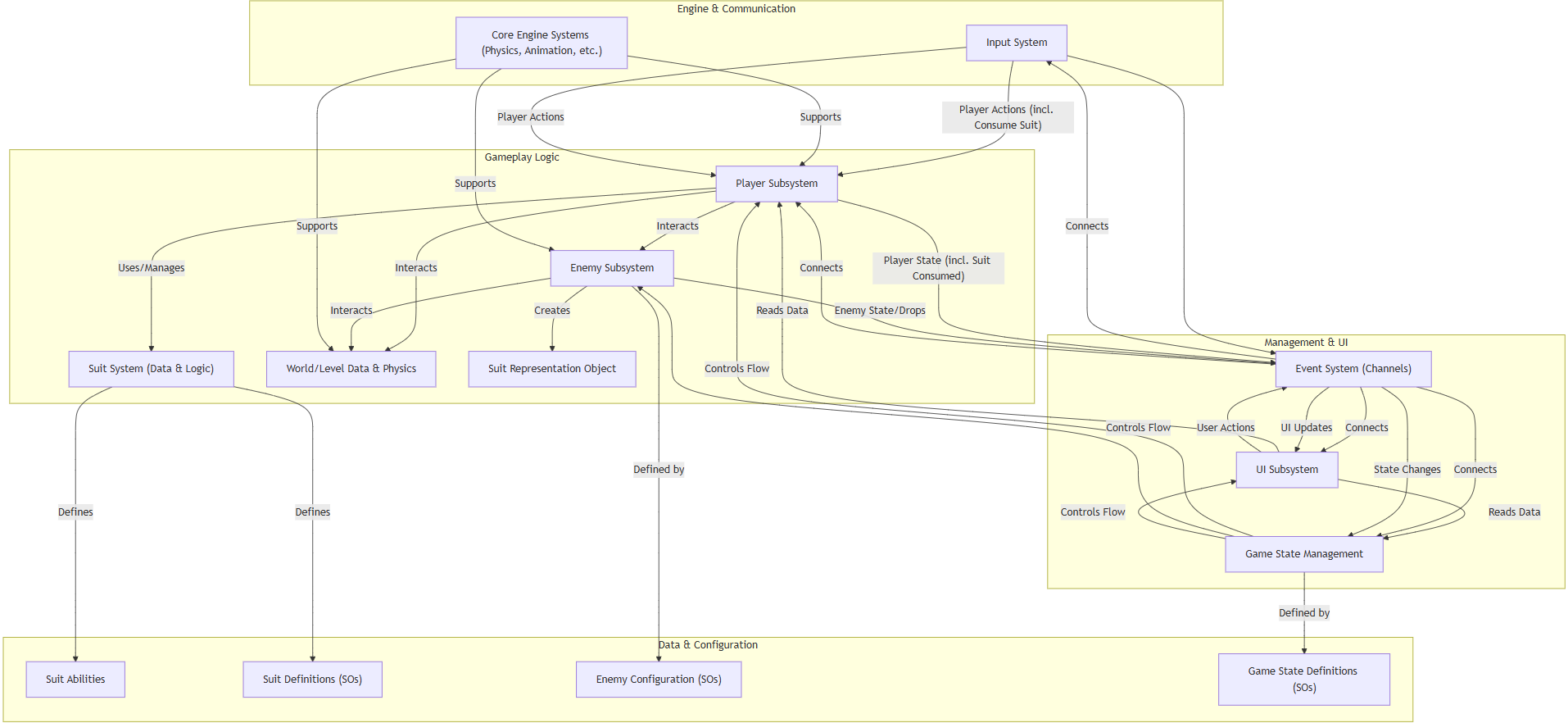
תכנון מפורט לפרויקט: Eye&Shadow

1. **הקדמה**Eye&Shadow הוא משחק פלטפורמה דו-ממדי מסוג מטרוידבניה (Metroidvania) המפותח במנוע המשחק Unity. המשחק מתמקד בחקירת עולם פתוח-למחצה, לחימה באויבים ורכישת יכולות חדשות החיוניות להתקדמות. מכניקת הליבה הייחודית היא מערכת חליפות: חליפות שונות, המוגדרות כ"תבניות נתונים" באמצעות טכנולוגיה של Unity הנקראת Scriptable Objects (SO). ניתן לחשוב SO כעל מאגר מידע גמיש המשמש להגדרת תכונות ותצורות של אלמנטים במשחק, בדומה לרשומה במסד נתונים או מסמך NoSQL. כל חליפה כזו מגדירה את היכולות המיוחדות, המראה של השחקן, וערכי הריפוי שהיא מעניקה.

****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 משחק (session) בודד.

1. **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 (Game Objects) גישה זו מקדמת מודולריות וגמישות.

תקשורת בין מודולים:

תקשורת פנימית (בתוך תת-מערכת): רכיבים קרובים שעובדים יחד (למשל, השליטה, התנועה והלחימה של השחקן) מדברים ישירות אחד עם השני באמצעות הפניות ישירות. זה מהיר וישיר, אך יוצר קשר הדוק.

תקשורת חיצונית (בין תת-מערכות): כדי שתת-מערכות שונות (כמו השחקן, ה-UI ומנהל המשחק) יוכלו להודיע זו לזו על אירועים חשובים (כמו "השחקן מת" או "המשחק הושהה") מבלי להכיר אחת את השנייה ישירות, אנו משתמשים במערכת אירועים.

* 1. ערוצים (Channels): לכל סוג אירוע מרכזי יש "ערוץ" משלו (מוגדר כScriptable Object).
  2. משדרים (Senders): כשמשהו חשוב קורה (למשל, השחקן מת), הרכיב שאחראי על כך "משדר" הודעה בערוץ המתאים (לדוגמא "ערוץ מצב השחקן"). הוא פשוט מודיע על האירוע מבלי לדעת מי מקשיב.
  3. מאזינים (Listeners): רכיבים אחרים שמתעניינים באירועים מסוג זה (למשל, ה-UI שצריך להציג "Game Over", או מנהל המשחק שצריך לעצור את המשחק) "מאזינים" לערוץ הזה. כשהם שומעים את ההודעה, הם מבצעים את הפעולה הנדרשת.

1. **בסיס נתונים**

המשחק אינו כולל מערכת שמירה וטעינה של התקדמות השחקן בין הפעלות, ולכן אין בסיס נתונים במובן המסורתי לאחסון מצב משחק דינמי. מצב המשחק קיים רק בזיכרון במהלך סשן פעיל.

עם זאת, סביבת הפיתוח של Unity עצמה מתפקדת כמאגר נתונים ("בסיס נתונים") עבור הנכסים (Assets) והתצורות הסטטיות של המשחק. זה כולל:

* נכסים ויזואליים ושמיעתיים: קבצי תמונות, שמע, אנימציות.
* תצורות עולם: סצנות (Scenes) המגדירות מבנה שלבים.
* תבניות אובייקטים: פריפאבים (Prefabs) המאפשרים שימוש חוזר באובייקטים מוגדרים מראש (כמו אויבים, פריטים, פלטפורמות)
* נתוני משחק ותצורה (Scriptable Objects): משמשים באופן נרחב לאחסון נתונים הניתנים לעריכה נוחה, כגון:
  + הגדרות של חליפות ויכולותיהן (Suit.cs, SuitAbility.cs).
  + הגדרות התנהגות אויבים ומצבי משחק של State Machine
  + ערוצי תקשורת למערכת האירועים

Unity מנהל את הנתונים האלה באמצעות קבצי מטא-דאטה (.meta) השומרים על זיהוי ייחודי וקישורים בין הנכסים, ומנגנוני סריאליזציה פנימיים לשמירת ההגדרות בקבצים. למעשה, הפרויקט הוא בסיס הנתונים של עצמו עבור כל הנתונים שאינם משתנים באופן דינמי תוך כדי משחק.

1. **פרוט המודולים**

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

תת-מערכת השחקן: אחראית על כל ההיבטים של הדמות הראשית – קליטת קלט, תנועה, לחימה, ניהול החליפה הפעילה והפעלת יכולותיה. יורשת מהמודל הבסיסי של דמות מתקשרת עם מנוע הפזיקיה והאנימציה של המשחק, מערכת הקלט, עם מערכת חליפות ותתי מודלים לתזוזה ופעולות קרב.

תת-מערכת האויבים: מנהלת את כל האויבים במשחק אחראית על בינה מלאכותי(AI) , תנועה, התקפות, תגובה לנזק, והפלת חליפות יורשת מהמודל הבסיסי של דמות מתקשרת עם מנוע הפזיקיה ,האנימציה והשכבות של המשחק, מכונת מצבים של האויב ועם מערכת החליפות.

מערכת החליפות: מגדירה את הנתונים והיכולות של כל חליפה באמצעות (Scriptable objects) מכילה את הלוגיקה לאיסוף חליפות (הממומשת בתוך תת-מערכת השחקן) מתקשרת עם מודל הדמות הראשית האויב ותת מודל הקרב של הדמות הראשית.

ניהול מצב המשחק: אחראית על המצב הכללי של המשחק (תפריט, משחק פעיל, סיום וכו'..) והמעברים ביניהם מייסם תבנית עיצוב State,מתקשר עם ממשק המשתמש ובמצבי המשחק המוגדרים כScriptable objects .

מערכת ניהול שלבים: אחראית על מעבר בין שלבים, מיקום השחקן וטעינת האובייקטים המתאימים לסצנה החדשה.

תת-מערכת ממשק המשתמש: אחראית על הצגת מצב המשחק תפרטים ומידע שנדרש לשחקן ומגיבה לאינטראקציות משתמש. בעזרת מערכת הקלט ואירועים במשחק. מתקשרת עם מערכת ניהולל מצבי המשחק, רכיבי ממשק משתמש של המנוע והדמות הראשית.

מערכת קלט: מתווכת בין הקלט הפיזי (מקלדת/בקר) לבין פקודות משחק סמנטיות מתקשרת עם השחקן, מערכת האירועים ומערכת מצבי המשחק .

מערכת אירועים: מאפשרת תקשורת מבוזרת בין תת-המערכות השונות.

מערכות ליבה של המנוע: המערכות המובנות של Unity (פיזיקה, אנימציה, רינדור, שמע) שעליהן

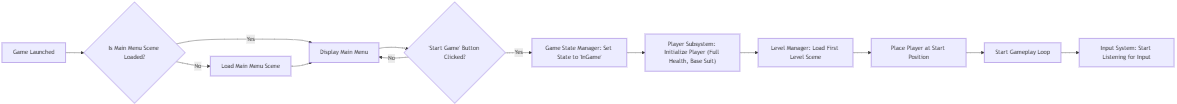
המשחק נבנה. נמצא בתקשורת עם הדמות הראשית, האוייבים, מערכת החליפות ומערכת ממשק המשתמש.

## הערות:

* זהו אחד החלקים החשובים ביותר בקובץ הזה, הרחיבו עוד על כל מערכת, מה המטרה שלה? כיצד היא פועלת? עם מי היא מתקשרת? האם המודול מממש תבנית עיצוב?

1. תמונה שמכילה טקסט, צילום מסך, גופן, קו

   תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**תמונה שמכילה טקסט, קו, תרשים, עלילה

   תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.תרשים זרימה של פעולות המערכת (תרחישים לדוגמא)**תרחיש 1: תחילת סשן משחק  
   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"

## הערות:

* גם כאן, התרשימים קטנים מידיי

1. 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.

שימוש מרכזי עבור Scriptable Objects:

* 1. תצורה ונתונים: הגדרת חליפות, יכולות, סוגי אויבים, מצבי משחק – מאפשר שינוי נתונים מבלי לשנות קוד.
  2. מערכת אירועים: הגדרת ערוצי התקשורת (Channels) והאירועים עצמם.

Event Driven Architecture: שימוש במערכת האירועים (Channels) כדי להניע חלקים מהלוגיקה ולהפחית צימוד בין תת-מערכות.

Pathfinding Algorithm: שימוש באלגוריתם (כמו A\*) הממומש על ידי מערכת ה-NavMesh שנבחרה, לצורך ניווט אויבים.