גרסת UNITY: 2018.3.0f2

עדכון אחרון: ‏יום שלישי, 26 בפברואר 2019

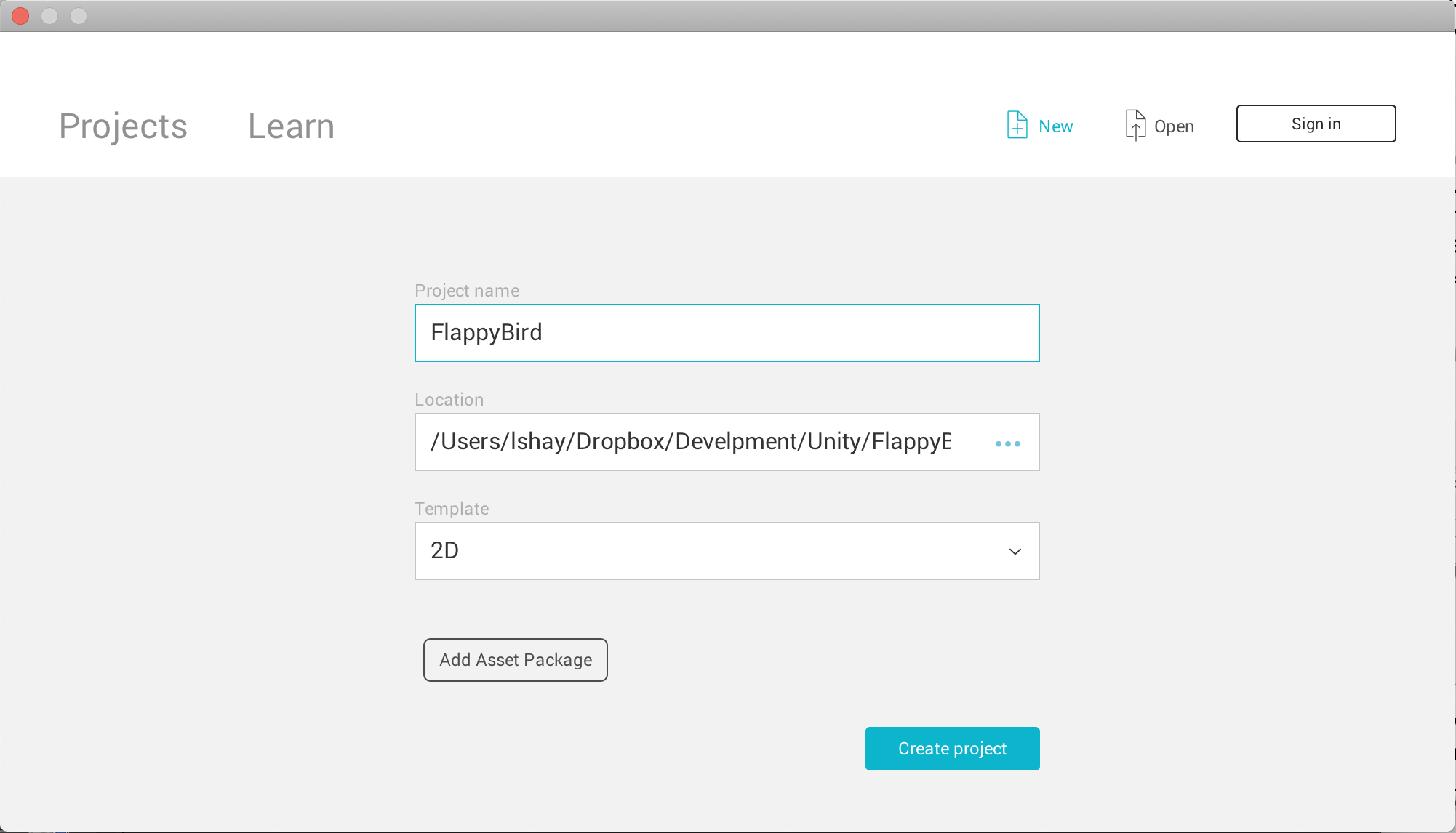
תיקיות עזר בדיסק:

1. תיקיית Resources מכילה שתי חבילות unity שנייבא במהלך העבודה
2. התיקיות FlappyBirdAI0x מכילות שלבים במהלך ביצוע העבודה, כך שהערך של x מייצג את סיום השלב. *ניתן לדלג על כל חלק בהדרכה ע״י פתיחת הפרויקט FlappyBirdAI0x מהדיסק (אם נגמר הזמן או במידה שהפרויקט הנוכחי השתבש)*

01: יצירת פרויקט הבאת נכסים ויצירת הסצנה הראשונה

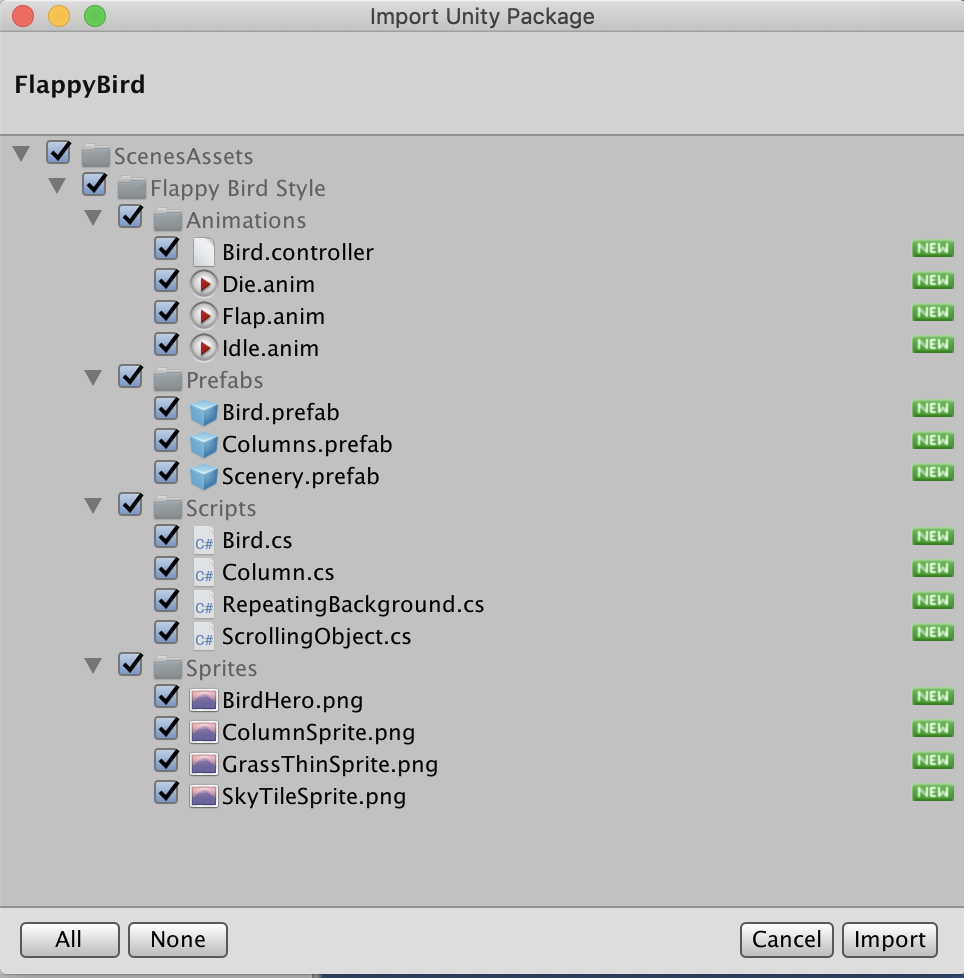
**צעד 1**: הזנק את האפליקציה UNITY ובחר ב new:

* בשדה שם הפרויקט הקלד: FlappyBird.
* בחר מיקום לשמירת הפרויקט
* בשדה התבנית (Template) בחר: 2D
* הקש על יצירת פרויקט

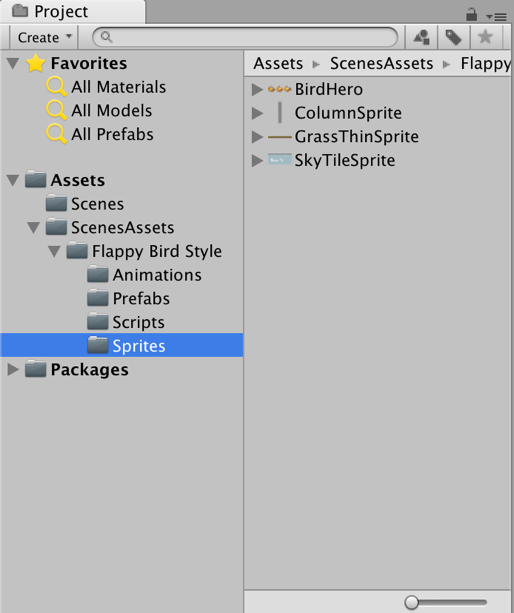


**צעד 2**: ייבא מהדיסק את הקובץ: FlappyBird.unitypackage שנמצא בתיקייה: Resources

* לייבא קבצים לתוך הפרויקט פשוט גרור את הקובץ לנכסי הפרויקט.
* UNITY יציג חלון ייבוא, לחץ ייבא (Import)

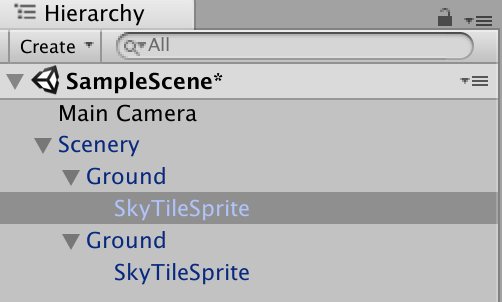


* נקבל את הקבצים הבאים:

 קיבלנו מספר ספריות חדשות בתוך הפרויקט שמכילות:

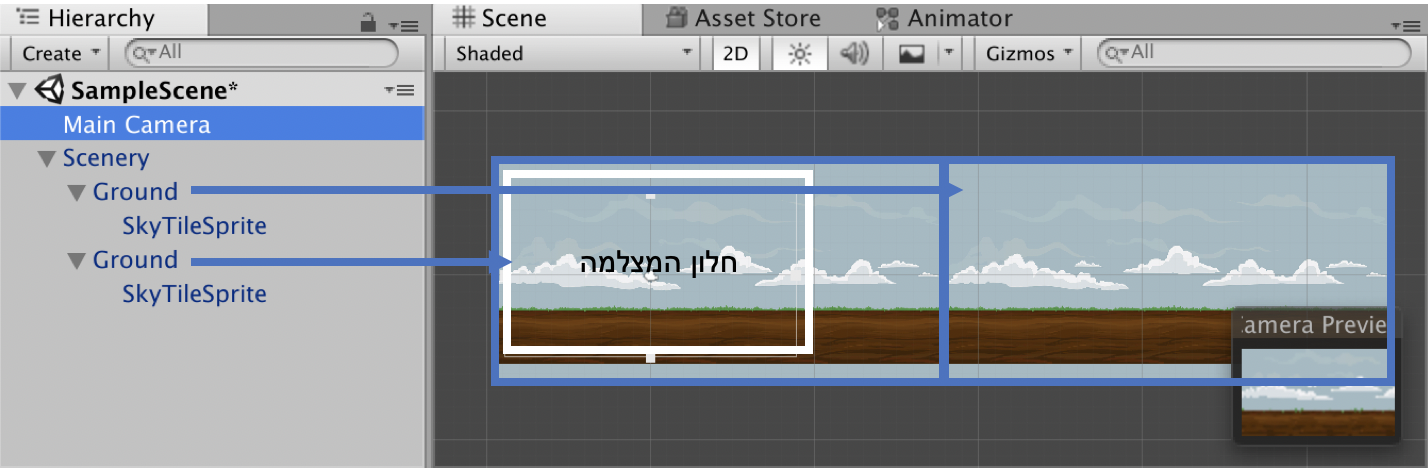
* Animations מכיל אנימציה של הציפור
* Prefabs מכיל אובייקטים מוכנים מראש: ציפור, עמודים, וגרפיקת רקע.
* Scripts מכיל קבצי קוד שבהמשך נשתמש ונרחיב
* Sprites מכיל קבצי תמונה שבשימוש ע״י האנימציות והאובייקטים המוכנים מראש.

**צעד 3:** גרור את הScenery שנמצא בתיקיית הPrefab לתוך חלון היררכיית הסצנה. נקבל את המבנה ההיררכי הבא: הרחב את ההיררכיה של הScenery ע״י לחיצה על המשולש הקטן  לקבלת המצב הבא. שים לב שישנם שני אובייקטים זהים בשם Ground – זה יאפשר לנו לבנות רקע נגלל אין סופי.



הרץ את המשחק ובחן כיצד הרקעים נגללים ומתחלפים. על פעולה זו אחראי הקוד: RepeatingBackground אשר נמצא כרכיב (Component) על אובייקט Ground

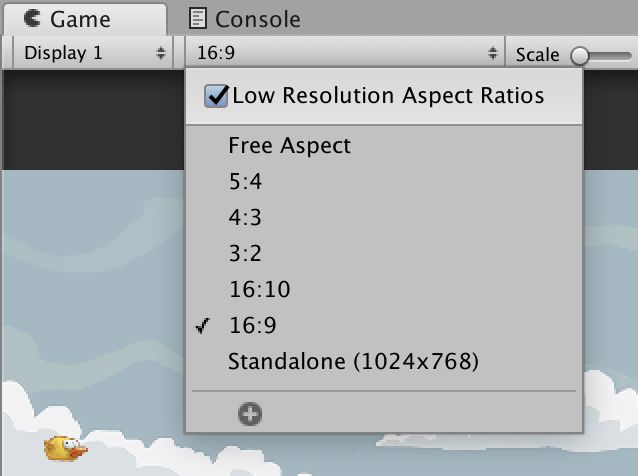
הסבר:



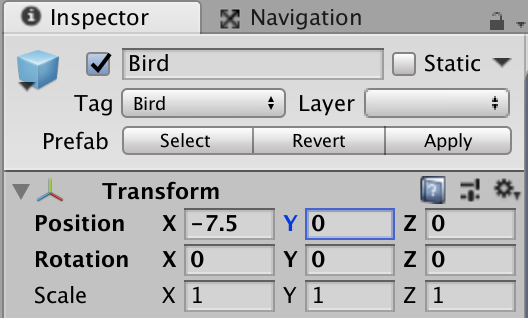
חלון המצלמה נשאר קבוע בזמן שהרקע נגלל.

שיכפול הרקע מאפשר להזיזו משמאל לימין בכל פעם שהמצלמה רואה רק את הרקע מימין.

\* בכדי שהסצנה תוצג כראוי בחלון המשחק, יש לוודא שיחס הרוחב לאורך של המצלמה (Aspect-Ratio) הוא 16:9. נשנה זאת בחלון המשחק:



**צעד 4:** גרור את הBird שנמצא בתיקיית הPrefab לתוך חלון הסצנה. מקם את הציפור בקורדינטות (-7.5, 0). העזר במפקח (Inspector) למקם את הציפור:



הרץ את המשחק. הציפור תיפול לקרקע ואנימציית ״פסילה״ (Die) תנוגן. הסיבה שבגללה הציפור נופלת היא הפיסיקה של המשחק. (ראה הסבר בשקפי המרצה).

02: שליטה על הציפור והוספת פיזיקה למשחק

**צעד 5:** הוספת שליטה של השחקן בציפור. פתח את הקוד Bird (הקוד נמצא בתיקיה Scripts שבחלון הפרוייקט) בעורך – ע״י הקשת עכבר כפולה על הקובץ. בעורך, הוסף את הקוד הבא התוך הפונקציה FixedUpdate :

**if** (Input.GetMouseButtonDown(0)) {

*//...tell the animator about it and then...*

 anim.SetTrigger("Flap");

*//...zero out the birds current y velocity before...*

 rb2d.velocity **=** Vector2.zero;

 rb2d.AddForce(**new** Vector2(0,130));

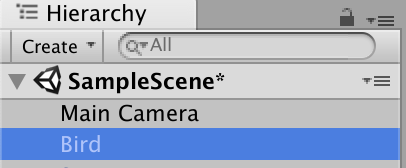
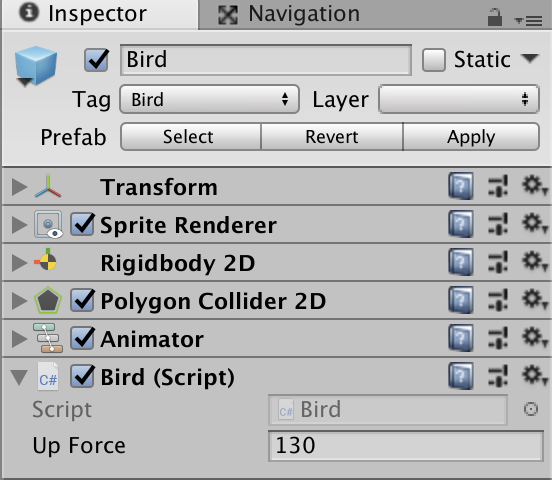
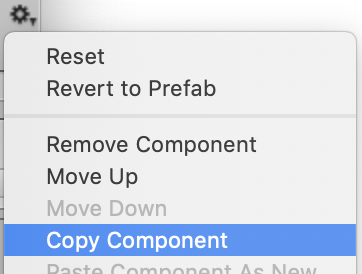
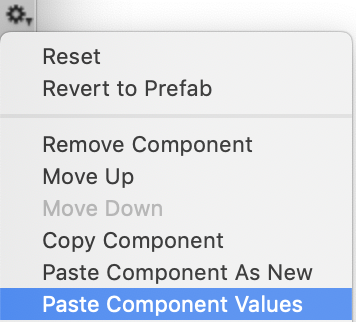
}

הקוד בודק כל Update של מנוע המשחק האם השחקן הקיש על כפתור העכבר השמאלי. במידה שכן: מפעיל את האנימציה Flap, דואג לעצור את תזוזת הציפור ע״י שינוי הvelocity של רכיב הRigidBody ל0. לבסוף מוסיף כח לרכיב הRigidBody בכיוון ״למעלה״ בכדי לדמות נענוע כנף של הציפור. מנוע הפיזיקה של UNITY ידמה הוספת כח בדיוק לפי חוקי התנועה של ניוטון.

**צעד 6:** **אתגר**: כאשר הציפור מתנגשת בקרקע – הוסף קוד שמתעלם מהקלט של השחקן. רמז: הפונקציה OnCollisionEnter2D מטפלת במקרה של התנגשות.

**צעד 7:** הוסף משתנה public בשם upForce לBird מסוג float ואתחל אותו ל130. (בC# ניתן לבצע אתחול משתנים בשורת ההגדרה: public float upForce = 130) החלף את המספר 130 בקריאה לפונקציה AddForce במשתנה החדש. (הקריאה לפונקציה AddForce מתבצעת בתוך הפונקציה FixedUpdate שאותה עדכנו בצעד 5). עכשיו נוכל לשנות את עוצמת הכח של נענוע הכנף דרך חלון המפקח (הגדרת ה public חושפת את המשתנה למפקח). נוכל לעשות זאת גם תוך כדי ריצה! שים לב ששינויים בזמן ריצה לא נשמרים.

שינוי משתנים במפקח:

1. בחר את אובייקט הציפור בהיררכיה  
   
2. במפקח גלול לרכיב Bird בכדי לשנות את הupForce  
   
3. ניתן לבצע שינויים גם בזמן ריצה! 
4. בכדי לשמור שינויים שבוצעו בזמן ריצה – לחיצת עכבר שמאל על גלגל השיניים ובחר: Copy Component. עצור את הריצה וכעת בחר Paste Component Values.   
    

\* קומפילציה (הידור) ב Unity מתבצעת ברקע והרצה מתאפשרת רק בסיומה בהצלחה. במידה שיש שגיאות בקוד ניתן לראותן החלון ה Console. ניתן להציג חלון זה דרך התפריט: Windows/General/Console.

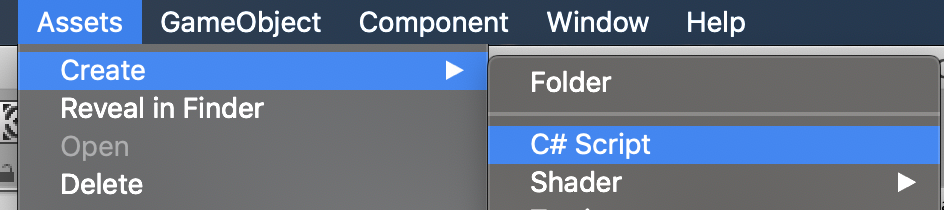
\* ניתן לעצור (pause) את ריצת המשחק בכדי לשנות את ה upForceבמפקח ע״י קיצור המקשים: Ctrl+Shift+P

**צעד 8:** **אתגר**: הוסף אובייקט חדש לסצנה שיהווה ״תקרה״ למקרה שהציפור יוצאת מחוץ לגבולות המסך ״למעלה״. רמז: האובייקט Ground הוא דוגמא טובה.

**צעד 9:** אתגר: חפש בדפי העזרה של unity באינטרנט איך לשנות את השליטה על הציפור (או להוסיף במקביל) מעכבר שמאל למקש רווח. האתר: <https://docs.unity3d.com/ScriptReference/index.html> וחפש עזרה על האובייקט Input.

**צעד 10:** הוספת העמודים.

הוסף קובץ סקריפט חדש בשם ColumnSpawn שיהיה אחראי על יצירת העמודים. אחת הדרכים להוסיף קובץ חדש לפרוייקט היא ע״י שימוש בתפריט של התוכנה (קובץ הסקריפט החדש ייווצר בתיקיה הנוכחית בפרויקט, במידה והוא נוצר בתיקייה שונה מאשר התיקייה: Scripts ניתן לגרור אותו לשם. אין חשיבות למיקום בתיקיות בפרויקט):



פתח את הקובץ החדש והכנס את הקוד הבא:

public **class** ColumnSpawn : MonoBehaviour {

    public GameObject columnPrefab;*//The column game object.*

    public **float** spawnRate **=** 3f; *//How quickly columns spawn.*

    public **float** columnMin **=** **-**1f;*//Minimum y value of the column position.*

    public **float** columnMax **=** 3.5f; *//Maximum y value of the column position.*

    private **float** spawnXPosition **=** 10f;

    private **float** timeSinceLastSpawned;

*// Use this for initialization*

**void** Start () {

        timeSinceLastSpawned **=** spawnRate;

    }

*// Update is called once per frame*

**void** Update () {

        timeSinceLastSpawned **+=** Time.deltaTime;

**if** (timeSinceLastSpawned **>=** spawnRate)

        {

            timeSinceLastSpawned **=** 0f;

*//Set a random y position for the column*

**float** spawnYPosition **=** Random.Range(columnMin, columnMax);

*// Spawn the object and set position.*

            GameObject column **=** (GameObject)Instantiate(columnPrefab, **new** Vector2(spawnXPosition, spawnYPosition) , Quaternion.identity);

        }

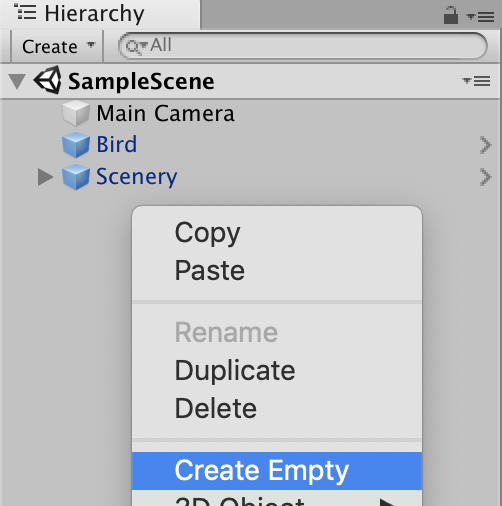
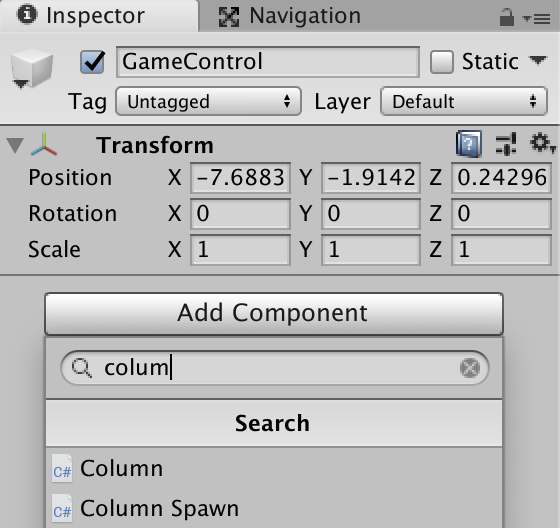
    }

}

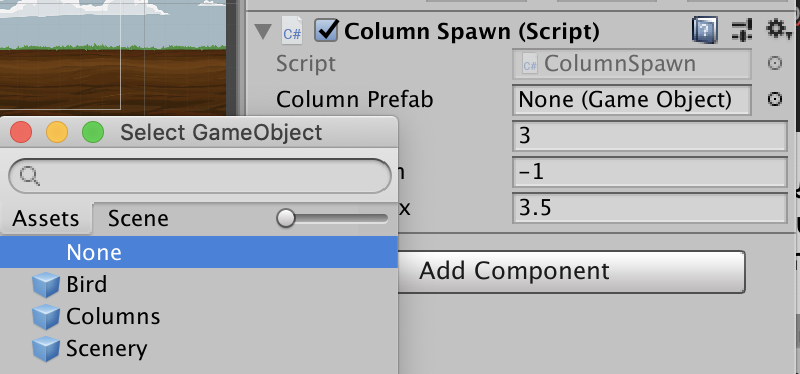
הקוד מכיל מספר משתנים שקובעים את ההתנהגות של יצירת העמודים: כל כמה זמן: spawnRate, גבולות מיקום: columnMin, columnMax ואיזה אובייקט ליצר: columnPrefab.

\* ב C# כאשר מאתחלים משתנה מסוג float בערך בעל נקודה עשרונית, יש להוסיף את הסיומת f למספר.

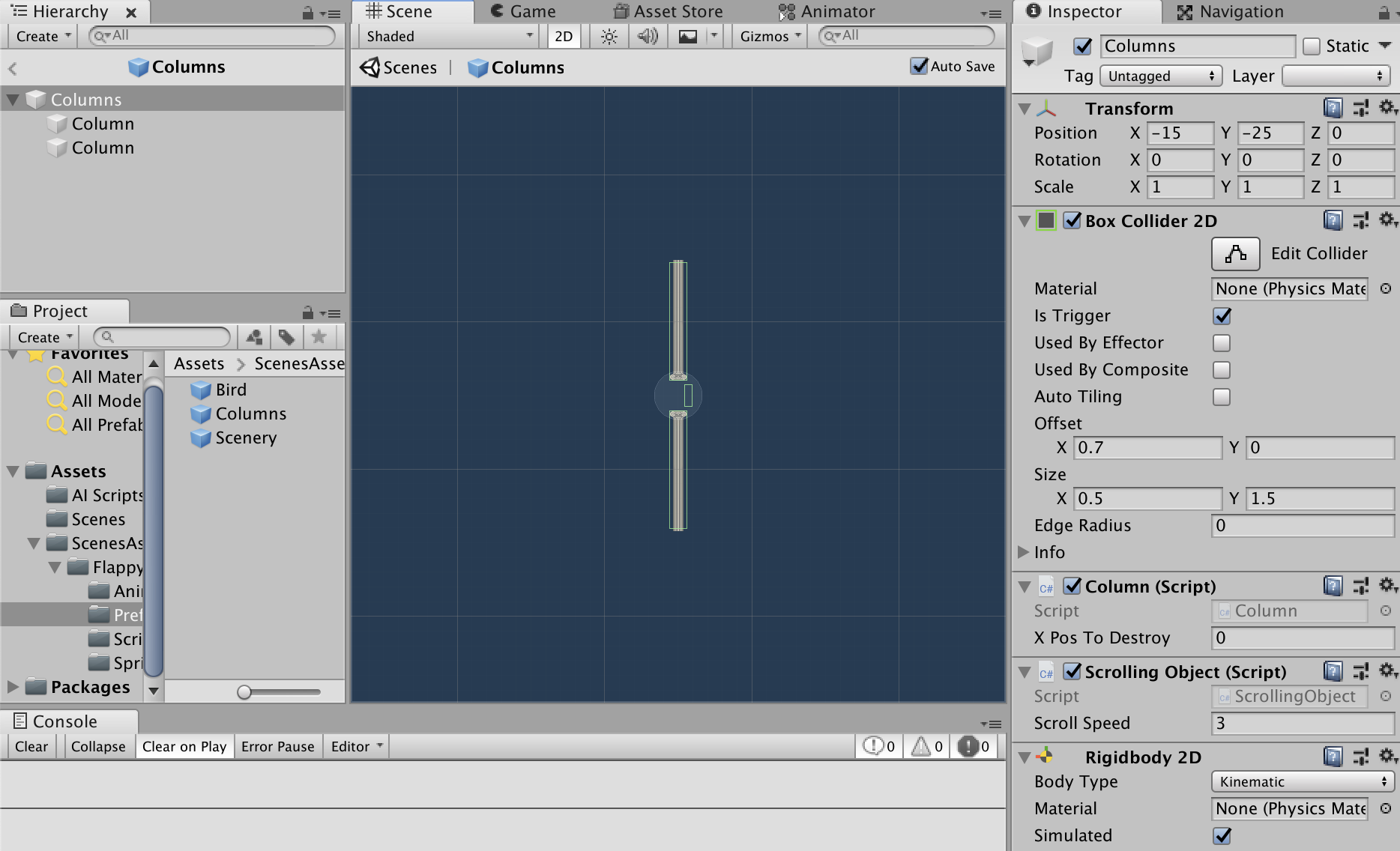
**צעד 11:** צור אובייקט ריק חדש בחלון היררכית הסצנה (שנה את שם האובייקט לGameControl) והוסף את הסקריפט החדש אליו.

**צעד 12:** בחר את האובייקט Columns בחלון הסצנה. בחלון העורך נבצע השמה של הprefab בשם Columns עבור המשתנה columnPrefab של הסקריפט ColumnSpawn (לחץ על העיגול הקטן ליד השדה, ראה תמונה):



האובייקט Columns הינו Prefab במונחי unity. המשמעות היא שזה אובייקט מוכן מראש שניתן ליצר באופן דינמי דרך סקריפט או ע״י גרירה לסצנה (כמו שעשינו עם הציפור). אם נתבונן מה מכיל האובייקט ע״י לחיצה עליו בחלון הפרויקט (תחת התיקייהScenesAssets/Flappy Bird Style/Prefabs) ואז פתיחתו ע״י לחיצה על open prefab בחלון המפקח.



האובייקט מכיל את האלמנטים הגרפיים של העמודים, את האלמנטים הפיסיקליים (BoxCollider2D וגם RigidBody2D) וכן שני קבצי סקריפט: אחד שגולל את העמודים בדיוק כמו גלילת הרקע והשני שישמש אותנו בהמשך לזיהוי מעבר של הציפור דרך שני העמודים בצורה מוצלחת.

הרץ את המשחק ובדוק את יצירת העמודים, נסה לעבור בין העמודים.

03: הוספת לוגיקת משחק

**צעד 13:** בשלב זה נוסיף סקריפט נוסף לאובייקט GameControl בשם GameLogic, נרצה שבמשחק תהיה קיימת רק ישות יחידה מהסוג הזה. נעשה זאת ע״י שימוש במשתנה static שיחזיק מצביע (pointer) לאותה ישות ויהיה זמין לכל שאר הסקריפטים במשחק.

הוסף סקריפט חדש בשם GameLogic והוסף אותו בComponent לאובייקט GameControl בהיררכית הסצנה (האובייקט GameControl התווסף בשלב הקודם).

public **class** GameLogic : MonoBehaviour {

    private static GameLogic instance **=** null;

    public static GameLogic GetInstance() { **return** instance; }

**void** Awake() {

        instance **=** **this**;

    }

**void** Start () {

    }

}

הסבר קצר על הפונקציות השכיחות של MonoBehaviour (כל סקריפט שנוסיף לפרויקט יורש מממנו):

Start: הפונקציה נקראת פעם אחת בלבד, בפריים (frame) הראשון כאשר האובייקט מאופשר (enabled) ולפני Update.

Awake: הפונקציה נקראת פעם אחת בלבד, כאשר האובייקט נוצר, לפני Start.

Update: הפונקציה נקראת כל פריים (frame).

FixedUpdate: הפונקציה נקראת כל פריים (frame) בקצב קבוע ומשמשת בעיקר לקריאה לפונקציות שקשורות למנוע הפיזיקלי.

**צעד 14:** כעת נוסיף משתנה score לציפור שנוכל לגשת אליו מהGameLogic. הסיבה שנוסיף משתנה זה לציפור ולא ללוגיקת המשחק היא שבהמשך נרצה מספר רב של ציפורים שלכל אחת יהיה score משלה. הוסף משתנה זה כרגע.

**צעד 15**: נוסיף פונקציה חדשה לGameLogic בשם BirdScore שמקבלת את הBird בתור משתנה. בתוך הפונקציה נוסיף 1 לscore הנוכחי של הציפור וגם נוסיף הדפסה שנראה שהפונקציה נקראת.

public **void** BirdScored(Bird bird) {

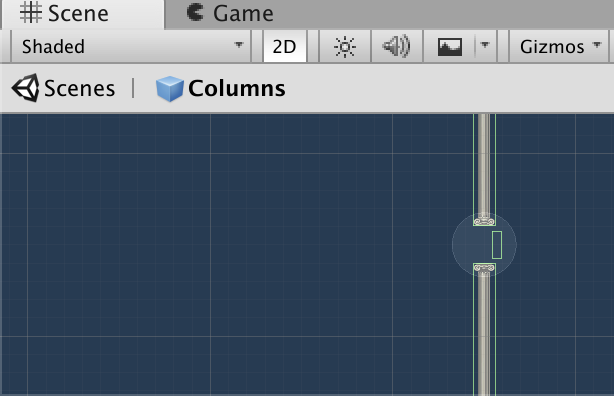
        bird.score **+=** 1;

*Debug.Log("Score: " + bird.score);*

    }

**צעד 16:** כעת נקרא לפונקציה הזאת כאשר הציפור עברה דרך שני העמודים בהצלחה. אתגר: נסה לממש זאת לפני קריאת הפתרון למטה.

הprefab של ה Columns מכיל BoxCollider2D בין שני העמודים שמשמש כטריגר בלבד (בניגוד לשני BoxCollider2D על גבי העמודים עצמם שמשמשים כאלמנט פיזי שחוסם את הציפור).



BoxCollider2D שמשמש כטריגר בלבד

פתח את הסקריפט Column והוסף קריאה לBirdScore בפונקציה שמקבלת את הטריגר:

**void** OnTriggerEnter2D(Collider2D other)

    {

        Bird bird **=** other.GetComponent<Bird>();

**if**(bird **!=** null)

        {

GameLogic.GetInstance().BirdScored(bird);

        }

    }

המשתנה other מייצג את האובייקט שגרם לטריגר. בעזרת הפונקציה GetComponent של unity נוכל לקבל את ישות (instance) הסקריפט של הציפור שגרמה לטריגר. ואז נקרא לפונקציה הstatic של הGameLogic שיצרנו קודם.

הרץ את המשחק ובדוק את ההדפסות שהוספנו בפונקציה BirdScore לתוך חלון הConsole. ניתן להציג חלון זה דרך התפריט: Windows/General/Console.

צ**עד 17:** סיום המשחק.

כאשר הציפור נתקעת בעמוד נרצה להפסיק את גלילת הרקע.

הוסף משתנה חדש לGameLogic בשם gameOver וקבע את ערך האתחול שלו לfalse. הוסף פונקציה חדשה לGameLogic שנקראת BirdDied שמקבלת Bird בתור משתנה. הפונקציה תקבע את משתנה הgameOver לtrue.

בסקריפט של הBird קרא לפונקציה BirdDied של הGameLogic כאשר הציפור נתקעת בעמודים, בפונקציה: OnCollisionEnter2D.

בסקריפט ScrollingObject בפונקצית הUpdate בדוק את משתנה gameOver של הGameLogic ואם המשחק הסתיים שנה את הvelocity של הrigidbody לאפס בכדי להפסיק את התזוזה:

**void** Update()

    {

*// If the game is over, stop scrolling.*

**if** (GameLogic.GetInstance().GameOver **==** true) {

            rb2d.velocity **=** Vector2.zero;

        }

    }

בדוגמה למעלה, יצרנו readonly property למשתנה gameOver בGameLogic בצורה הבאה:

private **bool** gameOver **=** false;

    public **bool** GameOver {

**get** { **return** gameOver; }

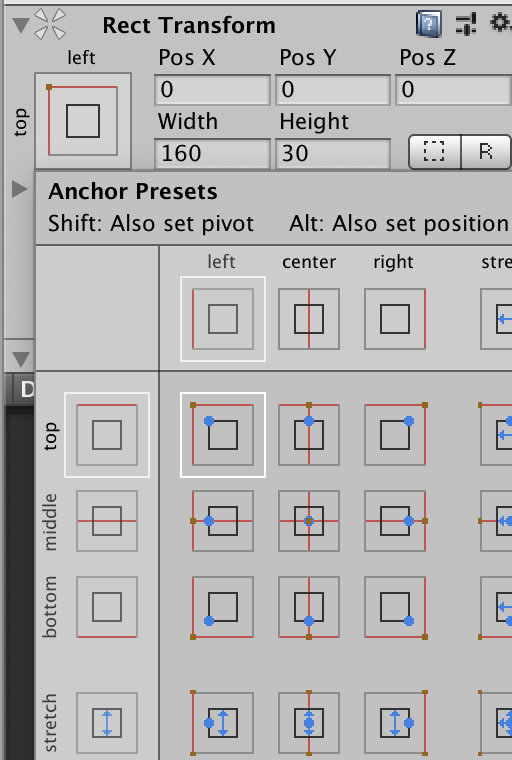
    }

הרץ את המשחק.

צ**עד 18:** **אתגר**: בכדי למנוע היווצרות אינסופית של עמודים, נרצה להוריד את האובייקט מהזיכרון ע״י שימוש בפקודת Destroy של unity. נסה למצוא דרך להוסיף זאת לתוך הסקריפט של Column.

**צעד 19:** הוספת כתוביות של ניקוד וסיום משחק. הוסף לסצנה אובייקט חדש מסוג: UI/Text. שים לב שנוצרו שני אובייקטים חדשים: Canvas וEventSystem. שני אובייקטים אלו משמשים לטיפול בכתוביות. בחר את האובייקט Text ושנה את שמו ל ScoreText. במפקח שנה את הפרטים הבאים:

תחת הכותרת RectTransform לחץ על המלבן שקובע את מיקום נקודות העוגן של הכתובית ולחץ למקמו בפינה השמאלית עליונה (השתמש בshift+alt בזמן הלחיצה):



לחץ למקמו בפינה השמאלית עליונה השתמש ב shift+alt

לחץ על המלבן שקובע את מיקום נקודות העוגן

שנה את רוחב הכתובית ל160.

שנה את גודל הגופן ל24 ואת הצבע לאדום.

שנה את השדה text לscore: 0.

בסקריפט GameLogic נוסיף שימוש ב UnityEngine.UI בראש הקובץ ע״י שימוש במילת הקוד: using. הצהרה זו תאפשר לנו להשתמש בפקודות ששולטות בכתוביות.

בסקריפט GameLogic נוסיף משתנה חדש להצביע על הכתובית שיצרנו: public Text scoreText.

בסקריפט GameLogic בפונקציה BirdScored עדכן את הכתובית בתוצאה החדשה ע״י הוספת השורה הבאה:

scoreText.text **=** "Score: " **+** bird.score;

הרץ את המשחק ובדוק את התוצאה.

**אתגר:** הוסף כתובית נוספת לסיום המשחק כך שבתחילתו הכתובית תהיה ריקה ובסיום המשחק היא תכיל את ההודעה: Game Over.

04: הוספת כתובית להתחלת המשחק

*\* בחלק זה נבצע שינויים בקוד שיאפשרו לאתחל את המשחק לאחר פסילה. ניתן לדלג על חלק זה ע״י פתיחת הפרויקט FlappyBirdAI04 מהדיסק.*

**צעד 20:** הוספת כתובית להתחלת המשחק.

**אתגר**: עשה שימוש נוסף בכתובית סיום המשחק מהצעד הקודם כך שבתחילת המשחק תופיע הודעה ללחוץ על מקש Enter להתחלה. הנחיות כלליות: ניתן לעצור את הסימולציה הפיזיקלית של המשחק ע״י שימוש בTime.timeSclae של unity והשמתו ל0 בתחילת המשחק. הוסף משתנה isPaused שיאפשר לבדוק אם המשחק בעצירה או לא. בסקריפט GameLogic בצע בדיקה אם המשחק עצר ובדוק קלט של השחקן להתחלת המשחק.

**צעד 21:** אתחול מחדש.

בצעד זה נרצה לאתחל את המשחק לאחר פסילה של השחקן ולהתחיל מחדש את המשחק. לשם כך נרצה להוסיף פונקציית אתחול לסקריפטים הבאים: Bird, GameLogic, וColumnSpawn.

**אתגר**: נסה לכתוב פונקציות אלה בעצמך!

פונקציית אתחול של הBird:

public **void** Restart() {

        transform.position **=** initialPosition;

        isDead **=** false;

        score **=** 0;

        rb2d.velocity **=** Vector2.zero;

        transform.rotation **=** Quaternion.identity;

        rb2d.angularVelocity **=** 0;

    }

שים לב שבפונקציה שלמעלה אנו מאפסים את המהירות של הציפור, גם מהירות בציר הx,y וגם מהירות הסיבוב.

פונקציית אתחול של הColumnSpawn:

public **void** Restart() {

        timeSinceLastSpawned **=** spawnRate;

        List<Transform> childList **=** **new** List<Transform>();

**for** (**int** i **=** 0; i **<** transform.childCount; i**++**) {

            childList.Add(transform.GetChild(i));

        }

**foreach** (**var** trans **in** childList)

        {

            Destroy(trans.gameObject);

        }

    }

פונקציה זו משתמשת בפונקציות פנימיות של unity בכדי למצוא את כל הילדים בהיררכיה של האובייקט שאחראי על יצירת העמודים: GameControl ומוחקת את כולם.

לבסוף ניתן את התוכן הנוכחי של הGameLogic סקריפט:

private const **string** WellcomeText **=** "Flappy Bird, press Return to start.";

    private const **string** GameOverText **=** "Game Over, press Return to start.";

*// Use this for initialization*

**void** Awake() {

        instance **=** **this**;

    }

**void** Start () {

        centerText.text **=** WellcomeText;

        Time.timeScale **=** 0;

        gamePaused **=** true;

    }

*// Update is called once per frame*

**void** Update () {

**if** (gamePaused **==** true) {

**if** (Input.GetKeyDown(KeyCode.Return)) {

**if** (gameOver **==** true) {

*// Restart game:*

*// 1. restart Bird:*

                    bird.Restart();

                    GetComponent<ColumnSpawn>().Restart();

                }

                centerText.text **=** "";

                Time.timeScale **=** 1;

                gamePaused **=** false;

            }

        }

    }

    public **void** BirdScored(Bird bird) {

        bird.score **+=** 1;

scoreText.text **=** "Score: " **+** bird.score;

    }

    public **void** BirdDied() {

  gameOver **=** true;

        centerText.text **=** GameOverText;

        gamePaused **=** true;

        Invoke("PauseTime", 1);

    }

**void** PauseTime() {

**if** (gamePaused**==**true)

            Time.timeScale **=** 0;

    }

בקוד למעלה השתמשנו בשירות נוסף שunity נותן לנו בפקודת: Invoke. הפקודה מאפשרת לקרוא לפונקציה מסוימת לאחר זמן מוגדר. מה שמאפשר לנו לתת עוד שניה של זמן למנוע הפיזיקלי של המשחק לנוע ולציפור ליפול לרצפה לפני שעצרנו את המנוע ע״י הפקודה: Time.timeScale=0.

05: הכנות לחיבור המשחק לבינה מלאכותית ואלגוריתם פשוט ראשוני

צעד 22: הוספת אלגוריתם פשוט.

הוסף סקריפט חדש בשם AiLogic. המחלקה (class) החדשה שיצרנו לא צריכה לרשת מMonoBehaviour . הוסף פונקציה חדשה שתקרא FlapWings שמקבלת שני פרמטרים ובהתבסס על פרמטרים אלו תחליט אם ל״נפנף בכנפיים״ או לא:

public **class** AILogic {

    public **bool** FlapWings(**float** param1, **float** param2) {

**if** (param2 **<** 0 **&&** UnityEngine.Random.Range(0,100)**>**90f) {

**return** true;

        }

**if** (param2 **>=** 0 **&&** UnityEngine.Random.Range(0,100)**>**97f) {

**return** true;

        }

**return** false;

    }

}

הפונקציה שלמעלה לא משתמשת בשני הפרמטרים וגם לוקחת החלטות בצורה רנדומלית. בהמשך יינתן אתגר לשיפור הפונקציה.

צעד 23: חיבור לסקריפט של הציפור.

בסקריפט של הציפור צור משתנה חדש: AILogic ai = new AILogic(). בפונקציית הUpdate במקום בדיקה של הקלט מהמשתמש, קרא ל

ai.FlapWings(transform.position.x, transform.position.y) בכדי לבדוק אם ל״נפנף בכנפיים״ או לא.

הרץ את המשחק ובדוק את התוצאה.

אתגר: נסה לשפר את ביצועי האלגוריתם. מה לדעתך חסר בפרמטרים שהעברנו לאלגוריתם בכדי לתת פתרון סביר?

צעד 24: הכנות באובייקט הBird לפני יצירת עותקים רבים שירוצו בו זמנית.

נשנה את הפונקציה Awake של סקריפט הBird:

**void** Awake() {

*//Get reference to the Animator component attached to this GameObject.*

        anim **=** GetComponent<Animator> ();

*//Get and store a reference to the Rigidbody2D attached to this GameObject.*

        rb2d **=** GetComponent<Rigidbody2D>();

**float** xPosition **=** UnityEngine.Random.Range(**-**8f, **-**6.5f);

**float** yPosition **=** UnityEngine.Random.Range(**-**2f, 2f);

        initialPosition **=** **new** Vector2(xPosition, yPosition);

        gameObject.GetComponent<SpriteRenderer>().color **=** Random.ColorHSV();

        transform.position **=** initialPosition;

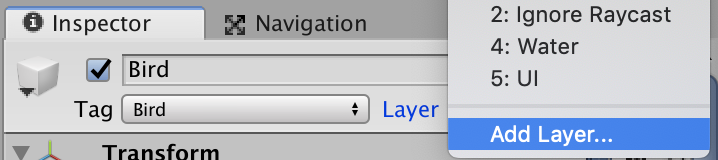
    }

הקוד שהוספנו ייצר את האובייקט של הציפור בכל פעם במקום קצת שונה ובצבע אחר.

בנוסף נרצה למנוע מציפורים להתנגש אחת בשנייה. כרגע המנוע הפיזיקלי של unity לא יודע להבדיל וכל Collider שישנו במשחק יתנגש עם כל Collider אחר. למנוע זאת נבצע את שני השינויים הבאים:

1. ניצור Layer חדש בשם Bird ונשנה את הprefab של הציפור לLayer הזה.

הוספת Layer ע״י בחירת הprefab של הציפור, פתיחתו ובחירה בתפריט Layer. בתפריט נבחר בAdd Layer….

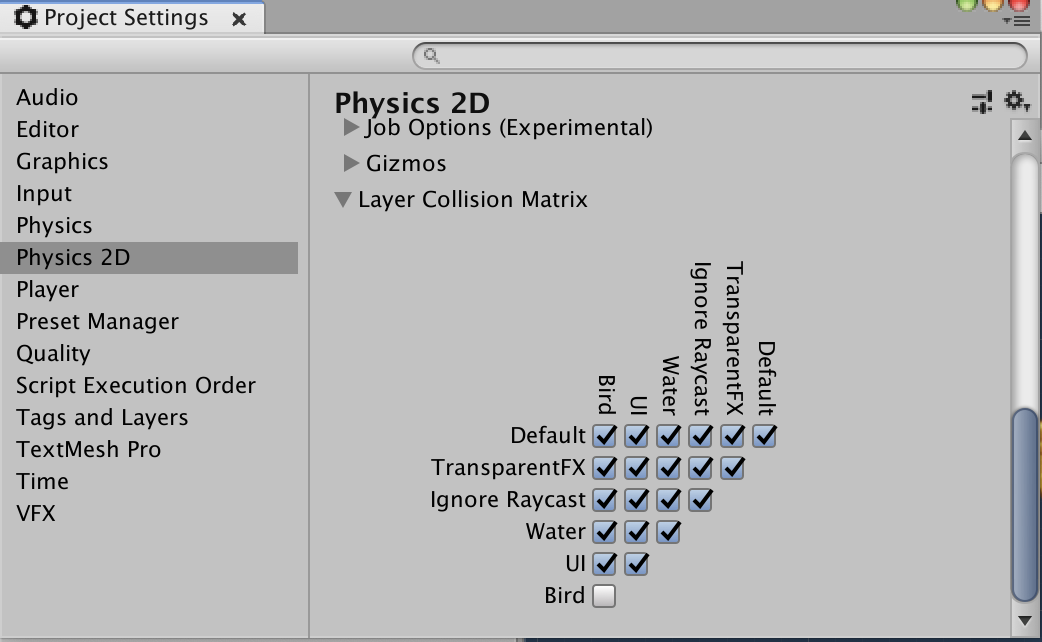


פעולה זו תפתח את מסך המפקח על עריכת Tags & Layers. נבחר במקום פנוי בטבלת הLayers וניתן שם חדש לLayer בשם Bird.

כעת נחזור לprefab של הציפור ונבחר בLayer החדש.

1. בהגדרות ה Physics2d תחת מטריצת הCollision נוודא שהLayer של הBird לא מתנגש עם עצמו.

בכדי לשנות הגדרות אלו נפתח את חלון הProject Settings מתפריט הEdit. נבחר את האופציות של Physics 2D ומשם נגלול למטריצת הLayer Collision Matrix ונבטל את האפשרות של הBird להתנגש בעצמו.



ביטול האפשרות של הBird להתנגש בעצמו.

צעד 25: יצירת מספר עותקים של הציפור.

בשלב ראשון נמחק את אובייקט הציפור מהסצנה – נייצר עותקים של הציפור בקוד תוך שימוש בprefab של הציפור.

בתוך הסקריפט של הGameLogic נוסיף רשימה שתחזיק את העותקים של הציפורים:

List<Bird> birds = new List<Bird>(). (מחק את המשתנה שהחזיק את העותק היחיד של הציפור). שימוש בList מצריך הוספת הצהרת שימוש בראש הסקריפט:

using System.Collections.Generic.

נוסיף גם משתנה חדש שיחזיק את העותק של הprefab של הציפור:

public GameObject birdPrefab. ומשתנה של כמות הציפורים שנרצה ליצר:

public int numberOfBirds

. ניצור פונקציה חדשה שתיצור את הציפורים:

**void** InstantiateBirds() {

**for** (**int** i **=** 0; i **<** numberOfBirds; i**++**) {

            birds.Add(Instantiate(birdPrefab, Vector2.zero, Quaternion.identity).GetComponent<Bird>());

        }

    }

נתקן את הפונקציה Restart שתעבור על כל העותקים של הציפורים:

**foreach** (**var** bird **in** birds)

    {

        bird.Restart();

    }

הרץ את המשחק ובדוק את התוצאות.

**אתגר**: נוצרו כעת שתי בעיות: 1) הscore לא מייצג נכונה את התוצאה הגבוהה ביותר. 2) המשחק מסתיים שהציפור הראשונה מתה. תקן את שתי הבעיות הללו. התיקון הכרחי להמשך.

**אתגר**: נסה לשפר את ביצועי האלגוריתם כעת שיש מספר רב של עותקים.

06: חיבור לאלגוריתם גנרי עם רשת עצבית

צעד 26: ייבא מהדיסק את הקובץ: ai.unitypackage שנמצא בתיקייה: Resources. ייבוא זה הוסיף לפרויקט שלושה סקריפטים בתיקיה חדשה שנקראת AI Scripts:

* Genetic
* Mutation
* NeuralNetwork

קבצים אלו ממשים שני אלגוריתמים שנשתמש בהם: אלגוריתם גנרי ורשת עצבית (נוירונים).

צעד 27: שינויים בסקריפט הציפור.

1. הוסף משתנה חדש שמחזיק את האינדקס של ישות הציפור הנוכחית בתוך האלגוריתם הגנרי: public int mutationIndex
2. וודא שמשתנה isDead הוא public כדי לאפשר גישה אליו מסקריפטים אחרים
3. הוסף משתנה חדש: public bool flapOnNextFrame = false שישמש את האלגוריתם הגנרי.
4. שנה את פונקציית הFixedUpdate לבדוק האם המשתנה flapOnNextFrame הוא true ואם כן אז להניע את הכנפיים. דאג להחזיר את הערך של המשתנה לfalse.
5. וודא שאם המשתנה isDead הוא true אז נצא מפונקציית העדכון FixedUpdate לפני ביצוע כל פעולה.

צעד 28: שינויים בסקריפט GameLogic:

1. הוספת משתנים public חדשים לתמיכה באלגוריתם הגנרי:  
   int numOfUnitsInGeneration = 40  
   float topUnitsPercentage **=** 0.25f   
   float randomUnitsPercentage = 0.10f  
   משתנים אלו מגדירים את גודל האוכלוסייה בכל דור, את כמות האוכלוסייה שלא תעבור מוטציה בין דורות ואת כמות האוכלוסייה שתשתנה באופן רנדומלי כל דור.
2. הוסף משתנה שיעקוב אחרי מספר הדורות: int generationNum = 0
3. הוסף משתנה חדש שיעקוב אחר המרחק שהציפורים החיות עוברות: float distance
4. עדכן משתנה זה בפונקציית הUpdate ע״י הוספה של Time.deltaTime בכל עדכון. זה אמנם לא באמת המרחק, אבל משתנה זה יאפשר לנו לדרג את הציפורים.
5. הוסף משתנה של האלגוריתם הגנרי: Genetic genetic. את האתחול של האלגוריתם הגנרי נבצע בStart ע״י:  
   genetic = new Genetic(numOfUnitsInGeneration, topUnitsPercentage randomUnitsPercentage, CreateNetwork )   
   השתמשנו במשתנים שהגדרנו מוקדם יותר לאתחול. הפרמטר CreateNetwork הוא מצביעה לפונקציה שמאפשרת לייצר רשת נוירונים – נגדיר את הפונקציה בשלב הבא.
6. הפונקציה CreateNetwork:

Mutation CreateNetwork(Mutation parent**=**null) {

**return** **new** NeuralNetwork(2, 1, **new** **int**[1]{6});

    }

פונקציה זו מייצרת רשת נוירונים בעלת שתי כניסות ויציאה אחת ובנוסף תוואי חבוי (hidden layer) ברוחב 1 וגודל 6.

1. שנה את הפונקצייה: InstantiateBirds כך שנאתחל את המשתנה mutationIndex של הציפור לפי האינדקס של הלולאה:

**void** InstantiateBirds() {

**for** (**int** i **=** 0; i **<** numOfUnitsInGeneration; i**++**) {

            Bird bird **=** Instantiate(birdPrefab, Vector2.zero, Quaternion.identity).GetComponent<Bird>();

            birds.Add(bird);

            bird.mutationIndex **=** i;

        }

    }

1. כשהציפור מתה הוסף את השורה הבאה:

genetic.SetFitness(bird.mutationIndex, distance);

שורה זו תעדכן את ערך הfitness (הכשירות) של הציפור לפי המרחק שהיא עברה.

צעד 29: שינוי פונקציית הUpdate. זהו השינוי המשמעותי ביותר בGameLogic והוא ניתן בשלמותו:

**void** Update () {

**if** (**!**gameOver) {

            distance **+=** Time.deltaTime;

**for** (**int** i **=** 0; i **<** numOfUnitsInGeneration; i**++**) {

**if** (birds[i].isDead **==** false) {

**float**[] inputs **=** **new** **float**[2]; *// prepare the genetic activate function inputs*

                    columnSpawn.GetNearestColumn(birds[i].transform, out inputs[0], out inputs[1]);

**float**[] result **=** genetic.ActivateBrain(i, inputs);

**if** (result[0] **>** 0.5f) {

                        birds[i].flapOnNextFrame **=** true;

                    }

                }

            }

        } **else** {

            genetic.EvolvePopulation();

            generationNum **+=** 1;

            Restart();

        }

    }

שני שינויים משמעותיים:

1. ההחלטה האם להניע בכנפי הציפור נעשית ע״י קריאה ל״מוח״ של האלגוריתם. פונקציה זו מצפה לקבל מערך של קלט - הגודל זהה למה שהגדרנו ברשת הנוירונים: גודל 2. אנו מאתחלים מערך זה ע״י קריאה לפונקציית עזר שנממש למטה: GerNearestColumn. פונקציה זו מחשבת את המרחק מהציפור לרווח של העמודים הקרובים, כך שנקבל את המרחק בx,y אל הרווח.
2. אם כל הציפורים מתו – נייצר אוכלוסייה חדשה ונקדם את מונה הדור.

צעד 30: הוסף את הקוד הבא לסקריפט: ColumnSpawn

**float** deltaX **=** 0.80f;

    public **int** GetNearestColumn(Transform from, out **float** horizontalDistance, out **float** heightDifference) {

**int** bestFound **=** **-**1;

        horizontalDistance **=** 30;

        heightDifference **=** from.position.y **-** 0.558f;

**for** (**int** i **=** 0; i **<** transform.childCount; i**++**) {

**float** testDistance **=** (transform.GetChild(i).transform.position.x**+**deltaX) **-** from.position.x;

**if** (testDistance **>**0 **&&** testDistance**<**horizontalDistance) {

                bestFound **=** i;

                horizontalDistance **=** testDistance;

                heightDifference **=** from.position.y **-** transform.GetChild(i).transform.position.y;

            }

        }

**return** bestFound;

    }

הרץ את המשחק.

**אתגר**: הצג כתובית של מספר הדור: generationNum.

**אתגר**: בטל את השימוש בpause, אין בו צורך יותר.

**אתגר**: הוסף תנאי חדש לסיום דור ואבולוציה של דור חדש: כאשר עוברים מרחק כלשהו.