# מטלה שבועית - ארכיטקטורת פרוייקט

**ארכיטקטורת פרויקט: פיתוח משחק "Collect & Avoid" בסגנון ארקייד דו-ממדי**

## סקירה כללית

המטרה כאן איננה לבנות משחק עשיר בפיצ’רים, אלא **לתרגל ארכיטקטורת פרויקט נכונה**, במשחק זה השחקן שולט בדמות דו-ממדית שנעה על המסך ואוספת חפצים (Collectibles) תוך הימנעות מאויבים (Enemies). הארכיטקטורה אמורה להיות מורכבת ממודולים נפרדים, שימוש במחלקות ו-OOP עם הפרדה ברורה בין רכיבים שונים (למשל, מנהלי משחק, מחלקות עזר וכו').

## מאפיינים מרכזיים (Features)

1. **דמות שחקן (Player)**
   * יכולה לזוז בארבעה כיוונים (למעלה, למטה, שמאלה, ימינה) או עם מקשי WASD.
   * כוללת מערכת חיים (לדוגמה, 3 פסילות).
2. **חפצים לאיסוף (Collectibles)**
   * מופיעים במיקומים אקראיים על המסך (או קבועים מראש).
   * כל איסוף של חפץ מעלה את הניקוד (Score).
3. **אויבים (Enemies)**
   * נעים על המסך בתבניות תנועה פשוטות (Patrol) או באקראיות מסוימת.
   * פגיעה בשחקן מורידה חיים לשחקן.
4. **ציון וממשק משתמש (UI)**
   * תצוגת ניקוד (Score) שמתעדכן כשאוספים חפץ.
   * תצוגת חיים (Lives).
   * אפשר להוסיף תצוגת זמן (Timer) אם רוצים לשלב הגבלת זמן.
5. **תנאי ניצחון/הפסד**
   * תנאי ניצחון: השחקן אסף מספר מסוים של חפצים לפני תום הזמן או הגיע לניקוד יעד.
   * תנאי הפסד: לשחקן נגמרות החיים או תם הזמן (אם משחקים עם טיימר).

## דרישות לארכיטקטורת הפרויקט

1. **עיצוב מחלקות ועקרונות OOP**
   * **Player Class** : אחראית על תנועה, גילוי התנגשויות (Collision) עם חפצים ואויבים, וניהול החיים.
   * **Enemy Class** : אחראית על תנועת אויב (ניתן ליצור EnemyBase ולהוריש תנועות שונות במחלקות בנות)
   * **Collectible Class** : אחראית על טיפול בסוג החפץ, אפשרויות Spawn, ואינטראקציה עם השחקן.
   * **GameManager** : מנהל את הזרימה הראשית של המשחק – אחראי על בדיקת תנאי ניצחון/הפסד, יצירת חפצים, אפשרות לספירת זמן וכו'.
   * **UI Manager** : אחראי על עדכוני הממשק הגרפי (ציון, חיים, מסכי סיום).
2. **מחלקות עזר (Helper/Utility Scripts)**
   * **SpawnManager** : מטפל ביצירת חפצים ואויבים במיקומים אקראיים או קבועים.
   * **Timer**/**Countdown** : במידה ומשתמשים במגבלת זמן
   * **Constants**/**Setting** : לשמור משתנים גלובליים כגון מהירות תנועה של דמות השחקן, זמן Spawn של אויבים, ניקוד מקסימלי וכו'.
3. **ארגון תיקיות/ספריות**  
   דוגמה למבנה אפשרי (התאימו למנוע המשחק או לסביבת העבודה שבחרתם):



1. **שמירה על קוד נקי (Clean Code)**
   * כל מחלקה צריכה להיות בעלת אחריות ממוקדת (Single Responsibility).
   * שמות משתנים ופונקציות צריכים להיות תיאוריים וברורים.
   * הימנעו ממחלקות ענקיות
   * מומלץ לכתוב הערות קצרות (Comments) רק היכן שהדבר תורם להבנה, ולא באופן מוגזם.

## תהליך עבודה (Workflow)

1. **תכנון מבנה המחלקות**
   * צרו תרשים UML בסיסי או שרטוט על דף שמראה איך המחלקות מתקשרות זו עם זו.
   * החליטו איך GameManager יידע לעדכן את UIManager כאשר השחקן אוסף חפץ, וכו'.
2. **מימוש מכניקות בסיסיות**
   * צרו PlayerController שמאפשר תנועה וכרגע רק בודק התנגשויות פשוטות עם הסביבה.
   * צרו מחלקה בסיסית לאויב (EnemyBase) עם לוגיקת תנועה מינימלית.
   * צרו CollectibleItem שניתן לאסוף ושהוא מעלה ניקוד.
3. **שילוב מנגנוני ניהול (Managers)**
   * מימוש GameManager שיטפל בלוגיקה המרכזית של התקדמות המשחק (הפעלת ספאון של חפצים/אויבים, בדיקה אם נגמרו החיים וכו').
   * UIManager שמקבל מידע על הניקוד והחיים הנוכחיים ומציג על המסך.
4. **הוספת "פוליש" (Refinements)**
   * ודאו שהתנאי לניצחון/הפסד פועל: אם הזמן נגמר, או אם אספתם מספיק חפצים וכו'.
   * הוסיפו מסך סיום (Game Over) או מסך ניצחון (Win Screen).
5. **Refactor - שיפור מבנה הקוד**
   * אם יש קוד שחוזר על עצמו (כמו אלגוריתם תנועה לאויבים שונים), שקלו להשתמש במחלקת אב משותפת או ממשק (Interface).
   * הקפידו שכל סקריפט נשאר ממוקד יחסית ואינו מתנפח ליותר מדי פונקציות ותכונות.

## תוצרים (Deliverables)

1. **ריפוזיטורי מסודר ב GitHub**
2. **README**  קצר שמסביר:
   * איך מריצים את המשחק.
   * תיאור קצר של יחסי המחלקות (Class Relationships).
   * הנחות שבוצעו או בחירות מיוחדות בארכיטקטורה.
3. **תרשים מחלקות** (UML) או תרשים בסיסי אחר, שמציג את המחלקות והקשרים החשובים ביניהן.
4. **בניה והעלאת המשחק** לItch.io בפורמט WebGL עם פירוט הוראות איך לשחק ולינקים לריפוזיטורי מItch ומהItch לריפוזיטורי.