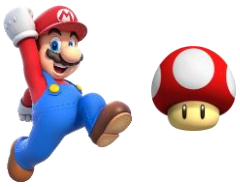


מבוא לתכנות משחקים ב3D

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-

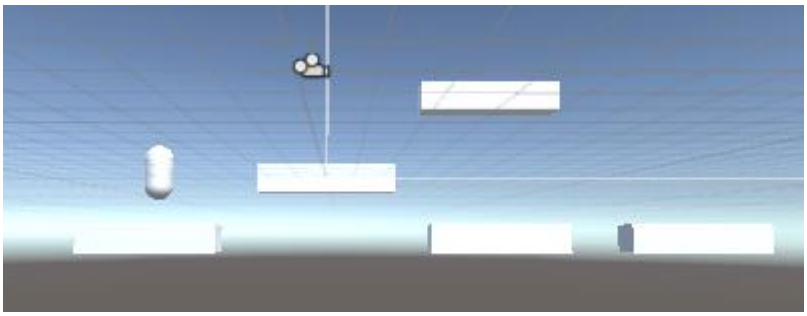
במשחקי תלת מימד, השימוש באלמנטים קבועים מראש הנטענים עם הסצנה, נפוץ יותר(אלמנטים כאלה יכולים להיות Power-ups, המשטח שעליו משחקים או המפה עצמה).
לכן חלוקה נכונה ומסודרת יותר של סצנה היא ע"י יצירת אובייקט ריק שמכיל את כל האלמנטים המרכזיים של השלב שאנו רוצים שיהיו קיימים כבר בשעת טעינת הסצנה. לצורך המשחק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

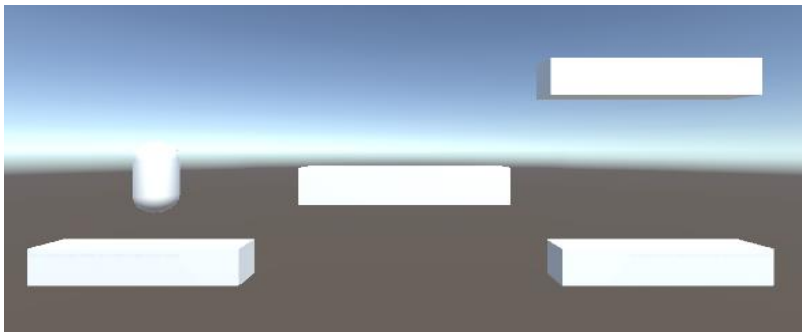
2.לפני שנתקדם, נשנה את זווית הראייה שלנו למישורי ה-x וה-y. ומעתה עד סוף המשחק נשמור שכל אובייקט שניצור כולל השחקן יהיה ממוקם ב0 על ציר ה-z.

3.כעת ניצור כמה שטחים או פלטפורמות עליהן יוכל השחקן ללכת.
נעשה זאת ע"י יצירת אובייקט Cube, ניצור ממנו prefab 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.

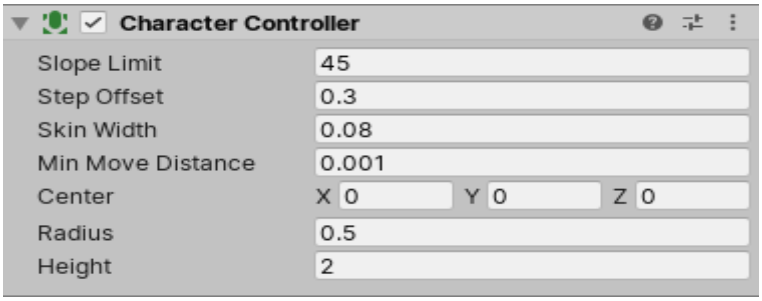
*לאחר שנלמד איך ליצור אובייקט collectable, נוסיף אותו לחלק מהאובייקטים הנכללים בשלב והוא אובייקט חיוני לתכנון סצנה.

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?