

# מבוא לממשק-משתמש (UI)



ממשק משתמש או User Interface הוא המרחב שבו מתנהלת התקשורת בין החלק התכנותי של המכונה לחלק הנשלט בידי אדם. בכלליות המטרה של עיצוב ופיתוח ממשק משתמש היא לייצר תקשורת קלה, יעילה ומהנה (User friendly) שמפעילה את המכונה בדרך הרצויה. בפיתוח משחקים המושג בדר"כ מתקשר עם כל התצוגה על המסך שלא קשורה ישירות לאובייקטים הפועלים במשחק, למשל כפתורים- הכפתורים מתחברים למשחק במין שכבה מעל שמפעילה את המנגנון בפנים, לעומת זאת inputs מהמקלדת לא נחשבים ממשק משתמש כי המשתמש לא רואה אותם על המסך(רק את התוצאה שלהם). גם דברים שלא ניתן לשלוט עליהם אך מקלים על המשתמש כגון תצוגה של נקודות בצד המסך וכדו' נחשבים ל-UI כי הם משפרים את חווית המשתמש.

ב unity השימוש ב-UI לא מסובך, למעשה יש כאלה שבונים אפליקציות שלמות דרך unity משום ש-unity פשוט מהרבה תוכנות עיצוביות ייעודיות לאפליקציות (אין צורך בשימוש ב xml), מתאים את עצמו לכמה פלטפורמות בקלות, ונח מאוד לתכנת בו.

## -Score text

בחלק הזה נתחיל לבנות את ה-UI הבסיסי. נרצה לבנות איזושהי גרפיקה שתציג את הנקודות שלנו במשחק, כמה חיים נשאר לנו ותפריט ראשי.

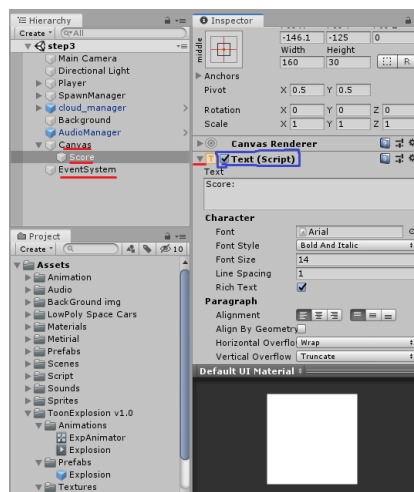
כדי שתהיה לנו אפשרות בכלל לבנות UI נצטרך קנבס (canvas) שעליו נציג. אם נלחץ על מקש ימני בחלון ההיררכיה ניתן לראות שיש את התת-אפשרות UI, שם ניתן לראות את כל האפשרויות שנכללות בתוך UI-טקסט, כפתורים וכדו'.


כבסיס ניצור משתנה שיציג לנו את כמות הניקוד שיש לנו- כל פעם שחיסלנו אויב נקבל נקודה. נלחץ מקש ימני-> UI-TEXT.

כאשר הוספנו את האובייקט למסך נוסף לנו עוד אובייקט אב לחלון ההיררכיה שקוראים לו canvas, הוא למעשה כמין מסך ריק שעליו "מדביקים" את אובייקטי ה-UI שניצור. דבר נוסף שמופיע לנו במסך ההיררכיה הוא אובייקט מסוג Event System, או בקיצור ES, הוא אובייקט שמאפשר לנו לתקשר עם ה-UI באמצעות "מאורעות" (דוגמא למאורע היא לחיצה על כפתור).

נשנה באינספקטור של הטקסט את הלשונית ברכיב טקסט (איפה אפשרות לשנות את הצבע הגודל וכו':

שם האובייקט ל-score, ונפתח את שכתוב (text(script) נראה שיש לנו הדיפולטיבי של הטקסט, את הפונט,



ניתן לשנות את מיקום הטקסט על הקנבס באמצעות חצי ההזזה שנמצאים מצד שמאל למעלה. נשים לב שאם נקטין את גודל המסך של המשחק הטקסט לא יתאים לגבולות החדשים, זה משום שלא התאמנו את האובייקט למסך הראשי. אם נסתכל על האינספקטור של הטקסט נבחין כי יש לנו רכיב מסוג Rect Transform שהוא אחראי על התצוגה של  האובייקט יחסית למסך המשחק, ברכיב שי לנו אייקון של ריבוע עם צלב באמצע:

הוא מסמן לנו היכן ממוקם כעת הטקסט יחסית למסך. אנחנו רוצים לשנות אותו ככה שיתאים למיקום שהצבנו לו- אם למשל הצבנו את הטקסט בפינה הימנית למעלה למטה נצטרך לשנות למצב המתאים לכך, בשביל לשנות את המצב הדיפולטיבי נלחץ על האייקון יפתח לנו חלון ה-Anchor Preset, ונבחר מתוכו את המצב שנראה לנו הכי מתאים. לצורך הדוגמא, אם אנחנו רוצים למקם את הטקסט בפינה הימנית העליונה נבחר ב- top ובתוכו נבחר ב-Right. הצלחנו למקם את הטקסט באזור מסוים אך עדיין גודל הטקסט נשאר קבוע כשאנחנו מגדילים ומקטינים את המסך. כדי לתקן את זה נלך לאובייקט קנבס, בחלון האינספקטור שלו מופיע רכיב Canvas Scaler, כברירת מחדל הוא מסומן כ- Constant Pixel Size, אנחנו צריכים לשנות אתו ל- **Scale With Screen Size**, ועכשיו הוא מתאים את עצמו לגודל המסך של המשחק.

בינתיים הטקסט שלנו מוגדר להיות "score" בלי שינוי מהותי, כדי שנוכל לשנות אותו תוך כדי משחק נצטרך לתכנן כמה דברים:

(1) לדאוג שהשחקן ישמור ניקוד עבור כל אויב שהוא חיסל. (2) שהקנבס יעדכן את הטקסט לניקוד הנוכחי של השחקן. תחילה נוסיף לשחקן מתודה חדשה void AddScore() שתוסיף ניקוד למשתנה עצם score\_ מסוג int כפי שראינו כבר כמה פעמים במהלך המדריך. ניכנס לקוד של האויב: האויב צריך לקרוא למתודה AddScore() של השחקן בכל פעם שמחסלים אותו בין במגע עם השחקן ובין ע"י הלייזר. כבר ראינו כיצד לבקש אובייקט משחק ספציפי, אך אם נבקש בכל פעם שנפגע אויב את האובייקט player נבצע פעולה יקרה, לכן עדיף לנו לשמר אובייקט מסוג Player כמשתנה עצם ולא לחלל אותו במתודה start() ואז מתי שהאויב מתנגש עם הלייזר הוא משתמש במתודה של האובייקט עצם שלו. מבחינת סינטקס זה יראה כך:

```
[SerializeField]
private Player _player;

void Start()
{
    _player = GameObject.Find("Player").GetComponent<Player>();
    . . .
}

. . .

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

חזרה לקנבס, בכדי לעדכן אותו בכל פעם שעלה הניקוד נצטרך שיהיה לו איזשהו סקריפט שינחה אותו, נוסיף סקריפט UIManager וניכנס אליו. אם ננסה להוסיף משתנה מסוג טקסט כנראה שלא נמצא אותו המאגר של visual studio, וזאת משום שכדי להשתמש באובייקטים של UI נצטרך להשתמש במרחב שם מיוחד של unity לשם כך- UnityEngine.UI.

אחר שהוספנו את מרחב השם אנחנו יכולים סוף סוף להוסיף את האובייקט טקסט שנקרא לו score\_. נגרוור ב unity את האובייקט טקסט score למקום המתאים באינספקטור של ה-canvas.

נרצה לאתחל את האובייקט שלנו כך שהניקוד ההתחלתי שלנו יהיה אפס לכן במתודה start() נגדיר את המשתנה כך:  
;score.text = "score" + 0\_  
ובשביל שנוכל להתעדכן בזמן אמת עם ה-player נצטרך מתודה שתשנה את אותו משתנה עצם בזמן אמת:

```
public void ScoreView(int Point)
{
    _score.text = "score: " + Point;
}
```

נעבור לקוד של השחקן. נצטרך משתנה עצם מסוג UIManager כדי להפעיל את המתודה ScoreView.  
יש לשים לב שהרכיב UIManager נמצא בתוך האובייקט Canvas, לכן אם אנחנו רוצים לאתחל אותו נצרך למצוא את האובייקט Canvas וממנו לבקש את הרכיב UIManager:

```
[SerializeField]
private UIManager _UImanager;
...
void Start()
{
    _score = 0;
    _UImanager = GameObject.Find("Canvas").GetComponent<UIManager>();
}
```

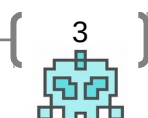
ועכשיו כל מה שנצטרך זה להפעיל את המתודה ScoreViewer ב- AddScore():

```
public void AddScore()
{
    _score+=10;
    _UImanager.ScoreView(_score);
}
```

## -Live sprite

בדיוק כמו שעשינו עם הניקוד, אנחנו שואפים שיהיה לנו איזשהו אינדיקטור לכמה חיים נשארו לנו.  
אבל בניגוד לניקוד שהיה עשוי מטקסט, עכשיו אנחנו עובדים על אובייקט תמונה. נוסיף אובייקט תמונה לקנבס במקש ימני -> image UI, נקרא לו בשם מתאים ונגרור את התמונה לתוך ה-image source באינספקטור של האובייקט שיצרנו הרגע.  
נמקם את הספריט במקום שנראה לנו מתאים בקנבס. אם הצורה של התמונה לא תואמת לאספקט של התמונה המקורית ניתן לשנות את זה ע"י בחירה ב- Preserve Aspect ברכיב image. כמוכן שנשכפל את התמונה לפי כמות החיים של השחקן וכל תמונה נמקם אחת ליד התמונה האחרת.  
נעברו לקוד: אנחנו צריכים ב- UI Manager שיהיה לנו מערך של התמונות (אובייקט מסוג Image), כך שלפי מצב החיים של השחקן המערך "יפעיל" את התמונות, למשל אם לשחקן שני חיים נשנה את ה- is active של אחת מהתמונות הקיצוניות ל- false, לשם כך ניצור מתודה חדשה שאחראית לבטל את הספרייטים לפי החיים של השחקן.  
בתמונות של הקנבס, בניגוד לאובייקט-משחק אחרים, כדי לבטל את הפעולה של אותו רכיב משתמשים במשתנה enable של אותו אובייקט. מבחינת סינטקס זה יראה כך:

```
[SerializeField]
private Image[] _lives;
...
public void LifeScore(int lives)
{
    if (lives < 3)
    {
        _lives[lives].enabled = false;
    }
}
```



ובקוד של השחקן, היות וכבר יש לנו משתנה מסוג UI manager נוכל להשתמש בו במתודה damage() כך שמתי שירידו לו נקודות הוא יעדכן את הקנבס להוריד תמונה אחת מתוך המערך:

```
public void Damage()
{
    life--;
    _UImanager.LifeScore(life);
    if(life<1)
    {
        StartCoroutine(Explosion());
    }
}
```

הערה: המתודה שיצרנו לוקחת בחשבון שלא תהיה תוספת חיים במהלך המשחק- אם ירדו לך חיים אז אתה נשאר עם אותם נקודות עד לפגיעה הבאה או סוף המשחק. במידה ואתם רוצים להוסיף power-up יש לממש את המתודה אחרת.

## -Game over Text

הפסדנו את כל נקודות החיים שלנו ונגמר המשחק, לא יהיה יותר נחמד אילו יכולנו להוסיף את הטקסט הכל כך "כייפי" Game Over ? לאחר שכבר ראינו איך ליצור טקסט לניקוד יהיה לנו פשוט לפענח איך לעשות את זה גם בטקסט -over

ניצור אובייקט טקסטGameOver חדש למסך ונכתוב את הטקסט הרצוי, נוסיף ל-UI Manager משתנה מסוג טקסט ונגרור אליו את האובייקטGameOver שיצרנו. ניצור מתודה שתציג את האובייקט על המסך, אם נרצה לעשות את זה בסטייל נדאג שהמתודה תפעיל מתודת coroutine שתפעיל את הטקסט כל שניה ותכבה אותו אחרי שניה, זה יצור אפקט RETRO.

בקוד של השחקן נוסיף במתודה damage() להפעיל את אותה מתודה של ה-UI Manager שמפעילה את האפקט הפליקינג:

```
public void Damage()
{
    life--;
    _UImanager.LifeScore(life);
    if(life<1)
    {
        StartCoroutine(Explosion());
        _UImanager.GameOverEnable();
    }
}
```

