



Artificial intelligence in Games

מבוא ל-AI

כאשר אנו רוצים ליצור לדמות שהיא לא השחקן (NPC) התנהגות ואינטראקציה עם הסביבה, אנו נשתמש ב-AI בכדי לנסות ולדמות לשחקן התנהגות אנושית. בשימוש ב-AI, חווית המשתמש היא הרבה יותר איכותית. ה-AI מעניק תחושה למשתמש שהוא משחק מול המחשב ומנסה לנצח אותו. זה גורם מרכזי לרצון של המשתמשים לחזור למשחק, ולנסות להתגבר על הקשיים שהמחשב מציב למשתמש. לכן כאשר AI במשחק מסוים נבנה בצורה לא טובה, זה יכול להיות מפתח מרכזי לכשלון של המשחק להוות חוויה שהמשתמש מעוניין בה. AI בעצם נבנה על הפונקציונאליות שמפתח המשחקים בונה. אם נבנה AI שהוא חכם מדי, אולי לא יהיה אפשרי לנצח את המחשב. לרוב זה דבר לא טוב. אם נבנה AI טיפס מדי, המשתמש ישתעמם מהר מן החוויה. גם לא טוב. לכן תחום ה-AI הוא תחום מרכזי במשחקים המבוססים על חווית משתמש יחיד או רבים מול מחשב, ותכנון ה-AI צריך להיות מדויק. אחרת, חווית המשתמש תהרס.

מנוע ה-AI של UNITY

ספרייה שמרכזת את כל שנצרך ליצירת AI במשחק שניצור בסביבת UNITY.

NevMeshAgent - כל האובייקטים והפונקציות הנגזרות מספריית המנוע של UNITY AI, קשורות ל-NevMeshAgent, שזוהי המחלקה האחראית על ההגדרות הקשורות לקשר שבין NPC לבין הפעילות שלו בסביבת NevMesh.

הפונקציות שהספרייה מספקת לנו, הן פונקציות חישוביות שקשורות למסלול והסביבה של ה-AI. הפונקציה המרכזית שבה אנו מחשבים מסלול היא הפונקציה Set Destination:

-NavMeshAgent.SetDestination(vector3)

כדי להודיע לשחקן לאן להגיע, אנחנו נשתמש בפונקציית setDestination. המנוע של יוניטי יחשב את הכיוון ואת המרחק שהשחקן ינוע. הוא יעשה זאת בזמן שהוא מתחשב גם באובייקטים השונים הנמצאים בדרך. הוא בעצם מחשב מסלול בהתחשב ב-NevMesh שבסצנה, שמראה איפה נמצאים קולידרים שיפריעו ל-NPC להתקדם במסלול. לכן יש לעקוף אובייקטים אלה, הנקראים בדרך כלל "מכשולים".

אז אם נרצה לגרום לאובייקט מסוים לעקוב אחרי השחקן, מה נעשה? נוכל למשל לייבא את האובייקט של השחקן, ולשלוח לsetDestination את transform.position של השחקן.

למשל כך:

```
// לשנח לא using UnityEngine.AI

Private NavMeshAgent _navMeshAgent;
Private Player _player;
Vector3 _dest;

private void Start()
{
    _navMeshAgent = this.GetComponent<NavMeshAgent>();

    //null לבדוק לא לשנח
    _player = FindObjectOfType<Player>();

    //null לבדוק לא לשנח
}

private void Update()
{
    _navMeshAgent.SetDestination(_player.transform.position);
}
```

התוצאה היא שכיוון האובייקט תמיד יהיה הכיוון שבו השחקן נמצא.

נניח ונרצה ליצור דמויות אחרות שנוכל לשלוט עליהם, ולהגיד להם לאיפה ללכת על ידי לחיצה על הכפתור הימני בעכבר.

איך נעשה זאת?

דבר ראשון נרצה לדעת איזה מיקום סימננו על המפה. פה נכנס כלי חשוב מאוד שכבר דיברנו עליו, RayCast.

נראה את הקוד הבא:

```
If(Input.GetMouseButtonDown(1))
{
    Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hitInfo;
    physics.Raycast(ray, out hitInfo)
    _navMeshAgent.SetDestination(hitInfo.point)
}
```

אם נשתמש בקוד הזה על כל NPC הוא יציית לפקודות שלנו.

זאת התחלה ליצירת Companion (שותף) כך שנוכל לשלוט בטקטיקה שלו בזמן משחק.

בהמשך נלמד קצת על ירי של NPC, וככה נוכל גם לגרום לCompanion שלנו לירות על NPC אחרים ובכך לעזור לנו לנצח במשחק!

-Enemy

במשחקי FPS (First Person Shooter) רוב ה-NPC משמים כאוייבים לשחקן. הם בדרך כלל ינסו לירות על השחקן ולגרום לו להפסיד במשחק.

מה אנחנו צריכים בעצם שהאויב יעשה?

הדברים הבסיסיים שאויב עושה זה: רץ, מתחבא, יורה/תוקף.

כמובן שזה תלוי במשחק ובצורך.

ישנם דברים כמו התנהגות בקבוצה, סולם העדפות לאויב, (האם לירות על השחקן? או להשמיד מטרה אחרת?), להיות תוקפני? או הפוך, דווקא להיות פחדן...

בסופו של דבר כל ההתנהגויות שניצור ל-NPC שלנו הן סינטטיות ונגרמות על ידי הלוגיקה שבנינו בקוד שלנו. נראה מספר דוגמאות, שיעידו כי כל התנהגות נופל על לוגיקה ופרמטרים:

כאשר אנו רוצים להגדיר לאויב לאיפה ללכת, אנו עושים זאת על ידי `setDestination`. אבל מה המקום אליו נגיד לו ללכת? אל השחקן? אולי, זה תלוי בכם.

אבל מה אם נרצה שהאויב שלנו יוכל לבחור בעצמו לאן ללכת?

במשחק שנבנה כאן, נשתמש ב-`terrain` שיצרנו במדריכים הקודמים, ונמקם בו חיילים.

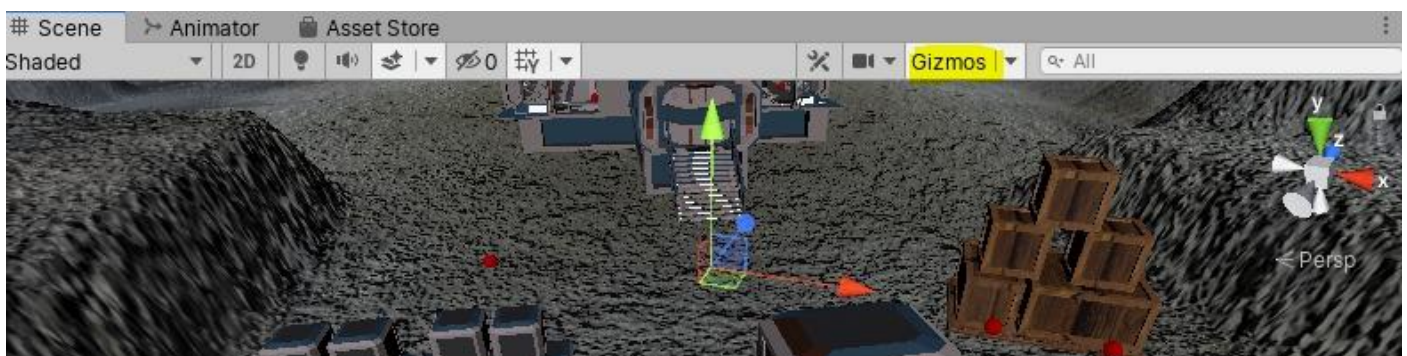
החיילים האלה ירוצו על גבי המפה, ויבחרו לאיזה מחבוא/או נקודה נרוץ הלאה.

לכל נקודה כזו נוכל לקרוא בשם `Target` והוא יהיה בעצם אובייקט ריק.

ניצור אובייקט ריק בשם `target` וניצור לו `script`. נשתמש ב-Gizmos כדי שנוכל לראות את `target` על `scene view`.

מחלקת Gizmos:

נועדה לעזור למשתמש על ידי מתן מראה ויזואלי לרכיבים ואובייקטים שונים בסצנה שלנו. כדי להפעיל את Gizmos נצטרך ללחוץ על כפתור Gizmos בחלון `scene view`.



בוא נראה את הקוד הבא לאובייקטי `Target` שלנו:

```
public class Target : MonoBehaviour
```

```
{
```

```
    private void OnDrawGizmos()
```

```
    {
```

```
        Gizmos.color = Color.red;
```

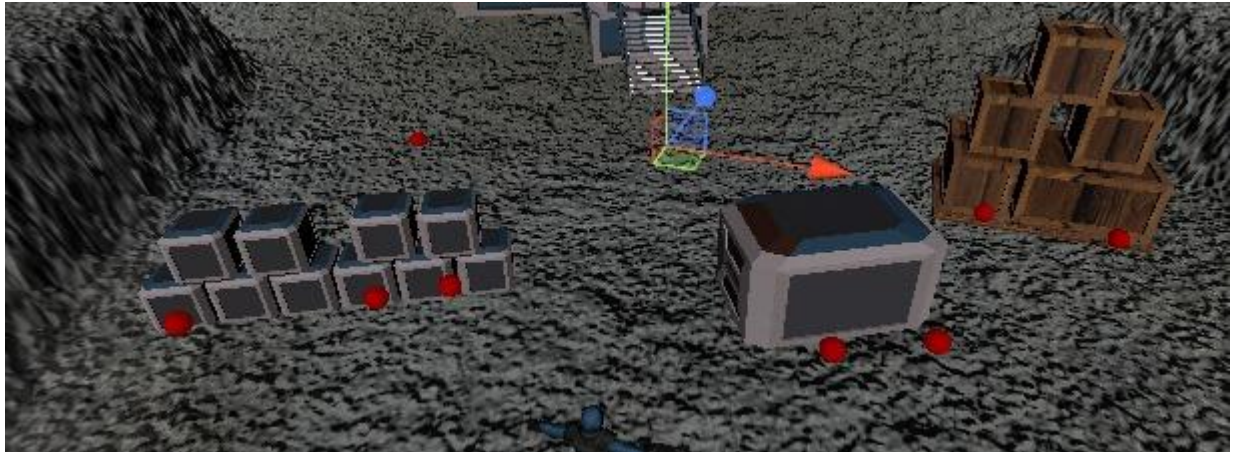
```
        Gizmos.DrawSphere(transform.position, 0.3f);
```

```
    }
```

כל מה שעשינו בקוד הזה, זה לצבוע כל אובייקט Target בצבע אדום.

נפזר כמה אובייקטים כאלה בנקודות מפתח שנרצה על גבי אזור המשחק שנו, ונשרשר את כולם לאובייקט ריק Targets, אך ורק בשביל הסדר בלשונית ההיררכיה.

זה צריך להראות כך:



כעת כשאנחנו יודעים לאן האויב שלנו אמור ללכת, בוא ניצור אותו!

הורידו את החבילה בשם ToonSoldier_demo מהחנות של unity. בתוך התיקיה של Asset הנ"ל, נכנס לתיקיית models ושם נגרור את האובייקט ToonSoldier_demo לאובייקט ניתן את השם Enemy. אם נסתכל עליו עכשיו בscene view נראה משהו ממש מצחיק. לכן נכנס עכשיו לתת נושא חדש ונסביר את הדברים.

- Character Animation

בשלב זה נלמד איך להתמודד עם אנימציה בUnity, ולא נלמד איך ליצור אנימציה בעצמנו. ישנם תוכנות ליצירת אנימציות כגון Blender, אך על זה אפשר ליצור קורס שלם בפני עצמו. במקום זה נראה מה עושים עם אנימציה מוכנה בunity.

דבר ראשון, נראה שהשחקן שלנו עומד בצורה מוזרה.

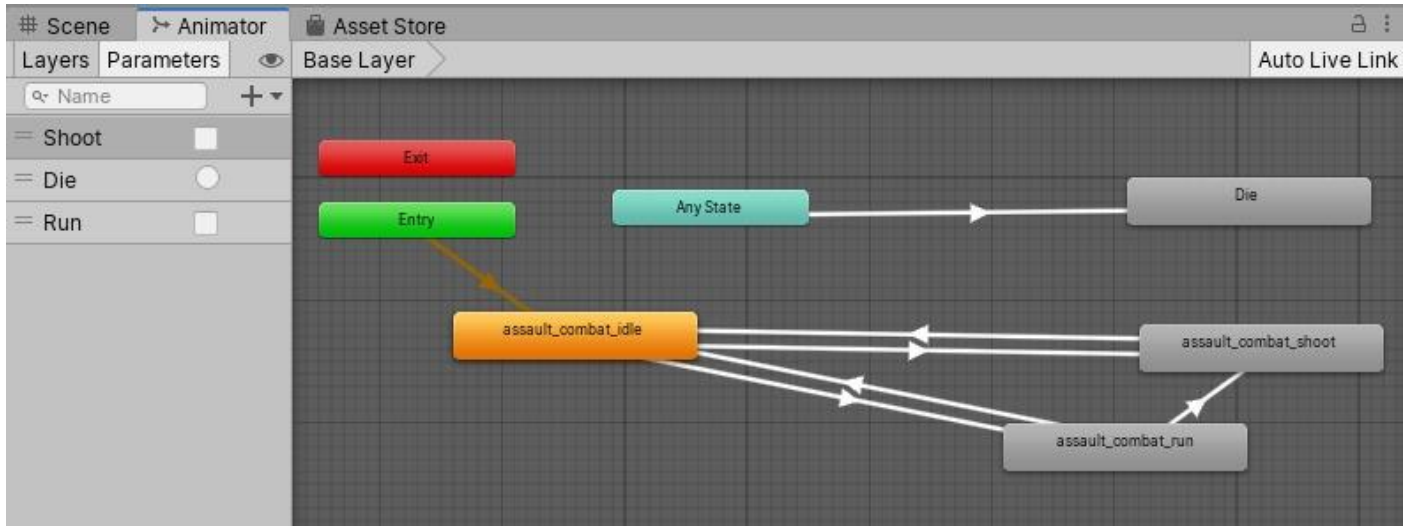


כן... אז יש שם לעמידה הזו. זה נקרא T-Pose.

T-Pose זוהי צורת הברירת מחדל שבה עומד אובייקט של שחקן לפני שנפעיל את Animator/לפני שה-NPC מתעורר.

זו היא נקודת התחלה שממנה מתחילים לבנות את התנועות השונות של הדמות. כאשר Animator מתחיל לפעול, נראה שהדמות תזוז לאנימציה ברירת המחדל של Animator.

כעת נסתכל על האנימציות השונות במכונת המצבים של Animator. נשים לב כי עם השחקן שהורדנו, לא קיימת מכונת מצבים כפי שמופיע בתמונה מטה.



כדי להוסיף מצב חדש, פשוט נגרור מתיקיית האנימציות של הasset שלנו את האנימציה לחלון מכונת המצבים. כעת נבחר באנימציה של idle כמצב default ע"י לחיצה על המצב עם כפתור ימני בעכבר ובחירה ב-set as layer default State.

לשם רענון, נזכר קצת מה הם החיצים השונים שנמצאים במכונת המצבים, ולמה יש מצבים בצבעים שונים.

המצבים האפורים – מצבים רגילים.

המצב הצהוב – Default State.

המצב הירוק והאדום – מצב כניסה ויציאה שקיים בכל animator, מצב הכניסה מוביל למצב הדיפולטיבי, ובדרך כלל מצב היציאה מגיע כאשר האובייקט כבר לא בסצנה נמצא.

(למשל דמות שרואים הולכת לאורך מסלול מסוים, ברגע שהדמות תצא מטווח הראייה האפשרי של השחקן, היא תכנס למצב Exit ומשם ל-T-Pose).

המצב הכחול – AnyState, מצב שמגיב כאשר Trigger מופעל. כאשר הטריגר מופעל נעבור מכל מצב שאנו נמצאים בו למצב זה.

החיצים- כל חץ בעצם נקרא Transition או טרנזאקציה בלועזית. זה בעצם אומר, שאם קורה משהו מסוים, נעבור ממצב אחד למצב אחר.

למשל החץ שעובר מהמצב idle (default staten) למצב Run.

כאשר הbool בשם run יהיה במצב true, נעבור בחץ (טרנזאקציה) הזה למצב run ממצב idle. וכעת תתחיל אנימציה של ריצה.

כאשר נסמן טרנזאקציה כזו עם העכבר, inspector יראה כך:



נראה כי נוכל ליצור condition. condition זה ההגדרה ל – איך נעבור בטרנזאקציה זו. במקרה שלנו, הגדרנו כי כאשר run יהיה שווה true, נעבור בטרנזאקציה.

איך בסקריפט שלנו ניגש למשתנים האלה?
 ניצור אובייקט מסוג Animator ונאתחל אותו על ידי GetComponent.
 לאחר מכן ניגש אל האובייקט ונפעיל את הפונקציה.SetBool.

```
_animator.SetBool("Run", true);
```

דברים נוספים:

1. בלשונית parameter נוכל להוסיף פרמטרים שונים, כגון bool triger על ידי לחיצה על כפתור ה+.
2. נוכל ליצור טרנזאקציה חדשה על ידי לחיצה על מצב עם כפתור ימני בעכבר, ושם לבחור באופציית Make Transaction.

כעת נסו ליצור את מכונת המצבים המופיעה מעלה, ותקבלו את האנימציה הרצויה לאויב שלנו.

כעת נחזור לנושא הפונקציות AI-ב.

כעת ניצור דרך לגרום לאויב לבחור target כלשהו על הסביבה שלנו.

ניצור אובייקט target ומערך של target לתוכו נכניס את כל אובייקטי הtarget שבפרוייקט. איך נעשה זאת?

בדרך הבאה:

```
using UnityEngine.AI;
```

```
public class Enemy : MonoBehaviour
```

```
{
```

```
    private NavMeshAgent _navMeshAgent;
```

```
    private Target[] _Targets;
```

```
    private Target _target;
```

```
    private void Start()
```

```
    {
```

```
        _Targets = FindObjectsOfType<Target>();
```

```
    }
```

```
    private void Update()
```

```
    {
```

```
    }
```

```
}
```

כעת נרצה לגרום לשחקן לבחור טארגט כלשהו.

איך נעשה? נוסיף גם בstart וגם בupdated:

```
_target = _Targets[Random.Range(0, _Targets.Length - 1)];
```

```
_navMeshAgent.SetDestination(_target.transform.position);
```

אז מה עשינו בעצם? גרמנו לאויב שלנו לבחור מספר רנדומאלי שלם, בין 0 לגודל המערך (פחות אחד), וזה יהיה האיבר במערך שיצרנו. כל איבר במערך הוא target, ואמרנו לNPC שלנו, לך לכיוון הtarget.

אם תפעילו כעת את המשחק, השחקן שלנו יתחיל לרחף לכיוון שהוא בחר.

מבוא לפיתוח משחקי מחשב ד"ר סגל הלוי דוד אראל, מיכאל למברגר, מעוז גרוסמן
אם נרצה נוכל להפעיל את האנימציה שלנו כדי שהשחקן יתחיל "לרוץ".
לכן נכתוב

```
_animator.SetBool("Run", true);
```

וכאשר NPC יגיע אל המקום אליו הוא רצה להגיע, נפסיק את האנימציה כך:
בתוך Update (או שנוכל ליצור פונקציה חיצונית)-

```
if (!_navMeshAgent.hasPath)
```

```
{
```

```
    _animator.SetBool("Run", false);
```

```
{
```

כעת זה נראה כאילו NPC מתרוצץ לו ממקום למקום, וקשה לנו לראות כאשר הוא מגיע ליעד שהוא הציב לעצמו.

לכן נרצה ליצור איזשהו מנגנון שיגרום לו להמתין במקום לפני שיבצע את הבחירה הבאה שלו.
נוכל ליצור coroutine חדש, ופשוט לגרום לפונקציה לחכות כפי שלמדנו.

נוכל ליצור גם אובייקט float שתסמן את הזמן עד המעבר לפונקציה הבאה. וכל פעם לחסר אותה ב-Time.deltaTime.
אם נגדיר 7f, לא נכנס לפונקציה 7 שניות וכו'.

משתנה מחלקה- לא לשכוח להצהיר://

```
private float NextState;
```

```
NextState -= Time.deltaTime;
```

```
if (!_navMeshAgent.hasPath)
```

```
{
```

```
    _animator.SetBool("Run", false);
```

```
if (NextState <= 0)
```

```
{
```

```
    _target = _Targets[Random.Range(0, _Targets.Length - 1)];
```

```
    _navMeshAgent.SetDestination(_target.transform.position);
```

```
    NextState = Random.Range(7f, 15f);
```

```
    _animator.SetBool("Run", true);
```

```
}
```

```
}
```

אז נבדוק קודם, האם NPC הגיע למקום היעד? או במילים אחרות, האם נשאר לו עוד מסלול לעבור עד ההגעה ליעד?
אם לא, עצור את אנימצית הריצה.
ולאחר שהNextState יגמר, נבחר יעד חדש לNPC שלנו.

במשחקים מסויימים, אולי נרצה לתת לNPC שלנו להסתובב בחופשיות על המפה שלנו בלי להיות כפוף לנקודות מסויימות על המפה. אפשר לדוגמא לעשות זאת בדרך הזאת:

```
private Vector3 GetRandomDir()
{
    float x = UnityEngine.Random.Range(-1f, 1f);
    float z = UnityEngine.Random.Range(-1f, 1f);
    Vector3 dir=new Vector3(x,0,z).normalized;

    return startPosition + dir * Random.Range(10f, 70f);
}
```

פשוט נחזיר את האובייקט Vector3 ונפעלי את פונקציית .setDestination.

כעת אנו רואים שיש לנו NPC שבוכחות עצמו מסתובב לנו על המפה ומחליט לאן ללכת.

הבעיה שכעת נרצה גם שהאויב יעשה איתנו אינטראקציה כלשהי. לעת עתה הוא ממש אנטיפת! דבר ראשון נרצה שהוא בכלל יביט לכיוון שלנו כשנתקרב אליו.

איך נעשה זאת?

דבר ראשון ניצור שלושה משתני מחלקה: הראשון הוא gameObject של Player שיהיה serializeField, ונכניס לתוכו את player שלנו. השני הוא Vector3 שייצג את המיקום של השחקן שלנו בכל שלב על המפה. השלישי הוא float שייצג את המרחק שהNPC "רואה".

```
private float lookRadius = 10f;//משתנה מחלקה

private Vector3 _playerPos;//משתנה מחלקה

[SerializeField]

private GameObject _player;//משתנה מחלקה

float distance = Vector3.Distance(_playerPos, transform.position);

private void Update()
{
    _playerPos = _player.transform.position;

    float distance = Vector3.Distance(_playerPos, transform.position);

    if (distance <= lookRadius)
```

```

{
    FacePlayer();
}
}

private void FacePlayer()
{
    Vector3 direction = (_playerPos - transform.position).normalized;
    Quaternion lookRotation = Quaternion.LookRotation(new Vector3(direction.x, 0, direction.z));
    // transform.rotation = lookRotation;
    transform.rotation = Quaternion.Slerp(transform.rotation, lookRotation, Time.deltaTime * 5);
}

```

הסבר:

בכל שלב אנו בודקים את המיקום של השחקן על גבי המפה.
אם הוא נמצא בתוך טווח הראייה של NPC (בדקנו זאת ע"י שימוש בפונקציה של מחלקת Vector3), נעבור לפונקציית facePlayer.

בפונקציית facePlayer אנו מוצאים את הכיוון הכללי של השחקן על המפה ע"י שימוש בפונקציית הנרמול.

לאחר מכן אנחנו משתמשים על אובייקט Quaternion על מנת להגדיר תנועה על גבי הצירים (רוטציה).
כדי לעשות זאת נשתמש בפונקציית LookRotation וניתן לה את הכיוון של הצירים השונים עם אובייקט Vector3.

לבסוף נצטרך להגדיר את transform.rotation. נוכל פשוט לתת לו את אובייקט lookrotation, אבל אם נרצה ליצור תנועה חלקה יותר, נוכל להגדיר על ידי פונקציית slerp את זמן גידול האפסילון של כיוון הרוטציה.

להרחבה בנושא:

<https://docs.unity3d.com/ScriptReference/Quaternion.Slerp.html>

גם כעת נוכל לגרום ל NPC לירות עלינו כשאנחנו נהיה בטווח שלו.
ממש כמו ב player ב נוסיף לו BulletHole muzzleFlash.

גם נוכל להוסיף לו AudioSource ולהפעיל אותו עם AudioClip שידמה ירי.

כך תראה הפונקציה:

```

private void ShootPlayer()
{
    _animator.SetBool("Shoot", true);
    _audioSource.Play();
    _muzzleFlash.SetActive(true);
    //position ray casted from
    Ray rayOrigin = new Ray(transform.position + Vector3.up * 1.5f, transform.forward);
    RaycastHit hitInfo;
    if (Physics.Raycast(rayOrigin, out hitInfo))
    {
        GameObject hitMarker = Instantiate(_bulletHole, hitInfo.point,
        Quaternion.LookRotation(hitInfo.normal)) as GameObject;

        Destroy(hitMarker, 1f);
    }
}

```

אז התלנו rayCast שיוצא בערך מהמיקום של הרובה של האויב.

מה אם נרצה למשל לדעת שפגענו בשחקן? אולי כדי להוריד לו מהחיים? איך נעשה את זה?

פשוט נכתוב את השורה הבאה:

```

If(hitInfo.collider.GameObject.tag == "Player")
//תעשה משהו

```

גם באותה דרך נוכל לגרום לcompanion שלנו לבדוק האם הוא פגע בשחקן שלנו.

נניח ולא נרצה לתת לcompanion לפגוע בשחקן.

קודם נבדוק האם RayCast שלנו פגע בשחקן. נניח והוא פגע, נוכל לוותר על כל האנימציה של הירי, וזה יראה כאילו companion שלנו מעולם לא ירה על השחקן.

במטלה תצטרכו למצוא דרכים יצירתיות לפתח את AI שלנו. כעת הוא די טיפש, אבל אתם תצטרכו לגרום לו להחכים מעט.