מטלה – דגמים וטריגרים

א. שינוי ושיפור המשחק מהשיעור

הורידו מגיטהאב את הקוד של משחק-החלליות שבנינו בהרצאה
(זה כאן https://github.com/gamedev-at-ariel פרויקט מספר 02).

שימו לב: אין להוריד בעזרת zip אלא בעזרת git clone. כמו כן כדי לקצר את זמן ההורדה מומלץ להגביל את העומק:

```
git clone --depth=1 https://github.com/<repository-name>.git
```

ודאו שאתם מבינים את הקוד. לאחר מכן, בצעו לפחות שינוי אחד מתוך הרשימה הבאה:

1. לשחקן יש כמה נשקים שונים עם מאפיינים שונים (לדוגמה, לייזר בגודל שונה, מהירות שונה, כמה יריות בכל לחיצה, וכו'). לכל נשק יש צירוף מקשים (combo) שמפעיל אותו. לדוגמה, הצירוף abcab מפעיל לייזר בגודל כפול, וכד'. כתבו מערכת כללית, שתאפשר לשנות את הצירוף מתוך הinspector.

2. הניקוד של השחקן לא מוצג מעל החללית, אלא במקום קבוע על המסך, למשל בפינה הימנית-עליונה. יש לבדוק שהניקוד מוצג במקום הנכון גם כשגודל המסך משתנה, כשהמסך מסתובב וכו'. *רמז*: השתמשו ב Canvas.

3. שנו את גבולות המשחק: הפכו את העולם לעולם עגול בציר האופקי (כשהשחקן מגיע לצד ימין של העולם הוא מופיע בצד שמאל ולהיפך), ועולם סגור בציר האנכי (השחקן לא יכול לצאת מגבולות המסך למטה או למעלה). השתמשו ברכיבי-התנגשות ולא במספרי-קסם.

4. החללית של השחקן לא נהרסת מייד כשהוא מתנגש באויב, אלא יש לו בתחילת המשחק 3 "נקודות בריאות" (health points), כל פגיעה באויב מורידה לו נקודה אחת, ורק כשהוא מגיע לאפס הוא נהרס. מדי פעם מופיעות על המסך "נקודות בריאות" שהשחקן יכול לאסוף ולהוסיף לעצמו. – מצאו דרך יפה להציג לשחקן כמה נקודות בריאות יש לו.

5. השחקן לא יכול לירות לייזרים בלי הפסקה, אלא חייב לחכות זמן מסוים (נניח חצי שניה) בין ירייה לירייה הבאה. בנוסף, יש לו תחמושת מוגבלת, וכשהתחמושת נגמרת, הוא לא יכול יותר לירות וחייב לעבור לשלב הבא. בתחילת השלב התחמושת מתחדשת.

6. יש שני סוגי אויבים: אחד איטי וכשפוגעים בו מקבלים נקודה אחת, והשני מהיר וכשפוגעים בו מקבלים שתי נקודות (כיתבו קוד כללי, שיהיה קל להרחבה והתאמה לכל מספר של סוגי אויבים).

7. המגן לא נמצא על המסך בהתחלה, אלא מופיע מדי-פעם בנקודה אקראית. כשהשחקן מתנגש במגן, נוסף עיגול מסביב לחללית של השחקן. צבע העיגול נחלש משניה לשנייה עד שהוא נעלם אחרי 5 שניות.

8. מדי-פעם מופיע על המסך תותח, בנקודה אקראית. כשהשחקן אוסף את התותח, הוא יכול לירות לייזר גדול וחזק יותר, למשך מספר שניות. התותח הוא חד-פעמי כמו המגן – נעלם אחרי שהשחקן אוסף אותו [אם אתם מסתבכים תוכלו למצוא הסבר בגיטהאב של הקורס. אבל תנסו קודם לבד].

בנוסף לשינוי אחד מהרשימה, הוסיפו לפחות שינוי אחד מקורי.

ב. מימוש תהליכי-ליבה

כזכור, **תהליך-הליבה** הוא אוסף הפעולות המתבצעות שוב ושוב במהלך המשחק. הנושאים שלמדנו בשיעור הזה מאפשרים לכם כבר לממש תהליכי-ליבה של הרבה משחקים קלאסיים. ממשו את תהליך-הליבה במשחק כלשהו לבחירתכם. **רצוי לבחור משחק מקורי.** אם שריר-המקוריות שלכם עייף, בחרו אחד מהמשחקים הבאים:

1. Pac Man – צריך לעבור במבוך ולאכול את כל הנקודות בלי להתנגש ברוחות רפאים. יש פירות שמוסיפים נקודות בונוס, ויש נקודות מיוחדות שנותנות חסינות למשך מספר שניות.
2. כורה הזהב – Gold miner: אתם שולטים בגמד עם כלי-חפירה; אתם צריכים לכוון את כלי-החפירה שלכם לכיוון המתאים כדי שיאסוף קוביות זהב גדולות ככל האפשר.
3. נינג'ה פירות – Fruit Ninja: פירות נופלים מהשמיים; השחקן מחזיק חרב וצריך להזיז אותה בכיוון הנכון כדי לחתוך את הפירות.
4. דו קרב: בנו משחק דו קרב יריות בין שני עצמי קובייה. השחקנים יזוזו משני קלטים שונים במקלדת (למשל קובייה אחת תזוז מהחיצים והשנייה מהמקשים a,s,d,w) הקוביות צריכות לעמוד אחת מול השנייה כמו כן שימו לב שהכיוון של הלייזר של כל קובייה צריך להתאים למיקום של הדמות שיורה אותו. המנצח הוא מי שהצליח ראשון לפגוע שלוש פעמים בקובייה היריבה.
5. Tetris racing - השחקן הוא מכונית המורכבת מקוביות. השחקן זז קדימה; מהצד השני מגיעות מכוניות שגם הן מורכבות מקוביות. השחקן צריך להיזהר לא להתנגש בהן. שימו לב שהשחקן זז רק על ציר ה-x.
   אם תרצו לשפר את המשחק הגדירו את התנועה של המכוניות האחרות מציר ה-z במקום מציר ה-y, והמצלמה קצת מעל לשחקן, כך המשחק ירגיש ייחודי.
6. Guitar Hero - קוביות בצבעים שונים יורדות באקראי מהחלק העליון של המסך. צבע של הקוביות נבחר בצורה אקראית מבין שלושה צבעים, וכאשר הקובייה מתנגשת באובייקט אחר שנמצא בתחתית המסך על השחקן ללחוץ על המקש המתאים בהתאם לצבע הקובייה. למשל מקשים z=אדום, x=כחול, ו-c =צהוב. לשינוי צבע אובייקט בזמן משחק מומלץ להיעזר בזה.

אין צורך לממש את כל המשחק (עם כמה רמות, חיים, ניקוד, גרפיקה וכו'), אלא רק את תהליך-הליבה.

דרישות לארכיטקטורת הפרויקט

יש לבנות את פרויקט המשחק מתוכנן ומסודר. כדוגמה, ניתן הנחיות לכתיבת פרויקט מסודר עבור משחק פק-מן. אם בחרתם משחק אחר, יש לבצע את ההתאמות הדרושות למחלקות.

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):

```
Assets/
├── Scripts/
│   ├── Managers/
│   │   ├── GameManager.cs
│   │   └── SpawnManager.cs
│   ├── Player/
│   │   └── PlayerController.cs
│   ├── Enemies/
│   │   └── EnemyBase.cs
│   ├── Collectibles/
│   │   └── CollectibleItem.cs
│   ├── UI/
│   │   └── UIManager.cs
│   └── Utils/
│       ├── Timer.cs
│       └── Constants.cs
├── Prefabs/
├── Scenes/
└── UI/
```

4. 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.

הגשה

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.

ההגשה של כל חלקי המטלה אישית. זאת כדי שכולכם תדעו לתכנת ותוכלו לתרום באופן שווה לעבודת הצוות בהמשך הקורס.