A black background with white text

Description automatically generated

**מפענחים את הלוח: אסטרטגיות מתקדמות בסולמות ונחשים**

עבודת סיכום קורס

שם הקורס: אלגוריתמים מתקדמים לתכנון ותזמון מערכות נבונות

מספר הקורס: 350062

שנה: תשפ"ד סמסטר: ב'

שם המרצה: ד"ר רינה צביאל גירשין

שמות התלמידים: נתן יצחק פור, עדן בוארון, ליאור מאמוס, אורי פלה

מספרי תעודות זהות: 211412986, 314643610, 301730610, 314974858

תאריך הגשה: 03.08.2024

תוכן עניינים

[הלינקים המבוקשים 3](#_Toc173589916)

[הנחיות הרצה 3](#_Toc173589917)

[סיכום 4](#_Toc173589918)

[מבוא 4](#_Toc173589919)

[כללי המשחק 5](#_Toc173589920)

[מהלך המשחק 6](#_Toc173589921)

[פיתוח המשחק והאלגוריתמים 10](#_Toc173589922)

[הקשיים והפתרונות 13](#_Toc173589923)

[פיתוח בעזרת המנוע Unity 13](#_Toc173589924)

[בניית המשחק כדינמי 13](#_Toc173589925)

[בניית האלגוריתם וההערכות הנדרשות 14](#_Toc173589926)

[קוד הפרויקט 15](#_Toc173589927)

# הלינקים המבוקשים

1. למצגת לחץ [כאן](https://www.canva.com/design/DAGMy6VpB1U/21dbedtke0MtNtiIX9GJPA/edit?utm_content=DAGMy6VpB1U&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)

2. לסרט ההדגמה של המצגת לחץ [כאן](https://youtu.be/aUht0zSUx3M)

3. לסרט ההדגמה של המשחק לחץ [כאן](https://youtu.be/qXYGJ8WaTzA)

4. ל-GitHub לחץ [כאן](https://github.com/NatiIPImages/AdvancedAlgorithmsProject)

לא היה לנו מקום להעלות את הסרטונים ל-GitHub או למודל. בשניהם עברנו את המגבלה. אם תרצו להוריד את הסרטונים, העלנו אותם ל-Mega ב[קישור הבא](https://mega.nz/folder/PjwVhTxZ" \l "o0uRuSGPWUlMaHXC_mbi1w). אם האתר מבקש Decryption Key הכניסו: o0uRuSGPWUlMaHXC\_mbi1w

# הנחיות הרצה

- אם ברצונכם לפתוח את הפרויקט בתוך המנוע עליכם להשתמש בגרסה 2022.3.34f1 של המנוע. גרסה אחרת לא תעבוד בצורה טובה! (זו חובה של המנוע Unity, ואגב גם של המנוע המתחרה Unreal Engine). אחרי הורדת המנוע תצטרכו לעשות Git Clone לפרויקט ולפתוח את הפרויקט שנמצא בתיקיית Unity.

- צירפנו גם Build של המשחק ולכן לא צריך את המנוע בשביל להפעיל את המשחק בנפרד, אך כן צריך בשביל לפתוח את הפרויקט ולראות את תוכנו. מי שירצה להריץ את ה-Build, יעשה Git Clone, ויכנס לתיקייה:

Unity -> SnakesAndLadders -> Build

ואז יריץ את הקובץ SnakesAndLadders.exe

אם אתם מסתבכים בהרצה תפנו אליי ואסביר לכם מה לעשות.

# סיכום

השתמשנו במנוע המשחקים המוכר Unity בשביל לפתח גרסה מיוחדת של המשחק הקלאסי "סולמות ונחשים" ועליה יישמנו שכבה של טכנולוגיה חכמה בעזרת Alpha-Beta Pruning שתעזור לנו לפתור את המשחק באמצעות מחשב. הצלחנו בגרסה שפיתחנו להפוך את הנחשים ליריבים חכמים המהווים אתגר אסטרטגי לשחקן. ובנוסף, הצלחנו להפוך את המשחק לדינמי ולתמוך במספר כלשהו של נחשים, סולמות, גדלי לוחות, ועוד.

# מבוא

המשחק סולמות ונחשים הוא משחק קלאסי וידוע, המאתגר את השחקנים בתנועה על לוח בגודל קבוע והוא כולל אלמנטים של מזל ואסטרטגיה. המשחק מתנהל על לוח ריבועי, שבו השחקנים מתקדמים בתורם על פי קוביות, נתקלים בסולמות שמקנים להם קפיצות קדימה, ובנחשים שמורידים אותם אחורה. הגרסה המסורתית של המשחק היא פשוטה יחסית, אך יש לה פוטנציאל רחב יותר כאשר משולבות בה טכנולוגיות מתקדמות, ולכן שינינו בקצת את כללי המשחק. בשנים האחרונות, גישות מתקדמות בתחום הבינה המלאכותית (להלן, AI) אפשרו לפתח שיטות חדשות לשדרוג משחקים קלאסיים, על מנת להוסיף מרכיבים של אסטרטגיה ותחכום. אחת מהשיטות המתקדמות לשדרוג טכנולוגי מסוג זה במשחקים היא השימוש ב-Alpha-Beta Pruning, טכניקת אופטימיזציה המשמשת לשיפור התהליך של חיפוש והערכה של מצבים אפשריים במשחקים עם שני שחקנים. הטכניקה מספקת דרך יעילה לצמצם את מספר הצמתים שנבדקים במהלך החיפוש, בכך שהיא חוסמת את החיפושים מבלי לבדוק את כל האפשרויות, ומאפשרת לקבוע את מהלך המשחק האופטימלי בצורה מהירה יותר. בעבודה זו נבחן כיצד ליישם את Alpha-Beta Pruning בסולמות ונחשים.

# כללי המשחק

A screenshot of a video game

Description automatically generatedהשחקן יכול לבחור דרך התפריט הראשי כמה נחשים יהיו במשחק, כמה סולמות, גודל הלוח, מספר התורות המקסימלי, כל כמה תורות יהיה מותר לזוז באלכסון (או שאסור אלכסון), ומה העומק המקסימלי לאלגוריתם, ובאיזה מצב לשחק (שחקן מול מחשב או מחשב מול מחשב), או שהשחקן יכול להשתמש בערכי ברירת המחדל שאנו הגדרנו.

תרשים 1: התפריט הראשי של המשחק – בחירת את כל הקלטים, ואת מצב המשחק

## מהלך המשחק

לוח המשחק הוא מטריצה ריבועית בגודל NxN, כאשר N הוא קלט של השחקן. בתחילת המשחק נוצרים מספר כלשהו של סולמות ונחשים, לפי הקלט המבוקש, כאשר נדרש לפחות נחש אחד, ומספר כלשהו של סולמות. במצב ההתחלתי השחקן נוצר בנקודת ההתחלה של המפה (הפינה השמאלית התחתונה של הלוח, באינדקס אפס). נקודות ההתחלה והסיום של הנחשים והסולמות נבחרים באופן אקראי.

A screenshot of a game

Description automatically generated

תרשים 2: דוגמה של לוח בגודל 7x7 עם שלושה נחשים ושתי סולמות

מטרת השחקן במשחק היא להגיע לסוף המפה (הנקודה הימנית-עליונה, באינדקס האחרון, או 48 בדוגמה). מטרת הנחשים היא לעכב את השחקן בהגעה לסיום ככל הניתן, והם ינצחו אם יעבור מספר התורות המקסימלי של השחקן (להלן, Max Turns) שהוגדר בהתחלה לפי הקלט של השחקן בתפריט הראשי.

לשחקן מותר בכל תור לזוז קובייה אחת: למעלה, למטה, ימינה, שמאלה. או בקובייה אחת באלכסון בכל X תורים כאשר X הוא קלט השחקן בתחילת המשחק. יש גם את האפשרות לבטל לחלוטין תזוזה באלכסון (כאשר X הוא מספר הגדול מ-Max Turns). אם השחקן מגיע לתחתית של סולם כלשהו, הוא ישר מטפס למעלה. אם השחקן עולה על ראש של נחש, הוא ישר מועבר לזנבו, בין אם זה עוזר לו, ובין אם לא. לאחר כל תור של שחקן, יהיה תור של הנחש שראשו הכי קרוב לשחקן לפי מרחק מנהטן.

לכל נחש מותר בתורו לבצע אחד מהמהלכים הבאים:

א. לזוז ריבוע אחד לכל כיוון שיבחר, כולל אלכסונים (בהנחה ולא יצא מגבולות הלוח)

ב. לוותר על תורו ולא לזוז כלל

ג. אם ראש הנחש עומד על התחלת סולם, או שקרוב במרחק אחד מהתחלת סולם (כלומר מתחת, מעל, או מאחד הצדדים מהתחלת הסולם), הנחש יכול להזיז את הסולם למעלה או למטה משבצת אחת בדיוק (בהנחה שהסולם והנחש לא יוצאים מגבולות הלוח). אורך הסולם וכיוונו לא ישתנה, והתחלתו וסיומו יעלו/ירדו ביחד משבצת אחת בדיוק.

גודל הנחש וכיוונו לא יכולים להשתנות, אלא במקרי הזזת סולם בהם הנחש מתאמץ מאוד להזיז את הסולם.

A screenshot of a game

Description automatically generated

תרשים 3.1: המחשה להזזת סולם (לפני) - הנחש הורוד הולך להזיז את הסולם למטה

A screenshot of a game

Description automatically generated

תרשים 3.2: המחשה להזזת סולם (אחרי) - הנחש הורוד הזיז את הסולם למטה

מהכללים שקבענו נובע שהנחשים הם קבוצה, מטרתם זהה והיא לעכב את השחקן עד שנבזבז לו את כל התורות המותרים. ומכיוון שנתנו את כל האפשרויות הבאות כקלט של המשתמש: **גודל הלוח, מספר התורות המקסימלי, מספר התורות לתזוזה באלכסון, מספר הסולמות, מספר הנחשים, והעומק המקסימלי לאלגוריתם**, אזי המשחק מאוד דינמי ויש אינסוף אפשרויות להרכבת משחקים שונים!

בנוסף, יצרנו שני מצבי משחק שונים, באחד גם השחקן וגם הנחשים משוחקים על-ידי המחשב ואין לנו השפעה על מהלך המשחק (אנו רק צופים במשחק קורה), ובשני אנחנו משחקים בתור השחקן, בעזרת מקשי תזוזה יעודיים, מול הנחשים שאותם משחק המחשב.

A screenshot of a computer

Description automatically generated

תרשים 4.1: מקשי התזוזה בתור שבו השחקן לא יכול לזוז באלכסון

A screenshot of a phone

Description automatically generated

תרשים 4.2: מקשי התזוזה בתור שבו השחקן יכול לזוז באלכסון

יש לציין שהנחשים יניחו שאנו משחקים לפי Alpha-Beta Pruning באותה דרך (אותה הערכה, Utility וכולי) על מנת שנוכל לבנות להם עץ אסטרטגיה.

להגברת החוויה וההנאה מהמשחק, הוספנו Sound Track מתאים שניתן להנמיך או לבטל בעזרת ה-Slider שבתפריט הראשי (ראה תרשים 1 בצד שמאל למטה).

# פיתוח המשחק והאלגוריתמים

את המשחק פיתחנו באמצעות מנוע המשחקים Unity המתאים למשחקים קלי-גרפיקה ודו-מימדיים כמו שלנו. נעזרנו בכלי AI כמו ChatGPT, מדריכי יוטיוב, ותיעוד המנוע ללימוד בסיסי של המנוע על מנת שנוכל לעבוד איתו. בסך הכל הפרויקט כתוב על גבי שנים-עשר קבצי קוד, המכילים בתוכם כ-3,000 שורות קוד בשפת C#. את ה-Assets שאנו מרנדרים על המסך (לדוגמה, הנחשים והסולמות) לקחנו כתמונה מהאינטרנט ובאמצעות חישוביים מתמטיים, עיוותנו את התמונות לצרכינו, בעזרת שינוי גדלים וכיוונים, על מנת שיראו אותם לפי נקודת ההתחלה והסיום האקראית שיצאה להם.

הרכיב המרכזי במערכת הוא האלגוריתם Alpha-Beta Pruning וההערכה שלו, לקחנו את הגרסה הרגילה של האלגוריתם ושינינו אותה. קודם כל, יצרנו את בקר המשחק, המיוצג ע"י המחלקה GameController, שהוא האחראי לקבלת הקלטים מה-Main Menu ובעזרתם ליצור ולרנדר את הלוח על המסך, ביצוע התורות של הנחשים והשחקן, מה קורה בסיום משחק, ועוד. לצורך ריצת האלגוריתם ויצירת מצבים (נחוץ ל-Successors) יצרנו מחלקה בשם GameState המייצגת מצב כלשהו במשחק והיא מכילה את כל המידע הנדרש להרצת האלגוריתם, כמו מיקומי השחקן, הנחשים, והסולמות, התור של מי זה התור הנוכחי, גודל הלוח, וכולי. האלגוריתם ממומש בתוך מחלקה זו ולכן יש לו גישה לכל התכונות האלו. בעצם כשמגיע לדוגמה תור הנחש, בקר המשחק יריץ את האלגוריתם Alpha-Beta Pruning בעזרת ה-GameState הנוכחי, התוצאה תחזור לבקר והוא יבצע את הצעד הבא של הנחש לפי תוצאת האלגוריתם (האלגוריתם יחזיר את ה-GameState המייצג את התור הבא של הנחש). את האלגוריתם הגבלנו עד רמה D כלשהי (שניתנת לבחירת השחקן בתפריט הראשי) כי אי אפשר לחשב את כל העץ עד סופו בזמן סביר, ובמיוחד כשפתחתנו את גודל הלוח לקלט המשתמש (ניסינו!). ה-D המומלץ שמצאנו הוא 10. יחד עם זאת, אם מחשבכם לא מספיק טוב והוא לא עומד בזה, הקטינו את ה-D, ואם מחשבכם טוב ו-10 לא אתגר בשבילו (התורות מחושבים מהר) הגדילו את ה-D.

פונקציית ההערכה שבנינו בודקת את מרחק המנהטן מהשחקן למטרה (בהערכת תור שחקן) ומכפילה אותו במינוס אחד, ואת המרחק מהשחקן למטרה פחות המרחק מהשחקן לנחש הכי קרוב (בהערכת תור נחש). אם יש כמה מצבים עם הערכה שווה שינינו את האלגוריתם המקורי בכך שיבחר את המצב שימקסם את המרחק מהשחקן למטרה בתור של נחש, והמצב שימזער מרחק זה בתור של שחקן.

**הערה:** בכוונה הלכנו על מספרים שליליים (גם כאן וגם ב-Utility) כי רצינו שהשחקן יתעדף מרחק קצר יותר למטרה. לדוגמה: עדיף לשחקן להיות במרחק 2 מאשר 4 למטרה, וכמובן שמתקיים: **-2 > -4**

תרשים 5.1: שיטת ההערכה שבנינו

A screenshot of a computer program

Description automatically generated

תרשים 5.2: קוד מתודת ההערכה שכתבנו

פונקציית ה-Utility שבנינו מוגדרת עבור מצבים סופיים, והיא תחזיר אינסוף למפסיד, ומינוס אינסוף למנצח.

תרשים 6: שיטת ה-Utility שבנינו

# הקשיים והפתרונות

## פיתוח בעזרת המנוע Unity

זו הבעיה שאיתה התחלנו, לא באנו עם ידע קודם הקשור בפיתוח משחקים בעזרת המנוע Unity. היינו צריכים ללמוד מאפס בעזרת האינטרנט איך עושים במנוע הכל, וכך עשינו. בשביל ללמוד הכל השתמשנו בסרטוני יוטיוב, התיעוד של Unity, וכלי AI כמו ChatGPT. השקענו זמן חשוב על לימוד בסיסי של המנוע על מנת שנוכל לעבוד איתו לפיתוח המשחק. בחרנו את המנוע הזה ולא אחד מהמתחרים (למשל Unreal Engine שכן יש לנו ניסיון איתו) משום שהוא ידוע כמנוע המתאים לפיתוחים קלי-גרפיקה ומשחקים דו-מימדיים כמו במקרה זה.

## בניית המשחק כדינמי

כפי שראינו בקלטים של תרשים 1, המשחק דינמי מאוד ונותן למשתמש המון אפשרויות בחירה כמו: גודל הלוח, מספר הסולמות, מספר הנחשים, מספר התורות המותר, כל כמה תורות יהיה מותר לזוז באלכסון (או שלא יהיה מותר בכלל), העומק המקסימלי לאלגוריתם, ובנוסף ניתן לבחור בין שני סוגי מצבי משחק שבראשון המשתמש משחק את השחקן ובשני המחשב משחק את השחקן. בניית המשחק כדינמי התגלתה כתוספת מאוד מסובכת שהוסיפה המון קשיים, ודרשה המון מחשבה ופתרונות יצירתיים מההתחלה כמו איך לייצר ולרנדר את הלוח בכל אופציה שתינתן, איך לגרום לנחשים ולשחקן לקחת בחשבון אחד את השני ביחד עם הסולמות, איך לתמוך בכל מספר של נחשים וסולמות אשר יהיו, יצירת שני מצבי המשחק הנ"ל, איך להעביר את הקלטים בין הסצינות ועוד. את הבעיה הזו פתרנו באמצעות זמן רב שהשקענו **במחקר ופיתוח,** חיפשנו באינטרנט כל מיני דרכים לעשות את המשחק כה דינמי, ונעזרנו גם בכלי AI ובסוף הצלחנו! מצאנו כל מיני שיטות שעזרו לנו בבעיות, לדוגמה **PlayerPrefs** שעוזר לשמור הגדרות שחקן בין כמה Game Sessions, והשתמשנו בו לצרכינו כ-Local Storage להעברת מידע בין סצינות.

## בניית האלגוריתם וההערכות הנדרשות

קושי נוסף שנתקלנו בו זה איך להתאים את אלגוריתם ה-Alpha-Beta Pruning הרגיל לצרכינו, איזו פונקציית הערכה ו-Utility לבחור ואיך לגרום לזה לעבוד במצב של "אחד נגד קבוצה" – כלומר השחקן משחק כנגד כל הנחשים ביחד כקבוצה, ואיך לגרום לנחשים לקחת בחשבון אחד את השני, למען מטרתם המשותפת, לעכב את השחקן.

את הבעיה הזו פתרנו בהמון זמן והשקעה של **הבנת** האלגוריתם המקורי לצורך שינויו. אנו יכולים לומר שניסינו המון שיטות הערכה שונות, ממרחק מנהטן למטרה, לנחשים, לסולמות, שילוב שלהם כמו מרחק מינימלי בין כולם, מרחקים אוקלידיים, ועוד. שיטת ההערכה שבחרנו הניבה לנו את התוצאות הטובות ביותר ולכן בחרנו בה. בנוסף, לאחר המון ניסיונות הבנו שעלינו להפוך בצורה כלשהי את המספרים כי רצינו שהשחקן יתן עדיפות למרחק נמוך מגבוהה, ולכן הכפלנו אותם במינוס אחד. הוספנו גם לאלגוריתם המקורי שיטת לבחירת מצב כאשר שיטת ההערכה שלנו מניבה אותו ערך במקום לבחור בצורה אקראית, או את המצב שהגיע ראשון. שיטה זו תשתמש במרחק מנהטן ללא כל שינויים ותוספות על מנת לבחור את המצב הבא בתור, ובכך היא מעשירה את ההערכה הנוכחית. שיטה זו שיפרה מאוד את תוצאות האלגוריתם.



תרשים 7.1: שיטת בחירת המצב כאשר הערכתם שווה

A screen shot of a computer program

Description automatically generated

תרשים 7.2: קוד שיטת בחירת המצב כאשר ההערכה שווה

הצלחנו לגרום לסוכני קבוצת הנחשים "לעבוד ביחד" בכך שכאשר נוצר GameState כלשהו (ובפרט בזמן יצירת ה-Successors באלגוריתם) הבנאי של מחלקה זו מחשב את מרחק המנהטן מכל נחש לשחקן, ובוחר את הנחש עם המרחק המינילי לפעול, ולא בהכרח לפעול עם הנחש שהתור הנוכחי כרגע שלו. בכך, בכל פעם שמפעילים את האלגוריתם על נחש כלשהו הוא תמיד יקח בחשבון בצעדיו הבאים את הנחשים האחרים ואת מיקומם, והם ביחד יעשו את מאמצם החזק ביותר בשביל לעכב את השחקן ובכך לנצח.

## קוד הפרויקט

**קוד הפרויקט שלנו מכיל יותר מ-3,000 שורות על גבי שנים-עשר מחלקות** ולא ניתן לצרף אותו כאן, משום שזה יהפוך את הקובץ להיות באורך של לפחות 200 עמודים, יחד עם זאת, ניתן הסבר קצר למחלקות. צירפנו גם את קבצי הקוד בעבודה וגם ב-GitHub הכוללים תיעוד רחב.

יש לנו את 12 המחלקות הבאות:

**GameController** – זהו בקר המשחק למצב השחקן מול המחשב השולט על קבלת הקלטים מה-Main Menu ובעזרתם ליצור ולרנדר את הלוח על המסך, קריאה לאלגוריתם Alpha-Beta Pruning באמצעות המצב הנוכחי, וביצוע התורות של הנחשים והשחקן לפי תוצאת האלגוריתם או לחיצת השחקן על כפתורי התזוזה, הפעלת וכיבוי כפתורי התזוזה באלכסון לפי התור הנוכחי, מתי המשחק נגמר, מה קורה בסיום משחק, ועוד. קובץ זה מכיל כ-600 שורות.

A screenshot of a computer

Description automatically generated

תרשים 8: תרשים הרצף של המתודה SnakeTurn() המפעילה את האלגוריתם Alpha-Beta Pruning ומבצעת את הצעד הבא של הנחש הנוכחי לפי תוצאת האלגוריתם

**GameState** – מייצג מצב משחק כלשהו ב-Mode של שחקן מול מחשב. מכיל את כל המידע על המשחק כמו מיקומי השחקנים, הנחשים, והסולמות, גודל הלוח, התור הנוכחי, כל כמה זמן מותר לעשות תזוזה באלכסון ועוד. כשהאלגוריתם Alpha-Beta Pruning מייצר Successors של מצב מסוים הוא בעצם מייצר מופעים שונים של מחלקה זו. האלגוריתמים וההערכות שלנו ממומשים כאן והמחלקה מכילה כ-830 שורות.

**Ladder** – מכיל את המידע על אובייקט הסולם במצב המשחק של שחקן מול מחשב. מכיל מידע כמו היכן נמצא הסולם, האינדקס שלו, גודל הלוח, ובקר המשחק. מתודה עיקרית שלו לדוגמה היא UpdateLadderPosition שמקבלת מקומות התחלה וסיום חדשים לסולם ומרדנדרת אותו על המסך במקום החדש, מתודה זו משומשת כאשר נחש מזיז את הסולם.

**Snake** - מכיל את המידע על אובייקט הנחש במצב המשחק של שחקן מול מחשב. מכיל מידע כמו היכן נמצא הנחש, האינדקס שלו, גודל הלוח, ובקר המשחק. מתודה עיקרית שלו לדוגמה היא UpdateSnakePosition שמקבלת מקומות התחלה וסיום חדשים לנחש ומרדנדרת אותו על המסך במקום החדש, מתודה זו משומשת כאשר הנחש זז בתורו, בין אם הזיז סולם ובין אם לאו.

**ComGameController, ComGameState, ComLadder, ComSnake** – אלו מחלקות הדומות לארבע שהוצגו למעלה, רק למצב המשחק של מחשב מול מחשב. חלק מהקוד משוכפל במחלקות אלו **בכוונה.** כן ניתן לצמצם ולייעל את הקוד קצת, אבל זה היה דורש לימוד ארוך הרבה יותר (מהניסיון הרב שיש לנו עם Unreal Engine) של המנוע ומחלקותיו והורשה בו, מה שהיה מונע מאיתנו לפתח את האלגוריתמים שיותר חשובים, ולכן היה עדיף לשכפל קצת קוד.

**GameOverScreen** – מחלקה זו היא בשביל מסך הסיום של המשחק, בכל מצב משחק. כשהמשחק נגמר בקר המשחק יוצר אובייקט ממחלקה זו ומראה אותו על המסך, ביחד עם איזה טקסט להראות, לדוגמה “Victory” או “Defeat” לפי תוצאת המשחק.

**MainMenu** – זו המחלקה האחראית לתפריט הראשי, בה לוקחים את הקלט מהמשתמש ושומרים ב-PlayerPrefs ועוברים בין הסצינות לפי מצב המשחק שהמשתמש יבחר.

**VolumeSlider** – מחלקה זו היא לאלמנט ה-Slider שבתפריט הראשי, בעזרתה אנו שולטים על ה-Volume של המוזיקה, שומרים אותו ב-PlayerPrefs לפעם הבאה וטוענים משם את ה-Volume שנשמר בפעם האחרונה.

**MusicPlayer** – מחלקה זו אחראית על המוזיקה במשחק. **חייב** לשים אותה כמחלקה נפרדת ולא בתוך התפריט הראשי כי הצמדנו אותה לאובייקט שלא נהרס בין מעבר סצינות, בשביל שהמוזיקה לא תיפסק לעולם (אלא אם המשתמש כיבה אותה בעצמו בעזרת ה-Slider שבתפריט הראשי).