תכנון מפורט לפרויקט: [שם הפרויקט]

מאת: [שם מלא ות.ז שלך]

1. הקדמה: פסקה קצרה המתארת את הפרויקט

[שם הפרויקט] הוא משחק פלטפורמה דו-ממדי מסוג מטרוידבניה (Metroidvania) המפותח במנוע המשחק Unity במסגרת פרויקט גמר בלימודי מדעי המחשב. המשחק מתמקד בחקירת עולם פתוח-למחצה, לחימה באויבים (EnemyController.cs), ורכישת יכולות חדשות המאפשרות לשחקן (MainCharacter.cs) לגשת לאזורים שקודם לכן היו חסומים. מכניקת הליבה הייחודית היא מערכת חליפות (Suit System): חליפות שונות, המוגדרות כ-Scriptable Objects (Suit.cs), מעניקות לשחקן יכולות תנועה והתקפה מיוחדות (המוגדרות בSuitAbility.cs.

השחקן יכול להחזיק רק חליפה פעילה אחת בכל זמן נתון. כאשר לשחקן יש חליפה פעילה, הוא יכול לבצע פעולה אקטיבית של "המרת חליפה לחיים" באמצעות לחיצה על כפתור ייעודי. פעולה זו משמידה את החליפה הנוכחית בתמורה לכמות מסוימת של חיים (דרך קריאה לפונקציית הריפוי ומחזירה את השחקן למצב הבסיסי. החליפות הן מנגנון ההתקדמות המרכזי במשחק, אך גם משאב טקטי לריפוי. המשחק שם דגש על שליטה מדויקת בדמות, עיצוב שלבים מורכב המעודד חקירה, ומערכת לחימה מבוססת יכולות-חליפה ומתמקד בחווית משחק רציפה בתוך סשן (session) בודד.

2. ארכיטקטורה

פילוסופיית העיצוב:  
הארכיטקטורה של המשחק מבוססת על תכנות מונחה רכיבים (Component-Based Architecture), בהתאם לפרדיגמה של מנוע Unity. הלוגיקה מחולקת לרכיבים קטנים וניתנים לשימוש חוזר המוצמדים לאובייקטים במשחק (GameObjects). גישה זו מקדמת מודולריות וגמישות.

מבנה המערכת:  
המערכת מחולקת למספר תת-מערכות (Subsystems) לוגיות עיקריות, המקיימות אינטראקציה ביניהן:

1. תת-מערכת השחקן (Player Subsystem): אחראית על כל ההיבטים של הדמות הראשית – קליטת קלט (כולל הקלט הייעודי להמרת חליפה), תנועה, לחימה, ניהול החליפה הפעילה והפעלת יכולותיה, וביצוע ההמרה של החליפה לחיים כאשר הפקודה מתקבלת ויש חליפה פעילה.
2. תת-מערכת האויבים (Enemy Subsystem): מנהלת את כל האויבים במשחק – בינה מלאכותית (AI), תנועה, התקפות, תגובה לנזק, והפלת חליפות.
3. מערכת החליפות (Suit System - Data & Logic): מגדירה את הנתונים והיכולות של כל חליפה (באמצעות Scriptable Objects) ומכילה את הלוגיקה לאיסוף חליפות (הממומשת בתוך תת-מערכת השחקן).
4. ניהול מצב המשחק (Game State Management): אחראית על המצב הכללי של המשחק (תפריט, משחק פעיל, סיום) והמעברים ביניהם.
5. תת-מערכת ממשק המשתמש (UI Subsystem): מציגה מידע לשחקן (חיים, חליפה, תפריטים) ומגיבה לאינטראקציות משתמש.
6. מערכת קלט (Input System): מתווכת בין הקלט הפיזי (מקלדת/בקר) לבין פקודות משחק סמנטיות, כולל הפקודה הספציפית להמרת חליפה.
7. מערכת אירועים (Event System): מאפשרת תקשורת מבוזרת (Decoupled Communication) בין תת-המערכות השונות.
8. מערכות ליבה של המנוע (Core Engine Systems): המערכות המובנות של Unity (פיזיקה, אנימציה, רינדור, שמע) שעליהן המשחק נבנה.

תרשים מודולים קונספטואלי:

תמונה שמכילה טקסט, תרשים, קו, תוכנית

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

תקשורת בין מודולים:

* תקשורת פנימית (Intra-Subsystem): בתוך כל תת-מערכת, רכיבים מתקשרים לרוב באמצעות הפניות ישירות.
* תקשורת חיצונית (Inter-Subsystem): כדי למנוע תלויות הדוקות, נעשה שימוש נרחב במערכת אירועים מבוססת Scriptable Objects (Beacons/Channels).

3. פירוט המודולים / תת-מערכות

כל המודולים רצים בצד לקוח. אין רכיבי צד שרת.

סוגי משתמשים והרשאות:

* משתמש רגיל (Player):
  + ממשק: חלון המשחק, תפריטים (ראשי, פאוזה, הגדרות), אינטראקציה דרך מקלדת/בקר.
  + הרשאות: לשחק במשחק לפי הכללים, לשנות הגדרות, להתחיל סשן משחק חדש בכל הפעלה. ההתקדמות אינה נשמרת בין הפעלות.
* מנהל מערכת (Developer/Tester):
  + ממשק: סביבת הפיתוח של Unity Editor, כלי Debug מובנים (קונסול, תפריטי Debug).
  + הרשאות: גישה מלאה לכל רכיבי המשחק לצורך פיתוח ובדיקות.

תיאור תת-המערכות:

* תת-מערכת השחקן (Player Subsystem):
  + אחריות: מייצגת את השחקן. אחראית על תרגום קלט לפעולות, ניהול מצב השחקן, וניהול החליפה הפעילה. היא מחזיקה את ה-Suit.cs הפעיל, מיישמת את יכולותיו, מטפלת באיסוף חליפה (וריפוי), ומגיבה לקלט הייעודי של "המרת חליפה לחיים": כאשר הקלט מתקבל, היא בודקת אם יש חליפה פעילה, ואם כן, מפעילה את פונקציית הריפוי.
  + מימוש לדוגמה: MainCharacter.cs מרכז את הלוגיקה, משתמש ב-CharacterMovement.cs, CharacterCombat.cs ויורש מ Character.cs לריפוי/נזק.
* תת-מערכת האויבים (Enemy Subsystem):
  + אחריות: מנהלת אויבים, AI (מכונת מצבים), ניווט (NavMesh), והפלת ייצוגי חליפות.
  + מימוש לדוגמה: EnemyController.cs, EnemyStateMachine.cs, EnemyStateSO.cs ונגזרותיו.
* מערכת החליפות (Suit System - Data & Logic):
  + אחריות: מגדירה את הנתונים של החליפות (יכולות, ערכי ריפוי). הלוגיקה של השימוש בחליפות נמצאת בתת-מערכת השחקן.
  + מימוש לדוגמה: Suit.cs (SO), SuitAbility.cs ומימושיו.
* ניהול מצב המשחק (Game State Management):
  + אחריות: ניהול המצב הכללי (תפריט, משחק, סיום).
  + מימוש לדוגמה: GSManager.cs ו-GameStateSO.cs ונגזרותיו.
* תת-מערכת ממשק המשתמש (UI Subsystem):
  + אחריות: הצגת מידע (חיים, חליפה, תפריטים) ותגובה לאינטראקציות UI.
  + מימוש לדוגמה: UIManager.cs
* מערכת קלט (Input System):
  + אחריות: תרגום קלט פיזי לפקודות משחק, כולל הפקודה הייעודית להמרת חליפה.
  + מימוש לדוגמה: Unity Input System, InputReader.cs.
* מערכת אירועים (Event System):
  + אחריות: תקשורת מבוזרת בין תת-מערכות.
  + מימוש לדוגמה: BeaconSO.cs, GameStateChannelSO.cs, UIChannelSO.cs.
* מערכות ליבה של המנוע (Core Engine Systems):
  + אחריות: פיזיקה, אנימציה, רינדור, שמע.
  + מימוש: שימוש ברכיבי Unity.

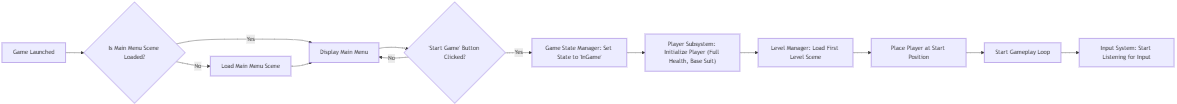
4. בסיס נתונים

סעיף זה אינו רלוונטי לפרויקט זה. לא ממומשת מערכת שמירה וטעינה, ולכן אין בסיס נתונים או מבנה נתונים קבוע לאחסון בין הפעלות. מצב המשחק קיים רק בזיכרון במהלך הסשן הפעיל ומתאפס בסיומו.

5. תרשים זרימה של פעולות המערכת/תאור תרחישים

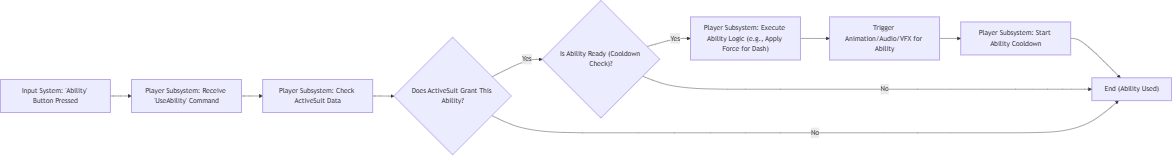
תרחיש 1: תחילת סשן משחק

* תיאור: השחקן מפעיל את המשחק, מגיע לתפריט הראשי ובוחר להתחיל משחק חדש. המערכת מאתחלת את מצב המשחק וטוענת את השלב הראשון.

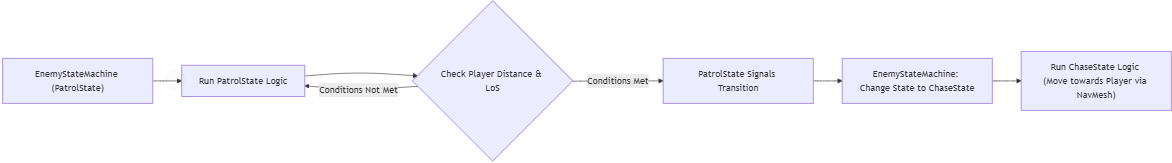


תרחיש 2: שימוש ביכולת תנועה מיוחדת (למשל, Dash)

* תיאור: לשחקן יש חליפה המעניקה יכולת Dash. השחקן לוחץ על הכפתור המתאים. המערכת בודקת אם היכולת זמינה ומפעילה את לוגיקת ה-Dash, כולל אפקטים וצלילים.



תרחיש 3: אויב מזהה את השחקן ועובר ממצב "סיור" למצב "מרדף"



6. פרוטוקולי התקשורת שבהם בוחרים

* תקשורת פנימית (Intra-Subsystem):
  + קריאות ישירות לפונקציות / גישה לרכיבים.
* תקשורת חיצונית (Inter-Subsystem):
  + מערכת אירועים מבוססת Scriptable Objects (Beacons/Channels).
* תקשורת עם מערכת ההפעלה:
  + File I/O: אינו בשימוש.

7. שונות / אלגוריתמים / דפוסי עיצוב

* Component-Based Design: הגישה המרכזית של Unity, המאפשרת בניית ישויות מורכבות מהרכבת רכיבים קטנים.
* State Machine Pattern: שימוש נרחב לניהול התנהגות מורכבת של אויבים ולניהול מצב המשחק הכללי. מאפשר הפרדה ברורה של לוגיקה עבור כל מצב.
* Scriptable Objects: שימוש אינטנסיבי ומרכזי עבור:
  + תצורה ונתונים: הגדרת חליפות, יכולות, סוגי אויבים, מצבי משחק – מאפשר שינוי נתונים מבלי לשנות קוד.
  + מערכת אירועים: הגדרת ערוצי התקשורת (Channels) והאירועים עצמם.
* Event-Driven Architecture: שימוש במערכת האירועים (Channels) כדי להניע חלקים מהלוגיקה ולהפחית צימוד בין תת-מערכות.
* Object Pooling (אופציונלי): אם יש יצירה והשמדה תכופה של אובייקטים (כמו קליעים או אפקטים), ניתן לשקול שימוש ב-Object Pooling לשיפור ביצועים. (לא צוין, אך רלוונטי לשקול).
* Pathfinding Algorithm: שימוש באלגוריתם (כמו A\*) הממומש על ידי מערכת ה-NavMesh שנבחרה, לצורך ניווט אויבים.
* עיצוב שלבים (Level Design Patterns): מבוסס על Suit Gating, תוך התחשבות במכניקת המרת החליפה לריפוי כמכניקה טקטית.