי אם הכנסתם את הקוד של הירי למתודה בפני עצמה והפעלתם אותה ב update) , נעשה את השינויים נעשה תנאי: אם TripleShotActive חיוב, ניתן ווקטור כיוון ליריה המשולשת, ונשתמש ב-instantiate כדי לאתחל האובייקט, אחרת נעשה את אותה פרוצדורה שעשינו עד עכשיו עם יריה אחת:

```
if (!_tripleShotActive)
            {
                Vector3 laser_position = new Vector3(transform.position.x, transform.position.y + 1, 0);
                Instantiate(laser, laser_position, Quaternion.identity);
            }
            else
            {
                Vector3 laser_position = new Vector3(transform.position.x, transform.position.y + 1, 0);
                Instantiate(_tripleShot, laser_position, Quaternion.identity);
{
```

(ייתכן ונצטרך לאתחל את היריה המשולשת במרחק יותר גדול מהשחקן).
עכשיו נצטרך להוסיף את המתודה הייעודית ליריה המשולשת, לצורך הדוגמא נחכה חמש שניות עד שנגמרת היריה:

```
public void TripleShotActive()
    {
        _tripleShotActive = true;
```

```
        StartCoroutine(TripleShotRoutine());
    }

    IEnumerator TripleShotRoutine()
    {
        yield return new WaitForSeconds(5f);
        _tripleShotActive = false;
{
```

נחזור לגוף ה power up שלנו אנחנו צריכים להוסיף לו רכיבים שמאפשרים התנגשות כלומר collider2D כלשהו , rigidbody וכמובן סקריפט שעליו הוא ירוץ. לאחר שהוספנו לו את הרכיבים הרלוונטיים נתמקד בקוד שלו.
דבר ראשון אנחנו רוצים שה power up שלנו ינוע בקו ישר כלפי מטה, ובניגוד לאויביים שמגיחים ממקומות שונים על המסך, ה power up יופיע רק עד שהוא חוצה את המסך ואז יושמד.
היות וכבר ראינו איך לעשות את זה כל כך הרבה נראה לי שמיותר לציין את זה כאן. מי שעדיין לא זוכר כדאי לו להסתכל בפרק על הלייזר. בדיוק כמו שעשינו אם האוייב גם כאן נשתמש בפונקציה המפורסמת OnTriggerEnter2D ובדיוק כמו באויב גם כאן נברר תחילה אם ל-other יש את התג של השחקן, במידה וכן נרצה לקבל את הרכיב של הסקריפט מ other ולהפעיל את המתודה tripleshotactive() של השחקן. ולאחר שהפעלנו נשמיד את ה power up כי כבר השתמשנו בו:

```
private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            Destroy(this.gameObject);
            Player player = other.transform.GetComponent<Player>();
            if (player != null)
                player.TripleShotActive();
            Destroy(this.gameObject);
        }

{
```

אם נריץ את המשחק ונוודא שהכל כשורה, נראה שעדיין יש לנו בעיה אחת שמציקה לנו- האובייקטים של הירייה המושלשת עדיין מופיעים לנו על חלון ההיררכיה. זה משום שהרסנו את הלייזר, אבל לא את האבא שלו, כלומר ה power up עצמו של הירייה המשולשת. לשם כך נצטרך לחזור לסקריפט של הלייזר ולבדוק האם יש לנו גם אובייקט אב ללייזר, במידה וכן נשמיד את האובייקט האב ביחד עם הלייזר כאשר הוא יוצא מגבולות המסך:

```
void Update()
    {

        transform.Translate(Vector3.up * Time.deltaTime*_speed);
        if (transform.position.y>8.0f)
        {
            if (transform.parent != null)
            {
                Destroy(transform.parent.gameObject);
            }
            Destroy(this.gameObject);
{
{
```

Spawn power up-

כמו שיצרנו מתזמן לאויב נוכל ליצור גם ל.power up למעשה נוכל להשתמש באותו מתזמן שהשמשנו בו לפני, מה שנצטרך זה להוסיף משתנה עצם של power up ופונקצית coroutine חדשה שתרוץ פעם בכמה שניות כמו שעשינו ל enemy.
בשביל לעשות את זה מעניין אפילו יותר, נדאג שהקריאה למתודה תהיה באופן רנדומלי בין 3 ל- 8 שניות:

```
IEnumerator SpawnPowerUpRoutine()
    {
        while (true)
        {
            Vector3 postospawn = new Vector3(Random.Range(negative, positive), 13, 0);
```

```
            GameObject new_powerUp = Instantiate(_powerUpPrefabs, postospawn, Quaternion.identity);
            yield return new WaitForSeconds(Random.Range(3, 8));
        }
{
```

וכמובן לא לשכוח לעדכן את start להפעיל את הפונקציה.

מגן-

ה power up השני שנעשה הוא מגן- כל פעם שניקח אותו תהיה לנו הגנה למשך זמן מוגבל מפני התנגשויות מאויבים.
בהתחלה נבצע את אותם צעדים שעשינו עם ה-power up הקודם: נמצא לו תמונה, נתאים אותה למסך, נוסיף לו collider2D ו-rigidbody2D מתאים, נוודא שה-gravity שווה אפס, וה-is trigger מסומן.
לפני שנתעסק בכל התהליך של ה'מגן' נשנה את ה-spawn manager. אין סיבה באמת שנעשה מתודת coroutine חדשה עבור כל power up חדש שנוסיף, זה לא יעיל, ובעיני לאסוף כמה power ups במקביל הורס קצת את החוויה. אז למה שלא נשתמש באותה מתודה שאיתה אנחנו מתזמנים את ה-power ups כך שתגריל איזשהו אחד כל כמה שניות.
אם ככה נצטרך לשנות את המשתנה power up למערך של game object שמכיל את כל סוגי ה power ups שיש לנו.
נעשה את זה ונראה עכשיו שקיבלנו הערה מה visual studio היכן שאנחנו מאתחלים את ה power up. נסמן אותה כהערה(עם שני קווים אלכסונים) בינתיים ונחזור ל unity. נשים לב שכשיו ב inspector של ה spawn manager מופיע לנו המשתנה Power up עם אפשרות להגדרת גודל. בינתיים יש לנו שני power ups לכן נקבע את הגודל ל-2, ובהמשך אם נחליט להוסיף נגדיל את הגודל.
נגרור את האובייקטים למקום המתאים להם ב-inspector ונחזור לקוד. כעת מטרתנו היא שנאתחל power up אחד מתוך המערך כל פעם, ושנעשה את זה באופן אקראי. במילים אחרות אנחנו צרכים איזשהו משתנה אינטג'רי שמוגרל בין 0 לגודל המערך (או קרוב לגודל המערך), ואז נאתחל את האובייקט של המערך שנמצא במקום של הערך שיצא לנו.
מבחינת לוגיקת המגן צריך שיקרו שני דברים: 1) מתי שהשחקן לוקח את ה power up מופיעה תמונה של מגן ,או כל דבר שמסמל שהסטטוס של השחקן השתנה ועכשיו הוא חסין אויבים באופן זמני. 2)לדאוג מבחינת קוד שהוא לא יפגע, אבל אויבים יפגעו ממנו במשך כמה שניות מהאויבים.
ראשית נתעסק בחלק השני כי הוא יותר קל ליישום, וכבר ראינו דבר דומה עם היריה המשולשת. נצטרך להוסיף איזשהו משתנה בוליאני כך שמסמן לנו בקוד שעכשיו אנחנו במצב 'מגן' ומתודה damage() נוודא שאם הוא מופעל (עם ערך 'אמת') אז לא ירדו לנו חיים כלומר: if(_isShilded) return; (ואז שאר הקוד עם life - -וכו').
ובדיוק כמו שעשינו עם היריה המשולשת גם כאן נוסיף מתודת coroutine בשביל להגביל את זמן השימוש במגן.
באשר לקוד של המגן- ברמת העיקרון אין באמת צורך לבנות סקריפט חדש במיוחד למגן, נוכל פשוט להוסיף לקוד הישן של הסקריפט power up .
נגרור את הסקריפט לאובייקט 'מגן' ונערוך את הסקריפט באופן הבא: לכל סוג של power up ניתן איזשהו id ייחודי, שכן של כל אחד מפעיל מתודה אחרת של השחקן. לכן נייצר משתנה אינטג'רי חדש ונקרא לו _id, נדאג שנוכל לראות אותו ב unity.
במתודה ontriggerEnter נצטרך לחלק למצבים לפי ה-id של ה power up. יש כמה שיטות לעשות את זה, הקלה שבהם היא ע"י switch ו-case (לאו דווקא השיטה החכמה שבניהם). מבחינת סינטקס זה יראה כך:

```
private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            Destroy(this.gameObject);
            Player player = other.transform.GetComponent<Player>();
```

```
        if (player != null)
    {
        Switch(_id)
        {
          case 0:
            player.TripleShotActive();
           break;

          Case 1:
           player.ShiledActive();
           break;
        }
    }
     Destroy(this.gameObject);
     }

{
```

ועכשיו נצטרך לעדכן ב unity את ה id בהתאם.
לאחר שהגדרנו את השחקן כראוי נחזור לחלק הראשון- להוסיף תמונה שתתווסף לשחקן כאשר אנחנו לוקחים את המגן.
נמצא תמונה לייצוג המגן, נערוך אותה כך שתהיה נחשבת sprite, נמקם אותה בשכבה המתאימה ובמיקום המתאים יחסית לשחקן שלנו ונגרור אותה לחלון ההיררכיה. כדי שהתמונה תזוז ביחד עם השחקן היא צריכה להיות אובייקט 'בן' לשחקן, לכן נגרור את אובייקט התמונה לתוך אובייקט השחקן.
ברמת העיקרון מה שאנחנו רוצים לממש זה שכאשר לקחנו מגן אז התמונה שמייצגת אותו תופיע על המסך, וכאשר נגמר הזמן של המגן אנחנו חוזרים למצב שהתמונה לא קיימת. אם נשים לב ב inspector למעלה יש ריבוע קטן ליד שם האובייקט, הריבוע הזה הוא משתנה בוליאני 'Active' של האובייקט, הוא מסמן האם האובייקט פועל עכשיו או לא. מה שאנחנו רוצים לעשות זה בעצם להפעיל את המשתנה הזה של התמונה שמייצגת את המגן, כלומר לתת לו ערך חיובי כאשר מופעל המגן, ולבטל אותו (לתת לו ערך שלילי) כאשר הוא מסיים את העבודה שלו. ל unity יש מתודה מיוחדת שמביאה ערך לאותו משתנה:
Gameobject.SetActive(bool status). בנתיים נסמן את האובייקט **כלא** אקטיבי. נצטרך כמובן גם לידע את השחקן שיש אובייקט בן כזה ,כיצד נעשה את זה? כמו שעשינו עד עכשיו: ניצור משתנה עצם מסוג Gameobject ונגרור אליו את האובייקט בין באינספקטור. לאחר שהשחקן מודע לקיומו של ה"מגן" ניתן להפעיל עליו את ה-setActive כך:
נניח קראנו למשתנה שלנו shiled_pic, אז צריך, כדי להפעיל את המגן, את
shiled_pic.Gameobject.SetActive(true) נעשה את זה במתודת coroutine של השחקן שמפעילה את המגן, ו'נכבה' אותו בסוף המתודה.