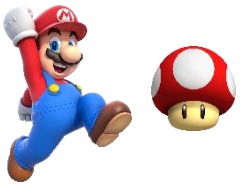


מבוא לתכנות משחקים ב3D

נקדים ונאמר שנכון להיום, מומלץ להשתמש בגרסא 2020.1.0a ומעלה של Unity לחלק זה של הקורס.
זאת מכיוון שנמצאו באגים בגרסאות קודמות בנוגע לתכנות משחק תלת מימדי.

מהו משחק 2.5D-

כהקדמה לשלב התכנות התלת מימדי של הקורס, ניצור משחק בסיסי ב2.5 מימדים. משחק 2.5 מימדים הינו משחק המשלב מצד אחד אובייקטים תלת מימדיים ומאידך הוא מוצג כמשחק דו מימדי.
דוגמא טובה למשחקים מסוג זה יהיו משחקי לחימה.
כיום הרבה ממשחקי פלטפורמת הדו מימד משלבים בהם אלמנטים של תלת מימד (דוגמא: סדרת המשחקים החדשה של סופר מריו), לכן חשוב לדעת לשלב את שתי הפלטפורמות יחדיו.

תכנון סצנה-

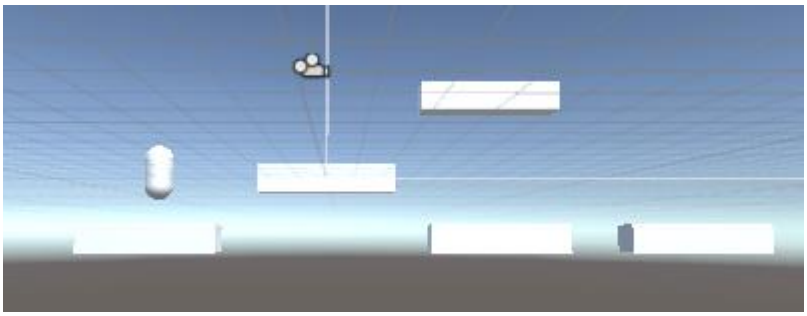
במשחקי תלת מימד, השימוש באלמנטים קבועים מראש הנטענים עם הסצנה, נפוץ יותר(אלמנטים כאלה יכולים להיות Power-up-ים, המשטח שעליו משחקים או המפה עצמה).
לכן חלוקה נכונה ומסודרת יותר של סצנה היא ע"י יצירת אובייקט ריק שמכיל את כל האלמנטים המרכזיים של השלב שאנו רוצים שיהיו קיימים כבר בשעת טעינת הסצנה. לצורך המשחק שאנו רוצים לבנות, נקרא לאלמנט ריק זה "Level".

1.ניצור גם שחקן (Player)מאלמנט copsule, והגדרותיו יהיו כמו שלמדתם בשיעורים הקודמים. אם האלמנט מגיע עם RigidBody הסירו אותו כי אנו הולכים ליצור לשחקן גרביטציה ותזוזה משלנו.

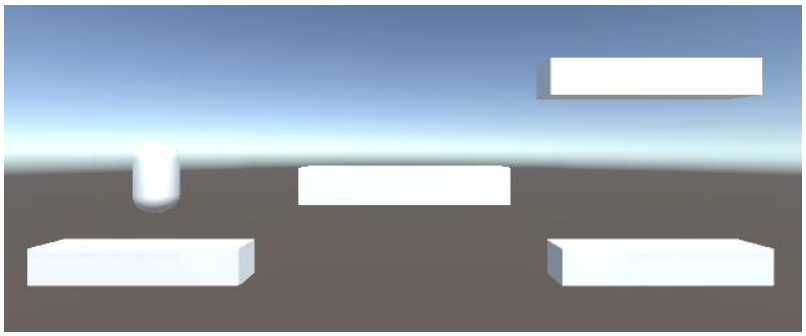
2.לפני שנתקדם, נשנה את זווית הראייה שלנו למישורי ה-x וה-y. ומעתה עד סוף המשחק נשמור שכל אובייקט שניצור כולל השחקן יהיה ממוקם ב0 על ציר ה-z.

3.כעת ניצור כמה שטחים או פלטפורמות עליהן יוכל השחקן ללכת.
נעשה זאת ע"י יצירת אובייקט Cube, ניצור ממנו prefab (ע"י הוספה לתיקיית Prefabs) ונחליט על גודל רצוי של הפלטפורמה על ידי כך שנשחק עם ציר ה-x שבתגית הtransform שבinspector.
כעת נשכפל את הפלטפורמה שבנינו כ5 פעמים.(שימו לב לתת שם לכל האלמנטים ולשמור על הכללים שלמדנו בבניית משחק דו מימדי).

השתדלו להגיע למצב דומה לזה שבתמונה:



איך זה נראה על המסך של המשחק:



את הפלטפורמות נשמור בתוך אובייקט ריק שנקרא לו Level כמו שהוסבר קודם לכן. (מקמו אותו כך שהtransform שלו יהיה 0 בx,y,z לצורך נוחות). אתם יכולים גם להחליט על נקודת התתחלה של השחקן בשלב זה.

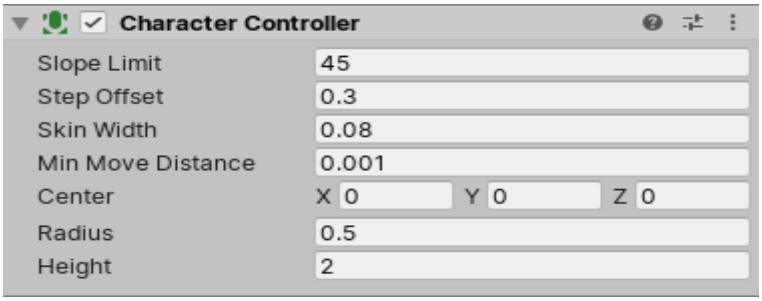
*לאחר שנלמד איך ליצור אובייקט collectable, נוסיף אותו לחלק מהאובייקטים הנכללים בשלב והוא אובייקט חיוני לתכנון סצנה.

פיזיקת משתמש (Character controller)-

בשונה ממשחק החלליות שיצרנו, במשחק שלנו אנו נרצה ליצור פיזיקה מסובכת יותר לתזוזת השחקן. אנו נוסיף אלמנטים כמו קפיצה, כח כבידה (אפשר להשתמש בUse Gravity שבrigid body, אבל אנחנו נרצה לעשות מניפולציות בכח הכבידה) ועוד בהמשך. לכן אנו רוצים ממשק פתוח להגדרת התזוזה של השחקן. אובייקט Character controller מאפשר לנו לעשות זאת בצורה נוחה, ובנוסף מעניק אפשרויות רחבות יותר מאשר התכנות בקוד שלמדנו עד כה.

אובייקט Character controller-

מאפשר לנו לשלוט בתנועה בקלות המוגבלת על ידי התנגשויות וטריגרים ללא הצורך בrigid body וcollider.
בשימוש אובייקט זה, השחקן שלנו לא יושפע על ידי כוחות אחרים ויזוז רק כאשר אנו נגיד לו. כל התזוזות שלו יהיו על ידי קלט מהשחקן או התנגשויות.
הרחבה על המאפיינים של האובייקט ניתן למצוא כאן: https://docs.unity3d.com/ScriptReference/CharacterController.html



מאפיינים מרכזיים של Character controller:

פונקציית **isGrounded**: פונקצייה בוליאנית הבודקת האם השחקן שלנו נמצא באוויר (false) או על משטח כלשהו (true).

פונקציית **Move**: כאשר נרצה להזיז את השחקן נשתמש בפונקציית move של הcharacter controller, נדאג לכך שנבצע מראש את כל החישובים בנוגע לכיוון שנרצה להזיז אליו את השחקן בכל פריים ואת הנתונים האלו נעדכן בפונקציית הmove.
למשל, אם נרצה להחליט שכאשר המשתמש לוחץ על מקש הspace השחקן יקפוץ, נוסיף לשחקן בפריים הבא מספר חיובי מסויים לציר הy כך שבפריימים הבאים הוא ישאף להגיע עד נקודה זו, ובכך יצרנו התנהגות פיזית של קפיצה.

דוגמא לקוד דומה:

public class ExampleClass : MonoBehaviour

```
{
    CharacterController characterController;

    private float _speed = 6.0f;
    private float _jumpHighet = 8.0f;
    private float _gravity = 1.0f;
    private float _yVelocity;
    void Start()
    {
        characterController = GetComponent<CharacterController>();
    }

    void Update()
    {
{
        float horizontalInput = Input.GetAxis("Horizontal");
        Vector3 direction = new Vector3(horizontalInput, 0, 0);
        Vector3 velocity = direction * _speed;//physical representation of the player movement.

        if(_controller.isGrounded==true)
        {
            if (Input.GetKeyDown(KeyCode.Space))
            {
                _yVelocity = _jumpHighet;
            }
        }
        else
        {
            _yVelocity -= _gravity;
        }
        velocity.y = _yVelocity;
        _controller.Move(velocity * Time.deltaTime);
    }
}
```

נשים לב שקודם יצרנו אובייקט vector3 שמכיל את הכיוון שלנו כמו במשחק חלליות, אך פה הוספנו לציר ה-y ערך חיובי כאשר המשתמש לחץ על מקש הspace, או הוספנו ערך שלילי לציר ה-y שידמה כח משיכה בכל זמן אחר. רק לאחר שסיימנו את החישובים בהתאם למצב הקיים, הכנסנו את הנתונים הרלוונטיים לפונקצייה כvector3.

שאלה: שימו לב שכאשר הוחלט להוסיף ערכים לציר הy של הוקטור velocity, היה צורך ביצירת משתנה מיוחד (_yvelocity) שיאכסן בתוכו את הערך של y, אחרת במקרים אחרים (למשל בתזוזה מהירה מאוד של השחקן) הקוד לא היה עובד. למה זה כך?

מצלמה במשחקי תלת מימד

במשחקי תלת מימד המרחב שלנו לעיתים נמצא מעבר לגבולות המסך. הדרך שלנו להגיע לגבולות האלה היא עם השחקן. לכן על מנת לראות מה שהשחקן רואה, בכל משחק תלת מימד, נשרשר את המצלמה לשחקן על ידי הנחת המצלמה על אובייקט השחקן.

בהמשך נרחיב על שימושים נוספים במצלמה במשחקי תלת מימד.

Trigger Events – הפעלת מאורעות

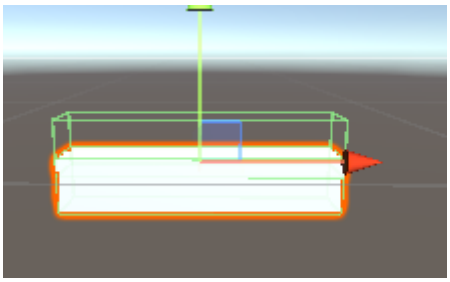
לעיתים נרצה ליצור אובייקט שבעת התנגשות, הוא יחל מאורע מסוים.
לדוגמא: מטבע במשחק שניתן לאסוף, דלת שניתן לפתוח, אזור מסוים במפה שבהגיעו אליו יגרום להתחלה של סצנה חדשה.

איך ניצור אזור שהוא טריגר?

על ידי יצירת Cube Object חדש עם collider מתאים וrigidBody ללא גרביטציה, וסימון הmesh renderer כ-False(מחיקת V באינספקטור על האובייקט של הmesh renderer).
נוכל להשתמש באובייקט זה כדי: לחסום שחקן מלהתקדם הלאה במפה, לסמן אזור שבו השחקן "מת" וכו'.

כעת ננסה להשתמש בטריגר כדי ליצור קשר בין שחקן לפלטפורמה זזה.

Moving Platform
ניצור פלטפורמה חדשה מהprefab שקיים לנו בתיקייה, ונקרא לו Moving_Platform.
לאחר שוודאנו כי יש לו collider חדש וrigidbody ללא גרביטציה, נוסיף לו עוד collider חדש מאותו הסוג ונמקם אותו קצת מעל הפלטפורמה בצורה הבאה:



המטרה היא, שכאשר השחקן יכנס לתוך אזור הטריגר, נרצה לשרשר את השחקן לתזוזת הפלטפורמה כך שהשחקן לא ייפול ממנה.

סקריפט- כעת ניצור את ההתנהגות של הפלטפורמה הזזה.
ניצור אובייקטים `private Transform _A, _B;`
קודם נאפשר את אובייקטים אלה כ-SerializeFiled.
בinspector נעדכן את המיקומים. כאשר מיקום A הוא המיקום של הפלטפורמה עכשיו, ולאחר מכן נזיז את הפלטפורמה אל נקודה מסוימת שנרצה שהיא תזוז (למעלה או קדימה, לצורך ההקדמה נבחר בקדימה) ונקרא לה מיקום B.

FixedUpdate()-

במקום להשתמש בפונקציית הupdate הרגילה שלנו, אנו נשתמש בפונקציית fixedUpdate() אשר מטרתה להתייחס למאורעות שונים אשר מעורבת בהם פיזיקה ממשית.

במקרה שלנו, אנו רוצים לשרשר שחקן אל הפלטפורמה. נרצה שהתזוזה של השחקן תהיה טבעית ומותאמת לתזוזה של הפלטפורמה שלנו, ולכן נצטרך להתגבר על כך שלשחקן שלנו יש גרביטציה ופיזיקה מיוחדת משלו.
נעשה זאת ע"י שימוש ב()FixedUpdate.

עוד בהרחבה על הנושא: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html

כעת ניצור את פונקציית העדכון שלנו: נרצה להזיז אותו מצד לצד, האימפלמנטציה היא פשוטה-

```
   void FixedUpdate()
    {
        if (_direction == false)
        {
            transform.position = Vector3.MoveTowards(transform.position, _B.position, _speed *
Time.deltaTime);
        }

        else if (_direction == true)
        {
            transform.position = Vector3.MoveTowards(transform.position, _A.position, _speed *
Time.deltaTime);
        }

        if (transform.position==_A.position)
        {
            _direction = false;
        }
        else if(transform.position== _B.position)
        {
            _direction = true;
        }
    }
```

כעת נשאר לשרשר את השחקן לפלטפורמה.
נשתמש בפונקציית הOnTriggerEnter :

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == player)
        {
           other.transform.parent = this.transform;
        }
    }
```

OnTriggerExit()

הינה פונקציה מקבילה לOnTriggerEnter אשר מופעלת לאחר מאורע יציאה מאזור של Trigger.

במקרה שלנו נרצה לבטל את השרשור שיצרנו בין השחקן לפלטפורמה הזזה:

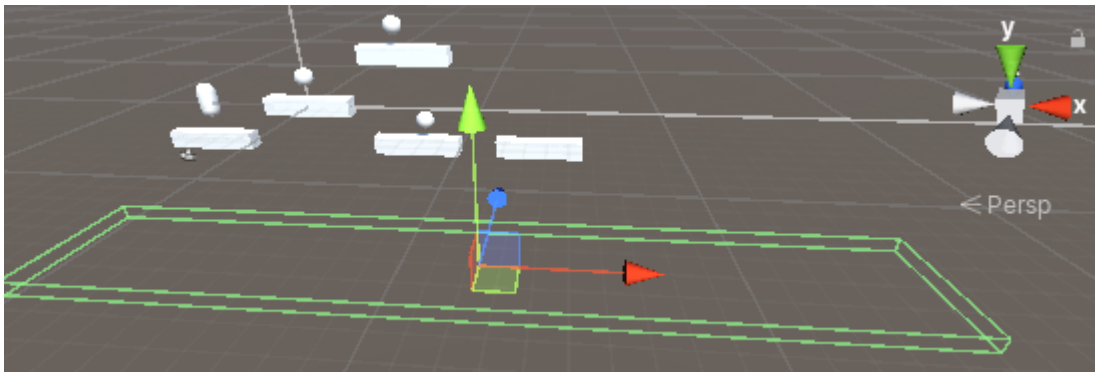
```
private void OnTriggerExit(Collider other)
    {

        if(other.gameObject==player)
        {
            other.transform.parent = null;
        }
    }
```

Dead-Zone

כאשר נרצה להגביל את השחקן שלנו לאזור מסוים נשתמש גם בטריגרים.

כעת נשתמש באובייקט קובייה כדי ליצור גבול מתחת לאזור המשחק. כאשר השחקן יגע באובייקט, הוא יחזור למקום התחלתי שאנו נקבע. (תיצרו אובייקט אזור טריגר חדש כמו שהוסבר למעלה)



סקריפט- כעת ניצור נהלים למקרה שבו השחקן נפגש באזור הטריגר שיצרנו:

```
[SerializeField]
private GameObject _respawnPoint;

private void OnTriggerEnter(Collider other)
    {

        if (other.tag == "Player")
        {
            Player player = other.GetComponent<Player>();
            CharacterController cc = player.GetComponent<CharacterController>();
            if (cc != null)
            {
                cc.enabled = false;
            }
            other.transform.position = _respawnPoint.transform.position;

            StartCoroutine(CCEnabledRoutine(cc));
        }
```

```
IEnumerator CCEnabledRoutine(CharacterController cc)
  {
        yield return new WaitForSeconds(0.5f);
        cc.enabled = true;
  }
```

הסבר:
ניצור משתנה מחלקה חדש של GameObject שהוא SeializeFiled כדי שנוכל לגשת אליו מה-inspector ואליו נכניס אובייקט מסויים שאליו נרצה שהשחקן יחזור בעת הטריגר.
לאחר מכן נוסיף פונקציית onTriggerEnter שבה נשנה את הtransform של השחקן לtransform של האובייקט.
אימפלמנטציה נוספת: אפשר פשוט לבחור מיקום על המפה במקום אובייקט ולתת לשחקן את המיקום הזה.

***כעת אם תנסו לגרום לשחקן ליפול (עם הקוד עד לעת עתה ולא הקוד המלא שמופיע למעלה), תגלו שהקוד לא עובד.**
הסיבה לכך היא מכיוון שהשחקן נע במהירות גבוהה והמערכת לא מצליחה להשתלט על הפיזיקה של השחקן שממשיכה גם לפריים הבא.
לכן ננטרל את הCharacter Controller של השחקן כפי שמופיע למעלה. כך השחקן יפסיק לזוז ונוכל להזיז את השחקן בצורה תקנית למקום ההתחלתי.

המשך הסבר:
כעת נרצה להחזיר את יכולת התנועה לשחקן. ההיגיון אומר שפשוט נכתוב בהמשך הפונקציה `.cc.enabled = true;`
זה לא יעבוד.
הסיבה היא שהפריים עדיין לא נגמר ומה שעשינו זה שפשוט כיבינו והפעלנו את הCharacter Controller והפיזיקה עדיין ממשיכה לשבש לנו את הפריים הבא.

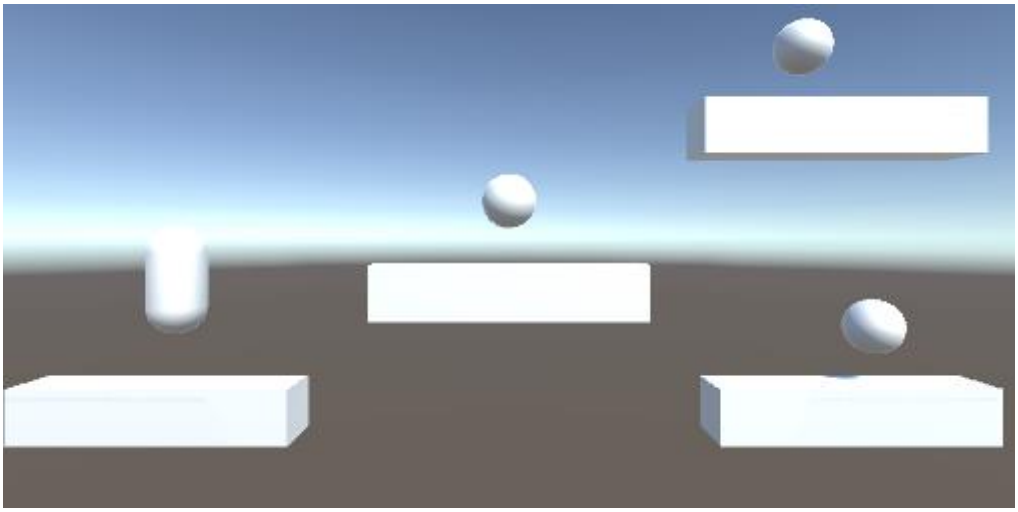
לכן ניצור Coroutine, שם נבקש להמתין זמן קצר כדי לעבור לפריים הבא, ולאחר מכן נפעיל חזרה את ה Character Controller.

-Collectable

בכל משחק טוב נוסיף חפצים\אובייקטים שניתן לאסוף אותם הנקראים Collectable. חפצים אלה יכולים להיות: מטבעות שאוספים לאורך השלבים, Power-Up , חיים, מגן ועוד שמטרתם להוסיף נפח ועניין למשחק.
במשחק הדו-מימד צברנו נקודות על ידי השמדת חלליות, לעומת זאת הCollectable שלנו היו Power-Up.
במשחק הנוכחי נוסיף מטבעות שיוסיפו לנקודות שהשחקן אסף.

נעשה זאת ע"י הוספת אובייקט חדש Sphere Object, נקרא לו coin או collectable ונוסיף אותו לתיקיית Prefabs.
כעת נוסיף אותו לתיקיית ה Level שלנו. ניצור שלושה מטבעות.

פזרו אותם על הפלטפורמות השונות.



לוגיקה של Collectable-
נוודא של- collectable שלנו יש collider מתאים ו-Rigidbody ללא גרביטציה.

סקריפט- נצטרך לעשות רק 2 דברים בסקריפט של הCollectable. אתם יכולים לנחש מה הם?

נצטרך ליצור פונקציית onTriggerEnter – כאשר הCollider השני שייך ל- player נשמיד את האובייקט.
בנוסף, נצטרך לעדכן את השחקן שהוא אסף מטבע.
בסקריפט של השחקן ניצור משתנה מחלקה חדש של הניקוד, וניצור פונקציה שכאשר קוראים לה היא מוסיפה x ניקוד לבחירתכם.

```
public class Coin : MonoBehaviour
{
    // Start is called before the first frame update
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            Player player = other.GetComponent<Player>();
            if(player!=null)
            {
                player.CoinCollected();
            }
            Destroy(gameObject);
        }
```

פונקציה בסקריפט של השחקן:

```
public void CoinCollected()
    {
        _Score += 100;
    }
```