בס"ד

# חלליות בשני ממדים - סיום

## טעינה מחדש של סצנות-

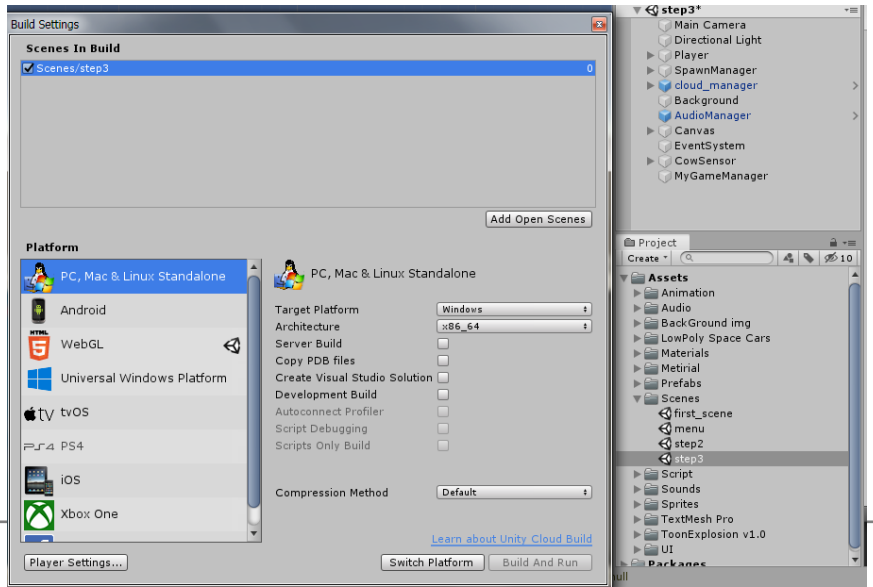
עד עכשיו המשחק שלנו אמור לרוץ כמו שצריך, הבעיה שהוא עדיין משחק חד פעמי- אחרי שהגענו לסוף המשחק אי אפשר לשחק בו שוב, אלא אם נצא ממנו ונתחיל אותו מחדש. אם נתייחס לסצנה כאל שלב במשחק, מה שבעצם נרצה שיקרה זה שנוכל להטעין מחדש את הסצנה למסך כך שכל ה"שלב" שהרגע שחקנו יתחיל מחדש אם הפסדנו.

ראשית נדאג שעם סוף המשחק יופיע לנו טקסט שמודיע לנו שכדי להתחיל מחדש יש להקיש על אחד המקשים במקלדת, כבר ראינו מספיק פעמים כיצד לעשות את זה- נצטרך להוסיף אובייקט טקסט ל-UI Manager, ולמתודה שמפעילה את הטקסט GAME OVER שתפעיל גם את הכיתוב הנ"ל. עכשיו בשביל להטעין את הסצנה נצטרך איזשהו אובייקט ריק שינהל לנו את המעבר בין הסצנות.

ניצור אובייקט כזה עם השם My Game Manager או משהו בסגנון, נוסיף לו סקריפט עם שם זהה וניכנס אליו.
אנחנו צריכים דבר ראשון איזשהו משתנה בוליאני עם ערך false שיהווה אינדיקטור שהמשחק נגמר, בהמשך נשנה את ערך המשתנה כאשר המשחק באמת נגמר ב- UI Manager. עכשיו במתודה Update() נצטרך לבדוק שני דברים: 1) האם לחצנו על הכפתור או על המקש שאמור להתחיל את הסצנה מההתחלה. 2) האם למשתנה הבוליאני שמסמן את סוף המשחק יש ערך true, ואם אנחנו מקיימים את שני התנאים נרצה שתטען הסצנה מחדש.
למזלנו ל-unity יש אובייקט מיוחד שאחראי לטעינת assets מהמשחק- Scene Manager וע"י המתודה LoadScene(int SceneName) נוכל לטעון סצנות . לשם כך נצטרך להשתמש במרחב שם ייחודי לכך :

using UnityEngine.SceneManagement.

המתודה שציינו קודם מקבלת כפרמטר מספר שלם שמסמן את המספר של הסצנה, כדי לבדוק איזו סצנה יש לנו נצטרך להוסיף את הסצנה שלנו ל-Build Setting. כדי להגיע ל- Build Setting נבחר File-> Build Setting , ולבחור "Add open scene" אמורות להופיע לנו כל הסצנות שיצרנו עד עכשיו במשחק, נוכל לבחור מבניהן מה אנחנו רוצים שיבנו ואילו לא, וליד כל סצנה מופיע מצד ימין המספר שמייצג אותה, כמו כן ניתן פשוט לגרור את הסצנה מחלון הפרויקט לחלון ה-build setting וגם כאן יופיע מצד שמאל בקטן המספר שמייצג את הסצנה:



אופציה נוספת היא להשתמש בשם הסצנה דרך הscene manager : `SceneManager.GetActiveScene().buildIndex` זה ייתן לנו את המספר סצנה שאנחנו עכשיו נמצאים בה.
מבחינת סינטקס במתודה ()update של MyGameManager נרצה שזה יראה כך:

```
[SerializeField]
    private bool _GameOver = false;
.   .   .
public void SetGameOver()
    {
        _GameOver = true;
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R) && _GameOver)
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
        }
    }
```

במקרה שלנו הסצנה מוגדרת להיות המספר 0, אבל היא יכולה להשתנות בהתאם למספר הסצנות שאנחנו בונים למשחק שלנו. נשאר לנו עדיין להתאים בין ה- UI Manager לבין My Game Manager כדי שיעדכן אותו להפעיל את המתודה SetGameOver() כאשר המשחק נגמר. בעיקרון ניתן לשמור אובייקט עצם מסוג MyGameManager אבל בנתיים זה דיי מיותר לכן נסתפק בלמצוא אותו ואת המתודה עם `FindObjectOfType<MyGameManager>().SetGameOver();` .
אותה נבצע ב-UI Manager איפה שמפעילים את הכיתוב Game Over .
נריץ את המשחק ונראה שכאשר באמת נפסלנו ואנחנו לוחצים על אותו כפתור הוא מאפס לנו את כל הערכים שהיו לנו לפני כן, זה משום שהטענה מחדש של הסצנה גם מאתחל את כל הערכים של האובייקטים של אותה הסצנה, יותר נכון לומר שכל האובייקטים שקשורים לסצנה נמחקים מהזיכרון ומאותחלים מחדש. בהמשך נראה כיצד ניתן לשמור על אובייקטים מסוימים במעבר בין הסצנות.

## תבנית סינגלטון - שמירה על נתונים במעבר בין סצנות

כפי שכבר נוכחנו לראות- כאשר אנחנו מטעינים את הסצנה מחדש כל המידע אודות האובייקטי משחק של הסצנה מתאפס.
כדי שנוכל לשמר מידע בין הסצנות נצטרך איזשהו אובייקט ריק שיספוג לתוכו את כל המידע הרצוי על כל אובייקט שנרצה לשמור על נתוניו לסצנה הבאה. כמין מסד נתונים קטן שילווה את כל המשחק.
אנחנו צריכים שהאובייקט:
1) יהיה נגיש לכל סקריפט במשחק.
2)מאותחל רק פעם אחת
3)ישמור מידע של כמה אובייקטים.
בחירה לוגית תיהיה לממש אותו בתבנית עיצוב סינגלטון כדי שלא יוכלו ליצור עוד אינסטנסים שלו במהלך המשחק (רק אובייקט אחד שימש אותנו לאגור את data).

דבר ראשון שנעשה- ניצור אובייקט ריק חדש ונשנה את שמו למשהו שמתאים לתפקיד שלו, בסגנון של "Global Object" .
נצמיד לאובייקט סקריפט עם שם זהה ונפתח אותו.
לפני שנמשיך, תזכורת לתבנית סינגלטון- בסינגלטון אנחנו יוצרים במחלקה משתנה עצם סטטי מסוג המחלקה ובודקים האם הוא כבר מאותחל, כלומר האם הוא null . במידה והוא לא מאותחל נאתחל אותו להיות this כלומר להיות המחלקה, אחרת, אם הוא כבר מאותחל, נשמיד את אותו אינסטנס. זה יבטיח לנו שיהיה רק אובייקט אחד כזה לאורך כל התוכנית(במקרה שלנו המשחק). בנוסף, כדי להגדיר את האובייקט כאחד שנשמר בין הסצנות נצטרך להשתמש בפונקציה: DontDestroyOnLoad(...) שכשמה כן היא- מקבלת כפרמטר איזשהו אובייקט ומגדירה אותו ככזה שנשמר בין סצנות.

היות ואנחנו רוצים שהאובייקט יאותחל לפני כולם (כי הוא מעדכן את שאר האובייקטים) נשתמש במתודה ()awake במקום ב-()start כדי לאתחל אותו. ומעתה כאשר נרצה להשתמש באותו "Global Object", נשתמש באותו משתנה סטטי של המחלקה ששאר האובייקטים יקבלו ממנו מידע ויתעדכנו ממנו (במתודה או בדרך אחרת) כל תחילת סצנה או סוף סצנה.
מבחינת סינטקס זה יראה כך:

```
public class GlobalObject : MonoBehaviour
{
    public static GlobalObject Instance;

    void Awake()
    {
        if (Instance == null)
        {
            DontDestroyOnLoad(gameObject);
            Instance = this;
        }
        else if (Instance != this)
        {
            Destroy(gameObject);
        }
    }
}
```

שאלה תכנותית: אנחנו שכבר מכירים דבר או שניים בתכנות בטח חושבים למה לא ניתן פשוט להשתמש באובייקט סטטי או משתנים סטטיים של מחלקה, הרי אובייקטים סטטיים משותפים לכל האינסטנסים של אותה סוג מחלקה.
בניגוד למה שאנחנו עלולים לחשוב אינטואיטיבית, אובייקטים סטטיים לא "מתמידים" לאורך כל המשחק. היות וכל סקריפט (כל מחלקה) מחובר לאיזשהו אובייקט משחק, היא תיהרס ביחד עם האובייקט כאשר אנחנו נטעין סצנה חדשה. אפילו אם בסצנה החדשה אנחנו מייצרים את אותו אובייקט עם שדות ומשתנים סטטיים, משום שהרסנו את האובייקט הקודם כאילו לא נוצר אף פעם אובייקט כזה (כאילו זאת תוכנית חדשה).
לכן ,כדי לשמר נתונים בין סצנות, אנחנו חייבים להשתמש בתבנית סינגלטון ובמתודה (...)DontDestroyOnLoad.

עכשיו כשכבר ראינו כיצד לדאוג שאובייקט ישמר במעבר בין הסצנות, נראה דוגמא על ה-AudioManager .
אנחנו רוצים שבכל פעם שנאתחל את הסצנה המוזיקה שלנו לא תתאפס אלא תמשיך מאותו מקום.
לשם כך נצטרך להגדיר אותו כ-סינגלטון להצמיד לו את מתודה (...)DontDestroyOnLoad כמו שכבר ראינו:

```
public class AudioManager : MonoBehaviour
{
    public Sound[] sounds;

    void Awake()
    {
        public static AudioManager Instance;

            void Awake()
            {
                if (Instance == null)
                {
                    DontDestroyOnLoad(gameObject);
                    Instance = this;
                }
                else
                {
                    Destroy(gameObject);
                    return; // So it won't do unnecessary commands
                }
            }

        foreach(Sound s in sounds)
        {
            s.source = gameObject.AddComponent<AudioSource>();
            s.source.clip = s.clip;
```

```
            s.source.volume = s.volume;
            s.source.pitch = s.pitch;
            s.source.loop = s.loop;
        }
    }
    . . .
}
```

## תוספת (אראל):

דרך חלופית לממש תבנית סינגלטון היא לכתוב רכיב כללי שאפשר להצמיד לכל אובייקט שנותנים לו "תג" (tag) ייחודי. הרכיב בודק אם כבר יש אובייקט עם אותו תג; אם כן, הוא משמיד את עצמו. אם לא, הוא מכריז על עצמו כאובייקט היחיד עם תג זה, שלא יושמד כשיטענו סצינות חדשות. בשיטה זו אין צורך במשתנים סטטיים (הם נחשבים בעייתיים).

```
public class Singleton : MonoBehaviour {
    void Awake() {
        string myTag = gameObject.tag;
        GameObject[] otherObjectsWithSameTag = GameObject.FindGameObjectsWithTag(myTag);
        if (otherObjectsWithSameTag.Length > 1) {
            Destroy(gameObject);
        } else {
            DontDestroyOnLoad(gameObject);
        }
    }
}
```

# -Main Menu

כשמדברים על תפריט ראשי מתכוונים לחלון הראשון או הסצנה הראשונה שאנחנו רואים כאשר אנחנו מפעילים את המשחק.
שלב ראשון ניצור סצנה חדשה במשחק, נקרא לה Menu או משהו בסגנון וניכנס אליה (כמובן לשמור לפני את שאר הסצנות).
כדי ליצור סצנה חדשה נעמוד על חלון הפרויקט (על התיקייה של הסצנות )-> מקש ימני-> Scene<-Create.
כדי להוסיף רקע לסצנה נוכל לגרור תמונת רקע ואז נצטרך להתאים אותה למסך, או שנוכל להוסיף אובייקט פאנל למסך ולגרור אליו את התמונה. בחלון ההיררכיה נלחץ מקש ימני -UI-> Panel, ונגרור ל- Source Image של הפאנל את התמונה הרצויה (יש לוודא שהתמונה היא מסוג (Sprite(2D and UI . בשביל לשפר את הנראות אפשר לשנות את הצבע שלה.
אם עדיין לא התאמנו את הקנבס כדי שיתאים את עצמו לגודל המסך זה הזמן לעשות את זה, **לתזכורת**.

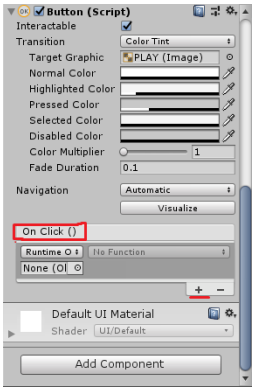
בתפריט נרצה שיהיו לנו כפתורים עם טקסט, וכפי שכבר נוכחנו לראות האובייקט טקסט של הקנבס לא נותן לנו יותר מידי אפשרויות למניפולציות, לשמחתנו החל מ-Unity2016 יש asset חדש שמאפשר משחק בטקסטים כמו צביעה הצללה ושיפור איכות אוטומטית- Text Mesh Pro. כדי ליצור אובייקט text-mesh נבחר מקש ימני מעל הקנבס-> UI-> Text Mesh Pro.
Unity אמור להוריד לנו את ה-asset אוטומטית. אם לא - אפשר להוריד אותו מה-Asset Store של unity בחינם.
אחרי שכבר יש לנו text-mesh נתאים את גודל הטקסט למסך ונשנה את שמו לפונקציה אותה הוא אמור למלא, למשל לאופציה שמאתחלת את המשחק נקראה "PLAY" (נהוג לכתוב את התפריט באותיות גדולות), ליציאה-"QUIT" וכו'.
בינתיים הטקסט עדיין צבוע לבן, אם נרצה להוסיף מעט הצללה נוכל ללכת לרכיב- underlay "נאפשר אותו" (enable) ונשנה את כפתורי ההזזה כאוות נפשנו.
להוספת צבעים נלך לרכיב-color Gradient ונבחר את ארבעת הצבעים שיראו לנו הכי מתאימים לטקסט.
אם נרצה לשמור על הגוונים שעשינו לטקסט שישמשו אותנו בהמשך לעוד אובייקטים כאלה נוכל ליצור asset של Color Gradient ולגרור אותו לכל אובייקט text-mesh חדש שניצור (ב-Gradient(Preset) ). בשביל ליצור asset כזה נלחץ החלון הפרויקט מקש ימני-> Create -> Text Mesh Pro<-Color Gradient.

כמובן שה-text-mesh אמור להיכנס לאיזשהו כפתור. ניצור כפתור חדש לקנבס, מקש ימני על הקנבס->UI<-Button .
בשביל להתאים את הכפתור נשנה את הצבע שלו לשחור(או צבע אחר שיתאים למסך) ולא נאפשר את הרכיב image שלו.

נשים לב שלאובייקט כפתור יש אובייקט בן- טקסט, ברמת העיקרון אנחנו רוצים להחליף בין האובייקט טקסט הזה לבין ה--text mesh שייצרנו. נמחק את האובייקט טקסט ונגרור לכפתור את האובייקט שיצרנו. כדי להקל עלינו נשנה את שם הכפתור למילה של שה-text-mesh מייצג ואת שם ה-text-mesh נשנה לטקסט. נשנה את המיקום של הטקסט ביחס לכפתור באייקון ה- Rect Transform של ה- text-mesh, נלחץ Alt ונבחר באייקון הימני למטה, זה ידאג שהטקסט יותאם לכפתור.
אם נריץ את הסצנה נראה שבאמת ניתן ללחוץ על האובייקט אך שום דבר לא קורה, אפילו אינדיקציה שלחצנו אין.
נחזור לכפתור ונאפשר שוב את הרכיב image. נשנה ברכיב button את המשתנים הבאים:
ב- Normal Color נשנה את הצבע אלפא (האות A בחלון שנפתח, אמור להיות האופציה האחרונה) ל-0, כלומר לצבע שחור.
ה-normal color הוא המשתנה שמייצג את הצבע רקע של הכפתור השיגרה.
ב- Highlighted Color נשנה את האלפא גם צבע כהה אך לא לגמרי שחור. ה- Highlighted מייצג את הכפתור כאשר עומדים עליו אבל לא לוחצים עליו.
ב- Pressed Color נשנה את הגוון של האלפא להיות כהה, אך אפילו יותר בהיר מהצבע של ה-Highlighted. ה-Pressed, כשמו כן הוא , מייצג את האובייקט כשלוחצים עליו.
אם נריץ נראה שוב נראה שהתוצאה די יפה –כשלא לוחצים על הכפתור נראה שאין רקע לטקסט; כשעוברים מעל הכפתור אך לא לוחצים נראה שיש רקע כהה מסביב הטקסט, אך עדיין רקע שקוף; וכשלוחצים על הכפתור יש רקע אפילו יותר כהה מסביב.
נשכפל את הכפתור ונשים את הכפתור השני מתחת לראשון. נניח הכפתור הראשון שעשינו הוא PLAY, אז נשנה את השם של הכפתור השני ל-QUIT למשל, ונשנה גם את הטקסט שלו בהתאם. בדוגמא הקרובה אנחנו נציג רק את שני הכפתורים האלו(PLAY ו- QUIT).
לפני שניתן פונקציונאליות לכפתורים, כדאי שנשים אותם תחת אובייקט ריק אחד מכמה סיבות:
*אנחנו רוצים את האופציה להזיז אותם יחד, מבלי להצטרך לגרור אותם אחד אחרי השני.
*יהיה לנו יותר קל אם יהיה לנו סקריפט אחד לשני הכפתורים, כפי שנראה בהמשך.
*נוכל לשכפל את האובייקט הנ"ל וכך, אם נצטרך עוד סוגים שונים של תפריטים, פשוט להעתיק את אותו אובייקט ולשנות אותו בהתאם לתפריט החדש (נהפוך אותו ל-prefab ונדרוס אותו).
לכן ניצור אובייקט ריק חדש בתוך הקנבס, נקרא לו בשם מתאים למשל Game Menu ונגרור לתוכו את הכפתורים.
נצמיד סקריפט לאובייקט עם שם זהה וניכנס אליו.
ברמת העיקרון האובייקט Game Menu הוא אובייקט זמני במשחק שלא מתעדכן כל פריים, וכנראה לא ישמור בתוכו אובייקטים, אפוא ניתן למחוק לו את שתי המתודות הדיפולטיביות שמגיעות איתו: Start() ו- Update().
אנחנו רוצים שתהיה מתודה מיוחדת לכל כפתור באובייקט: לכפתור PLAY אנחנו צריכים שתהיה מתודה שמטעינה את הסצנה הבאה (כלומר תחילת המשחק ממש). לפני שנכנס לקוד נחזור ל-unity וניכנס build setting, הסצנה הראשונה במשחק אמורה להיות הסצנה של הMenu דווקא ולא הסצנה שעבדנו עליה עד עכשיו, לכן פשוט נמחק את הסצנות מה-build setting ונגרור את הסצנה של התפריט הראשי קודם ולאחר מכן את הסצנה של המשחק.
נחזור לקוד. תזכורת: בכדי להשתמש בטעינת סצנות צריכים להשתמש במרחב שם מיוחד : using UnityEngine.SceneManagement, נבנה מתודה מיוחד שטוענת את הסצנה של תחילת המשחק:

```
public void PlayGame()
    {
        SceneManager.LoadScene(1);
    }
```

ונחזור ל- unity שוב. אם נסתכל באינספקטור של הכפתור PLAY נראה שם את הרכיב on click() , הרכיב אחראי למאורע שלחצנו על הכפתור, הוא יכול להוסיף פונקציונליות לכפתור. נלחץ על הפלוס מצד ימין למטה ונוסיף עוד פונקציה לכפתור:



עם הוספת הפונקציה קפצו לנו שלושה מלבנים: runtime, no function ו- None .
המלבן none מסמן האם יש אובייקט שאפשר להשתמש באחת(או יותר) המתודות שלו כאשר לוחצים על הכפתור. נגרור לשם את ה-Game Menu שלנו, כי אנחנו הולכים להשתמש במתודות שלו.
במקום ה- no function נבחר -> Game Menu (שנוסף לאחר שגררנו אותו פנימה) -> PlayGame() .
אם נריץ את המשחק נראה שאכן מתי שלוחצים על PLAY נטענת הסצנה הבאה כמתוכנן.
נחזור שוב לסקריפט של ה-Game Menu, ונוסיף מתודה ליציאה מן המשחק.
ניתן להשתמש בפונקציה: Application.Quit() אבל שימו לב שהיא לא תעבוד לנו כל זמן שאנחנו מריצים את המשחק דרך unity, אלא רק לאחר שנבנה את המשחק ממש כפי שנראה בהמשך.
מבחינת סינטקס זה אמור להיראות כך:

```
public void QuitGame()
    {
        Application.Quit();
    }
```

אותו דבר שעשינו עם הכפתור PLAY, נעשה גם עם הכפתור QUIT, רק שנטעין את המתודה QuitGame() במקום.
אם הפכנו את ה Audio Manager לאובייקט don't destroy on load נוכל להכניס אותו לתפריט, כך נשמע את מנגינת הרקע עוד מהתפריט והיא תשמר לנו גם כשנתחיל את המשחק.
לעוד מידע על האופציות שאפשר להוסיף לmain menu:

https://www.youtube.com/watch?v=YOaYQrN1oYQ

# –Post processing

בחלק זה נעסוק בשימוש בpost processing.
Post processing מאפשר לנו לשנות את האיכות\ הצבעים של התמונות המשחק, להוסיף פילטרים וכדו'. כמין Photoshop למשחק. בשביל להשתמש ב post processing נצטרך להתקין אותו תחילה. ניכנס לwindows-> package management ונחפש בשורת החיפוש post processing .
לאחר שהוספנו אותו לפרויקט שלנו, נצטרך להוסיף שכבה חדשה למצלמה שעליה נלביש את כל הפוסט פרוססינג: נלך לאינספקטור של אובייקט המצלמה -> Add Component->Post –processor Layer.נצטרך לקבוע שכבה חדשה שעליה יעבוד הפוסט פרוסור, עדיפות על שכבה גבוהה יחסית. נלך מעל לאינספקטור , היכן שכתוב layer, ונוסיף שכבה חדשה. נקרא לה בשם post processing, ונגדיר את השכבה של ה-post processor layer להיות אותה שכבה שכרגע הגדרנו.
ניצור אובייקט ריק חדש ונקרא לו post processor או משהו בסגנון. נגדיר את השכבה שלו באינספקטור להיות ה- post processing. נוסיף לו את הרכיב Post processing Volume ונגדיר אותו כ- Is Global,הגדרנו אותו כגלובלי כדי שההשפעה של האובייקט תהיה על כלל התמונות במשחק ולא על אובייקט ספציפי.
לאחר שהגדרנו את הרכיב הפוסט פרוסור של המצלמה והאובייקט הפוסט פרוסור שלנו להיות באותה שכבה נוכל להוסיף אפקטים חזותיים. בהתחלה נצטרך ליצור פרופיל חדש לאובייקט הפוסט פרוסור. נבחר ב- new היכן שכתוב profile באינספקטור של האובייקט. וכדי להוסיף אפקטים נבחר add effect.
הדרך היעילה ביותר לעבוד עם הפוסט פרוסור היא ע"י ניסוי וטעייה, נסו את האפקטים השונים ונוכחו לדעת מה הכי מדבר אליכם. להלן סקירה כללית:
*Anti-aliasing: מביא לגרפיקה מראה חלק יותר. טוב כאשר הקווים נראים משוננים או בעלי מראה מדורג( staircase appearance) זה קורה בדר"כ כאשר למרנדר הגרפי אין רזולוציה גבוהה מספיק כדי ליצור קן ישר.
*Bloom: יוצר שוליים או התרחבות של האור. מגביל או מוסיף אורות לתמונות. תורם לתאורה הכללית ולבהירות המצלמה.
*Chromatic Aberration: מחקה את אפקט המצלמה בעולם האמתי-כשהמצלמה שלה לא מצליחה לצרף את כל הצבעים

לאותה נקודה. התוצאה- "שוליים" של צבעים לאורך גבולות התמונה שמפרידים חלקים כהים ובהירים שלה. נותן תחושה של שכירות לתמונה.

***color grading**: משנה או מתקן את הזוהר שunity מספק. דומה ליישום filter באינסטגרם.

ל-color grading יש שלשה מצבים:

low definition range - אידיאלי לפלטפורמות עם איכות ירודה.

High definition range- אידיאלי לפלטפורמות התומכות HDR rendering .

External- מאפשר לספק טקסטורות מתוכנות חיצוניות.

***Deferred fog**: מתאים למשחקי תלת-ממד שיש בהם עומק. מייצר כמין אפקט של ערפל- נותן לאובייקטים צבע אפרפר יותר בהתאם למרחק שלהם מהמצלמה.

***Depth of field**: אפקט לאחר-עיבוד שמדמה את הפוקוס של עדשת המצלמה על אובייקט מסוים.

* **Auto Exposure**: מדמה איך העניים האנושיות מסתגלות לרמות שונות של חושך.

***Grain**: מחקה את האפקט של מצלמות בעולם האמתי מייצרות כאשר חלקיקים קטנטנים במסך גורמים לחספוס התמונה. בדר"כ משתמשים באפקט במשחקי אימה, שמנסים שהתמונה לא תהיה "מושלמת".

* **Motion blur**: מטשטשים אובייקטים מסוימים שנעים מהר יותר מהזמן חשיפה של המצלמה.

* **Screen space reflection**: מיצר השתקפויות עדינות שמדמות משטח רטוב או שלולית.

* **Vignette**- משאיר רק את מרכז התמונה מואר ומחשיך את הפינות.