Player- Navigation, Shooting, Props, Destructible, Events.

נקדים ונאמר שנכון להיום, מומלץ להשתמש בגרסא 2020.1.0a ומעלה של Unity לחלק זה של הקורס.

Player

דבר ראשון ניצור אובייקט Capsule חדש, ונקרא לו Player. נוסיף לו רכיב מסוג Character controller דרך הinspector.
הקפסולה הזאת תייצג את השחקן שלנו.

על Character controller הרחבנו בעבר כבר, ואם תרצו ללמוד על נושא זה מחדש תוכלו להיכנס לתיקייה Unity-3dgames ושם לחפש את text-intro-to-3d-games.

Player Movement

בשלב זה אנו נכתוב פונקציונאליות לתזוזת השחקן.
ניצור ל-player שלנו script.

אז דבר ראשון שנרצה לעשות זה לגשת לאובייקט הCharacter Controller מהscript, לשם הנוחות נקרא לו מעכשיו CC.

זוכרים איך עושים את זה? ניצור משתנה מחלקה CC, וניגש אליו עם פונקציית הGetComponent.

```
private CharacterController _cc;
    // Start is called before the first frame update
    void Start()
    {
        _cc = GetComponent<CharacterController>();
    }
```

Move function

לCharacter Controller קיימת פונקציה move שמזיזה את השחקן לכיוון הווקטור ע"י הפרמטר שהיא מקבלת. במשחקי תלת מימד אנחנו נעים על משטח לאורך ציר הX וציר הZ, וכאשר אנו קופצים או עפים לאורך ציר הY.

לכן נגדיר בפונקציית הupdate וקטור שאחראי לקבל את כיוון התזוזה של השחקן. בציר הX נכניס לו נתון של ימין או שמאל שמסמלים כיוונים אופקי, וב-Z נכניס לו נתון של קדימה אחורה שמסמלים כיוונים אנכיים.

```
Vector3 direction = new Vector3(Input.GetAxis("Horizontal"),0,Input.GetAxis("Vertical"));
Vector3 velocity = direction * _speed;
velocity.y -= _gravity*Time.deltaTime;
_cc.Move(velocity*Time.deltaTime);
```

נשים לב שinput זה בעצם הכיוון אליו לחצנו, ימינה או שמאלה (1 או -1 או 0 אם לא לחצנו) וקדימה אחורה (1 או -1 או 0 אם לא לחצנו כלום).
Y נשאר 0 כל עוד אנו לא קופצים, נתייחס לכך בהמשך.

Time.deltaTime – נשים לב שגם כפלנו את הוקטור שלנו בdeltaTime שבעצם ממיר את התזוזה של כל יחידה פר פריים

לזמן של שניה אחת מפריים לפריים הבא. אם נרצה להגדיל את כמות היחידות בכל תזוזה פר פריים/שניה, נכפיל אותו בn כלשהו, ומעתה נזוז n יחידות פר פריים/שניה. זהו משתנה speed , ניצור אותו כמשתנה מחלקה.

בCC שלנו אנחנו גם צריכים להגדיר גרביטציה, בשונה מrigid body בו אנחנו מקבלים אופציה לגרביטציה.
לכן בכל פריים נחסר מהווקטור בY את הגרביטציה. כח הכבידה על כדור הארץ הוא 9.81 מטר לשניה בריבוע, לכן נחסר בזה.
זה אם כן משתנה מחלקה _gravity המופיע בקוד.

Mouse Movement

במשחקי תלת מימד, נהוג שהכיוון אליו מביט השחקן זה כיוון אליו הוא מתקדם.
כדי לעשות זאת, דבר ראשון נרצה לקבע את המצלמה שתראה את מה שהשחקן רואה. כדי לעשות זאת נשרשר את המצלמה אל השחקן ונמקם אותה בposition 0,0,0. לאחר מכן נמקם את המצלמה במיקום של ראש השחקן.
world space- הכיוון אליו מביטה המצלמה שלנו. Local space- המקום אליו מביט השחקן שלנו.
אנו רוצים שהworld space והlocal space שלנו יהיו אותו דבר. כך שלא משנה לאיפה נמקם את מבטו של השחקן, המצלמה תזוז איתו לכיוון הזה.
כרגע הקוד שלנו מדבר בשפה של Local space לכן אנחנו צריכים לשנות את השפה כך שתתאים גם לworld space.
איך נעשה זאת?

```
velocity = transform.TransformDirection(velocity);
```

כעת הכיוון אליו נביט יהיה באמת הכיוון של השחקן.

עכשיו נוכל להגדיר לאיפה נביט על פי תזוזת העכבר.
ניצור סקריפט חדש שיהיה אחראי על תזוזת העכבר לצדדים (ציר ה-X) וסקריפט אחד שאחראי על להביט למעלה ולמטה (ציר ה-Y).
כעת נכנס לסקריפט של ציר הX:
בפונקציית update נרצה לעדכן את כיוון העכבר על הציר:
נשמור את התזוזה של העכבר

```
float _mouseX = Input.GetAxis("Mouse X");
```

*כל ההגדרות של ה-Input נמצאות בInput Manager.

לאחר מכן כדי לעדכן:

```
float x = transform.localPosition.x;
float y = transform.localEulerAngles.y + _mouseX;
float z = transform.localEulerAngles.z;
Vector3 v= new Vector3(x, y, z);
transform.localEulerAngles = v;
```

אז למה קידמנו את y ולא את x? הרי אנחנו רוצים להביט סביב ציר הx?
ספציפית בפונקציה transform.localEulerAngles הערכים מתקבלים בסדר הבא z ואז x ואז y. לכן כשאנחנו מקדמים את הy אנחנו בעצם בתוך הפונקציה עצמה מקדמים את x.

נשים לב שלא נוכל לגשת ישירות לy ולגשת אליו

```
transform.localEulerAngles.y = _mouseX//!שגיאה
```

בכל שלב נצטרך להתייחס גם לx ולz.
אם נרצה לשנות את מהירות תזוזת העכבר, נכפיל את כמות היחידות ב- mouseX במהירות שנרצה.

אפשר לרשום את אותו קוד כך:

```
float _mouseX = Input.GetAxis("Mouse X");
Vector3 rotation = transform.localEulerAngles;
rotation.y += _mouseX * _speedRotation;
transform.localEulerAngles = rotation;
```

אז עכשיו נרצה לנוע על ציר ה-Y, אז כצפוי הקוד יהיה כמעט זהה לקוד הקודם:

```
float _mouseY = Input.GetAxis("Mouse Y");
Vector3 rotation = transform.localEulerAngles;
rotation.x += _mouseY * _speedRotation;
transform.localEulerAngles = rotation;
```

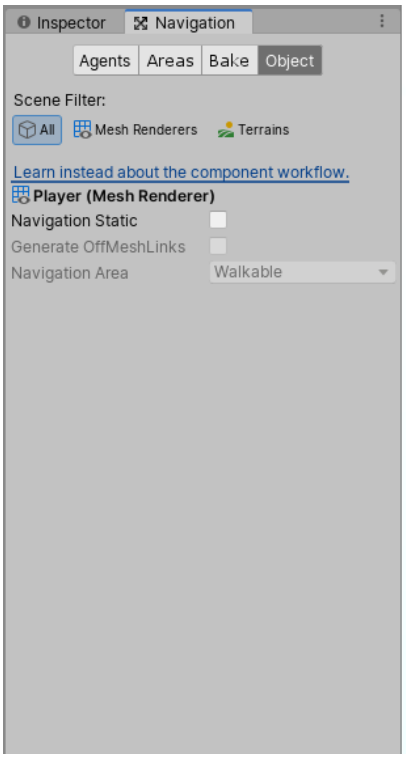
אבל אם ננסה כעת לזוז עם השחקן נראה שהתזוזה ממש בעייתית, כשנסתכל למעלה או למטה התזוזה תהיה מאוד מוזרה.
הרי השחקן שלנו מנסה לזוז לכיוון שאליו השחקן מסתכל. נרצה להתייחס רק למיקום שבו הוא מביט בציר ה-X.
כדי להתגבר על הבעיה, ניצור אובייקט ריק חדש (נקרא לו "UpDown" למשל), נשרשר אותו לPlayer ואת המצלמה לUpDown. את הסקריפט שיצרנו לציר ה-y ניתן לאובייקט UpDown.
כעת נראה שהתגברנו על הבעיה! המצלמה זזה, אבל אנחנו לא מתייחסים לכיוונים הללו מבחינת תזוזה.

להסבר נוסף על "מבט עוקב עכבר", ראו כאן: https://gamedev.stackexchange.com/questions/104693/how-to-use-input-getaxismouse-x-y-to-rotate-the-camera

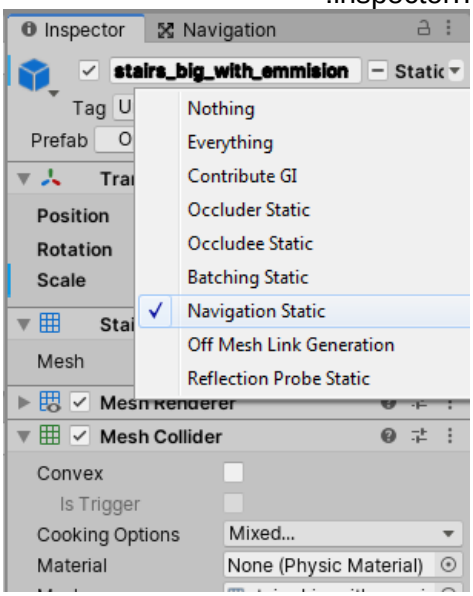
NavMesh

הגדרת מרחב התזוזה של השחקן. הUnity משתמש בCollidersים של האובייקטים כדי לחשב את המרחב הזה.
אנו נסמן את האובייקטים שיהוו את המרחב והUnity יחשב את השטח המשוכלל של כל האובייקטים הללו.
המרחב הזה יגדיר לשחקן ולדמויות אחרות במשחק עד איפה מותר להם לנוע. דבר זה למשל ימנע מהדמויות ליפול ממשטחים ללא צורך להגדיר גבולות בעצמנו.

ליד הinspector אמור להופיע לשונית Navigation, במידה ולא נלך ללשונית Window->AI->Navigation ונגרור את הלשונית של החלון שנפתח לבר של הinspector.
בלשונית הnavigation, הלשונית הראשונה שתפתח תהיה object, יש שם שלוש אפשרויות:
All – יציג לנו בחלון ההיררכיה את כל האובייקטים, ונסמן את אלה שהשחקן יכול לזוז עליהם. במקרה שלנו נסמן את הרצפה.
Mesh Renderers – יציג לנו את הcollidersים השונים של כל האובייקטים, משם נוכל לבחור באלה השייכים לרצפה.
Terrain – יציג לנו רק את אובייקטי המשטח שבסצנה.



לאחר שבחרנו, נסמן את כל האובייקטים הללו כnavigation static על ידי לחיצה על סמן הstatic בצד הימני העליון של הinspector.

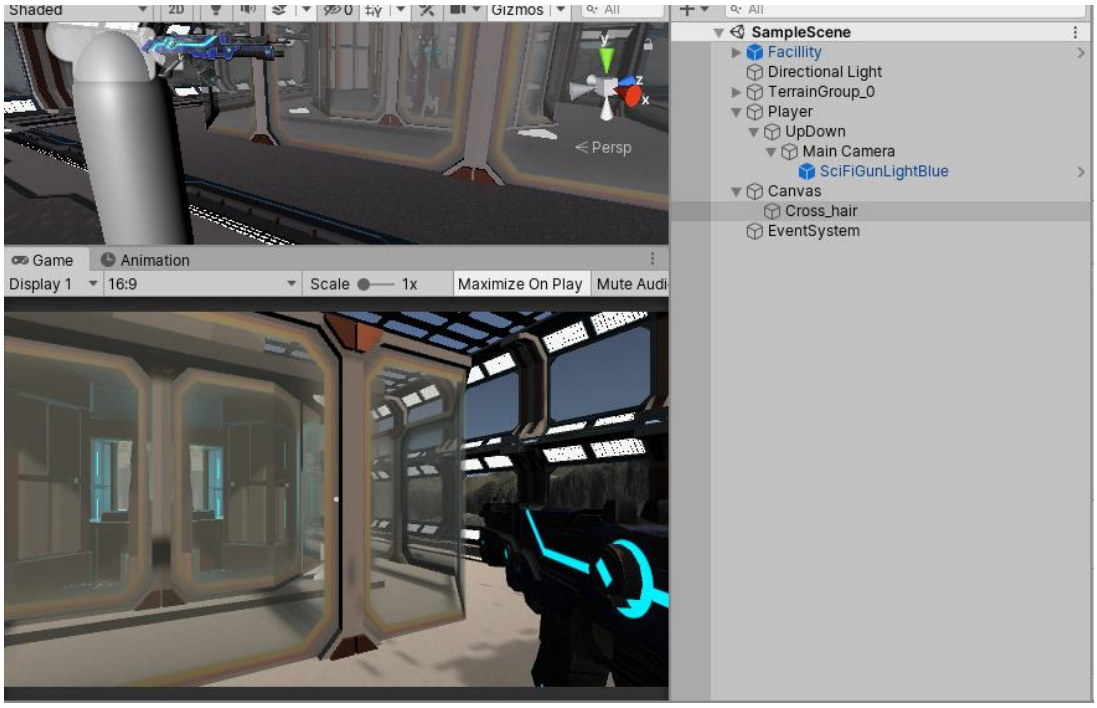


נעבור ללשונית Bake, נראה שם הגדרות מוזרות. כעקרון הunity צריך את השיפועים השונים של האובייקטים כדי לדעת לחשב את הNavMesh. משתני הAgent שמופיעים שם מתייחסים לרוחב וגובה השחקן וstep size מדבר על היכולת של האויבים והשחקנים לטפס על אובייקטים.

Shooting

קודם כל נוריד Asset מחנות unity בשם "Sci Fi Gun Light".
את הרובה הזה שהורדנו נמקם מימין לשחקן כך שיראה במצלמה כאילו מישהו מכוון נשק למרכז המסך.

ניצור canvas חדש ובו image שייצג את הכוונת שלנו.
בsource Image נבחר את האופציה knob. ונבחר את הצבע. אני בחרתי בצבע לבן.
כאשר נכוון על אויב, הכוונת תהפוך לאדומה. נעשה זאת בהמשך.



כדי שהעכבר לא יפריע לנו, נרצה להשאיר אותו חבוי. נעשה זאת בעזרת הקוד הבא. נשים לב שהוגדר שכאשר נלחץ ESC העכבר יחזור להופיע.

```
void Start() {
        _cc = GetComponent<CharacterController>();
        Cursor.visible = false;
        Cursor.lockState = CursorLockMode.Locked;
}

void Update() {
        if(Input.GetKeyDown(KeyCode.Escape)) {
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
        }
        MovementCalc();
}
```

RayCasting

קו או קרן אווירית היוצאת מאובייקט בנקודה אחת לאובייקט בנקודה אחרת.
המטרה היא שכאשר קרן זו פוגעת באובייקט אחר, היא יוצרת אינטראקציה ביניהם.
למשל, כאשר אנו יורים בנשק, אפשר להשתמש באפקטים חזותיים במקום באובייקט של כדור רובה.
אך איך נדע שפגענו באויב? באמצעות הraycast.

כעת נשתמש בדיוק בשביל המקרה המתואר בraycast.
ניצור קרן שתצא מאמצע המסך, וכאשר היא תפגע באובייקט, נדע שהצלחנו לירות עליו.
נבחר את הכיוון אליה הקרן נשלחת, וכן נבחר את אורך הקרן.

אז איך יוצר קרן כזו?

ניגש לסקריפט של האובייקט שממנו נרצה לשלוח את הקרן, לרוב זה יהיה הplayer שלנו.

```
  void Update() {
```

```
        if(Input.GetMouseButtonDown(0)) {
            //position ray casted from
            Ray rayOrigin = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
            RaycastHit hitInfo;
            if (Physics.Raycast(rayOrigin, out hitInfo)) {
                Debug.Log("ray hits: "+hitInfo.transform.name);
            }

        }
}
```

בהתחלה אנחנו מחכים ללחיצה על הכפתור השמאלי בעכבר (0 שמאל, 1 ימין).
אובייקט Ray: הוא הקרן שלנו, מאתחלים אותו על ידי מיקום התחלה. השתמשנו במצלמה לשם כך. במקרה שלנו אנחנו הגדרנו שהוא יצא בדיוק ממרכז המסך, ממרכז שדה הראייה של השחקן.
אובייקט RaycastHit: שומר נתונים בנוגע לאובייקט שבו הקרן פוגעת.
`Physics.Raycast()`: ניתן לתת לו כמה ארגומנטים. אובייקט Ray תמיד
1. אפשר להעביר לו גם אובייקט RaycastHit והפונקציה תאחסן באובייקט את הפרטים של האובייקט שנפגע.
2. אפשר להעביר לו מרחק וזהו. ואפשר גם להוסיף `int Layer` שזו הגדרה שנוכל להשתמש בה כאשר נרצה להתעלם מאובייקטים מסוימים בדרך/לפגוע רק באובייקטים ספציפיים.

Shooting Effect

נוריד את החבילה War Fx מהחנות של Unity, ושם נשרשר לרובה שלנו את האפקט "WFX_MF 4P RIFLE1" .
כעת נראה שהאפקט רץ בלופים.
נרצה דרך הסקריפט של Player להפעיל את האפקט רק כשאנו לוחצים על כפתור שמאלי בעכבר.
ניצור אובייקט מחלקה serialized filed שאליו נכניס את האפקט דרך הUnity:

```
[SerializeField] private GameObject _muzzleFlash;
```

לאחר מכן בupdate נגדיר:

```
if(Input.GetMouseButtonDown(0)) {
            _muzzleFlash.SetActive(true);
} else {
            _muzzleFlash.SetActive(false);
}
```

גם נרצה להשאיר סימן של הכדורים על המשטח שבו ירינו.

בנוסף נרצה גם אפקט פגיעה. נשתמש באפקט שהוא פריפאב מהasset שלנו בתיקיית Bullet Hole, ונרצה לגרום לו להופיע במקום אליו מגיע הRay שלנו.
קודם ניצור אובייקט מחלקה שיקבל את האפקט.
לאחר מכן ניצור את הפונקציונאליות:

```
if(Input.GetMouseButtonDown(0)) {
            _muzzleFlash.SetActive(true);
            //position ray casted from
            Ray rayOrigin = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
            RaycastHit hitInfo;
            if (Physics.Raycast(rayOrigin, out hitInfo)) {
               GameObject hitMarker =
(GameObject)Instantiate(_BulletHole, hitInfo.point, Quaternion.LookRotation(hitInfo.normal));
                Destroy(_BulletHole, 1f);
            }
}
```

זה הקוד שראינו קודם לכן, נראה שכעת הצהרנו על האובייקט כאשר הRay שלנו פוגע במשהו. יצרנו מופע שלו במקום שבו פגענו (hitInfo.point) וכן ווידאנו שהסיבוב שלו יהיה בהתאם לכיוון של הווקטור (לפי נורמל) של Ray (`Quaternion.LookRotation(hitInfo.normal)`).

השלב האחרון הוא להוסיף צליל של ירי. נלך לרובה שלנו ונוסיף לו רכיב Audio Source. נעביר לו קובץ audio כ audioClip כלשהו (קיים בתיקית הפרוייקט , ניתן לחפש בקלות בגוגל צלילים חינם).
נסמן ברכיב שהaudioClip יהיה במצב Loop ולא מצב play on awake.
כעת נכתוב לו את הקוד כך שבכל לחיצה שלנו על כפתור שמאלי בעכבר נשמע גם את האודיו קליפ.

ניצור אובייקט מחלקה שבו נכניס את הGameObject שממנו מגיע הסאונד, במקרה שלנו הרובה.
לאחר מכן בפונקציית Update:

```
  if(Input.GetMouseButtonDown(0)) {
            if (_rifleSound.isPlaying == false) {
                _rifleSound.Play();
            } else {
                   _rifleSound.Stop();

            }

}
```

דרך אחרת. ליצור Script לרובה, בו כאשר אנו לוחצים על כפתור שמאל בעכבר, אנו מפעילים ידנית את הaudioClip מהקוד.
למה צריך לדעת את זה? לפעמים השיטה הראשונה לא עובדת חלק.
ניצור אובייקט מחלקה AudioClip, וניצור את הסאונד.

```
AudioSource.PlayClipAtPoint(_audioRifle, transform.position, 1f);
```

Props

במשחקים יש גם אובייקטים שהשחקן יכול ליצור איתם אינטראקציה. האובייקטים יכולים להיות דמויות אחרות, תיבות, דלתות.
דבר זה מתבצע ע"י כניסה לאזור הcollider של האובייקט. כאשר הוא במצב triggered מתאפשרת לנו האופציה לבצע אינטראקציה.

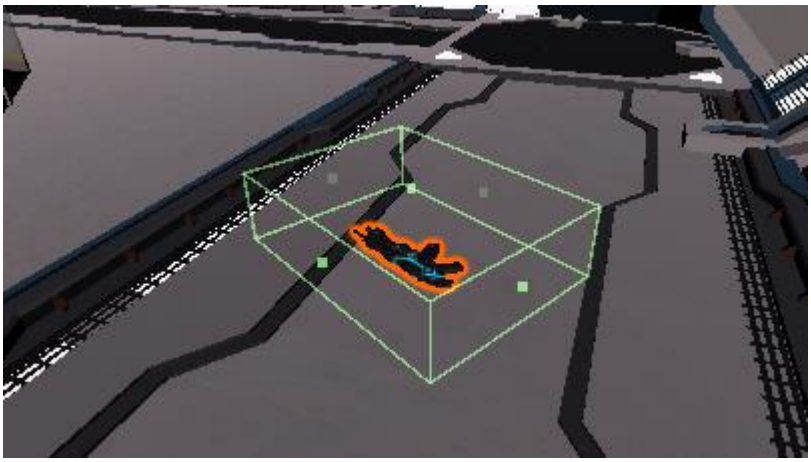
לעיתים גם אלה חפצים שניתן לאסוף. זה יכול להיות כסף, תחמושת, ערכה של עזרה ראשונה ועוד.

נדגים איך הדבר מתבצע.

במשחקים נהוג שכאשר אנו רואים נשק שזרוק על הרצפה (למשל בעקבות אויב שהפיל אותו לאחר שנפגע מירי), בדרך כלל תינתן לנו האופציה להשתמש בו במקום הנשק שיש לנו, או שנוכל לאסוף אותו ולקבל עוד תחמושת במקרה ויש לנו אותו נשק.

בשלב זה נניח שראינו על הרצפה נשק מאותו סוג כמו שלנו.

דבר ראשון נמקם אותו בscene view, ניתן לו collider כלשהו במצב isTrigger.



ניצור script לחפץ, כך שכאשר נהיה ממוקמים בתוך הcollider, נוכל על ידי לחיצת כפתור לאסוף את הנשק. נקבל עוד ammo, והנשק יושמד.

```
public class WeaponProp : MonoBehaviour
{
    private void OnTriggerStay(Collider other)
    {
        if (other.tag == "player")
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                Player player = other.GetComponent<Player>();
                if(player!=null)
                {
                    player._addAmmo();
                    Destroy(this.gameObject);
                }
            }
        }
    }
}
```

הפונקצייה בplayer:

```
  public void _addAmmo()
    {
        _ammo = _startAmmo;
    }
```

-Destructible

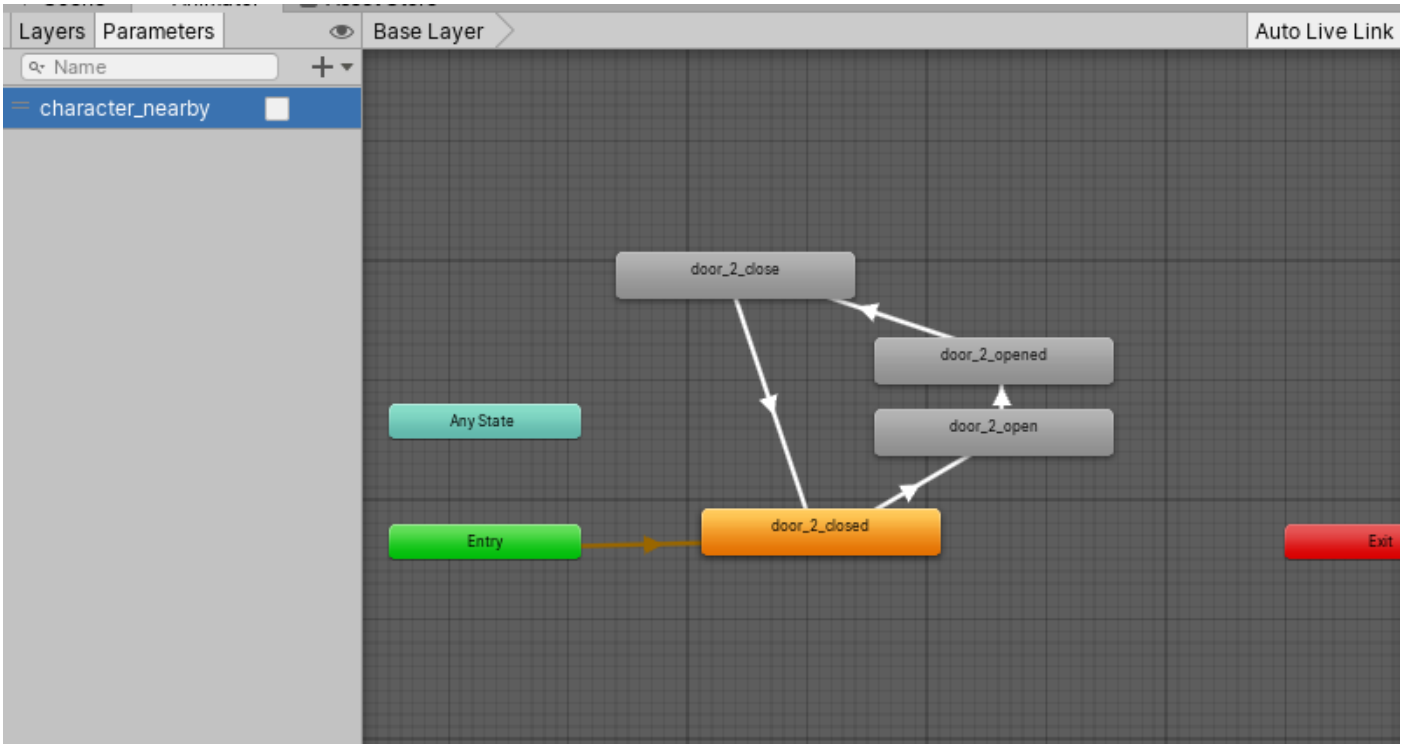
כאשר נרצה ליצור אובייקטים שניתן להרוס אותם. למשל ארגז שנשבר כשיורים עליו, או בקבוק שמתנפץ, השיטה הפשוטה ביותר היא ליצור אובייקט ואובייקט זהה לו, רק שבאובייקט אחד יהיה "מפורק" ככה שהוא יהיה מורכב מהרבה חתיכות קטנות. לכל חתיכה ניצור פיזיקה שתזרוק אותה לכיוון אחר.
כאשר הRay שלנו יזהה שהוא פגע באובייקט שאותו נרצה להרוס, נחליף אותו באובייקט הזהה לו אשר מפורק לחלקים ועליו נפעיל את הפיזיקה.

-Events

בפרוייקט שלנו גם קיימת דלת, שעד עכשיו לא השתמשנו בה. יש לה גם אנימציה! של דלת נפתחת ונסגרת.
נרצה שכאשר השחקן מתקרב לדלת, הדלת תפתח, וכאשר הוא מתרחק היא תיסגר חזרה, כמו דלת של סופר.

זה נקרא מאורע. כאשר אנחנו מעוררים אובייקט אחר על ידי יצירת אינטראקציה עם הcollider שלו.

אז ניצור script לדלת שלנו, ניצור אובייקט מחלקה Animator שבו האנימציה של הדלת.
נכנס לanimator-



נראה שאנחנו מתחילים בEntry, הכוונה יצרנו אינטראקציה עם האובייקט, ומסיימים בExit שאומר שאנחנו כבר לא באינטראקציה.
לאנימציה יש מצב דיפולטיבי שזה door_2_closed הדלת סגורה...
לאחר מכן מתבצעות מספר פעולות, הדלת נפתחת, הדלת פתוחה והדלת נסגרת.
בנוסף נראה כי קיים bool בשם character_nearby. הוא ישמש אותנו כדי לדעת אם השחקן באזור הדלת.
לכן נצטרך בקוד להגדיר את הbool כtrue כדי שהאנימציה תפעל.

```
public class Door : MonoBehaviour
{
    private Animator _animator;
    // Start is called before the first frame update
    void Start()
    {
        _animator = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            _animator.SetBool("character_nearby", true);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        _animator.SetBool("character_nearby", false);
    }
```

}