בס"ד

משחקי קלפים

רקע-

משחקי קלפים הם כל אותם המשחקים שבהם הקלפים משמשים ככלי המרכזי במשחק(למשל מונופול לא נחשב משחק קלפים כי הקלפים הם רק תוסף למשחק ולא החלק האינטגרלי בו).
משחקי קלפים רבים קיימים אך רק מעטים מהם שמשוחקים עם חפיסה תקנית בעלי חוקים פורמליים.
מרבית משחקי הקלפים מנצלים את העובדה שניתן לזהות קלפים רק מצד אחד, כך שרק השחקן שמחזיק את הקלפים יכול לדעת מה יש לו, אבל לא מה יש לאחרים. זאת אחת הסיבות לכך שבד"כ משחקי קלפים מאופיינים עם מזל.
ישנם כמה סוגים של משחקי קלפים:

- *Trick taking games*- בהם השחקן צריך לצבור כמה שיותר קלפים (או כמה שפחות קלפים רעים). למשל המשחק ברידג'.
- *משחקי התאמות*- לנסות להשיג כמה שיותר סטים של קלפים מאותו הסוג, כל סט מקדם אותך לניצחון. דוגמא למשחק: רמי
- *Shedding games*- משחקים בהם המטרה להיפטר מכמה שיותר קלפים ביד. דוגמא למשחקים: ספיד, רמי-קוב.
- *משחקי השוואה*- המטרה להגיע למצב שמספר הקלפים ביד של השחקן זהה למספר מוגדר מראש למשל: 21(בלאק-ג'ק).
- *משחקי מזל*- משחקי קזינו שבהם המנצח זוכה בכסף (או משהו יקר ערך אחר), למשל פוקר.
- *משחקי סוליטר(או סבלנות)*- משחקי המשוחקים בדר"כ ע"י אדם אחד. השחקן מתחיל ממבנה אחד ומטרתו היא להיפטר מהמבנה או לבנות להיפטר מהחפיסה בידו ע"י הצבה של קלפים במבנה.

כמובן שכל אחת מהקטגוריות יכולה להיות משולבת עם קטגוריה אחרת.

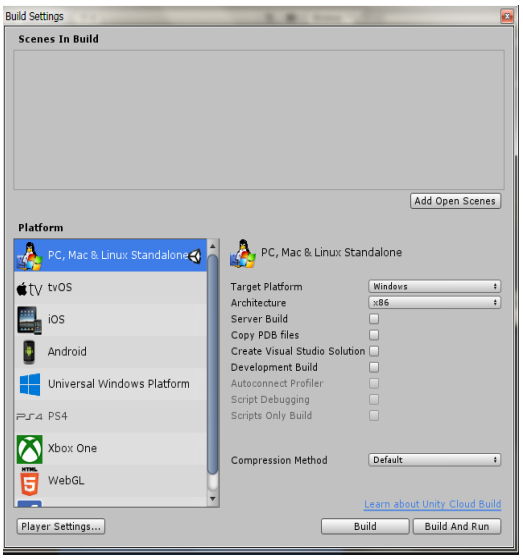
בפרק נבנה משחק סוליטר.
בסוף פרק זה יהיה לנו, לא רק משחק קלפים לטלפון, אלא גם אחלה תבנית למשחקי קלפים אחרים שנרצה לבנות בעתיד.
נראה כמה טכניקות חדשות, הכוללות: שימוש בקונפיגורציות XML ועיצוב משחקים לטלפונים סלולריים.

Set up-

במהלך השיעור נשתמש ב-asset שזמין באופן ציבורי: *Vectorized Playing Cards* של Chris Aguliar.
את ה-asset ניתן להוריד כאן, או באתר http://book.prototools.net/?page_id=519 למטה, היכן שכתוב Starter Package מתחת מופיע ה-asset : "C32_Prospector_Starter.unitypackage".
ניצור פרויקט חדש ב-unity (אין חשיבות אם 2D או 3D). ניתן לפרויקט שם, נייבא את ה-asset שהורדנו לפרויקט ע"י גרירה.
בשיעור נשתמש בסצנה שקיבלנו עם ה-asset שהורדנו: _Prospector_Scene_0 .
ניכנס לאובייקט המצלמה, ונוודא כי באינספקטור שלה היא מוגדרת כך: Position(0,0,-4),Rotation(0,0,0),Scale(1,1,1)
projection: Orthographic ו-size:10, ונוודא שיחס המסך הוא 4:3 (מעל חלון המשחק).

-Build Setting

את המשחק נבנה לטלפון, אך כמובן שניתן לבנות אותו לאיזו פלטפורמה שתעדיפו ובלבד שתתאימו אותו לאותה הפלטפורמה.
תזכורת: אם נרצה לשנות את פלטפורמת המשחק שלנו:1. נלחץ פעמיים על הסצנה (_Prospector_Scene_0) כדי לפתוח אותה.
2.מהתפריט הראשי נבחר *file*-> *build setting* שיפתח לנו את החלון הבא:



3. גררו את הסצנה לחלון scene In Build, או שלחצו על *Add Open Scene* להוספת הסצנה (אני מזכיר שהסצנה שנעבוד עליה היא _Prospector_Scene_0).
4. בחרו בפלטפורמה המתאימה (אנחנו נעבוד עם אנדרואיד ,אבל שוב למי שיש מכשיר של apple מוזמן לשנות לפלטפורמה שמתאימה לו), ואח"כ *Switch Platform*.
Unity מייבא כעת את כל התמונות כדי להתאים אותם לברירת המחדל של הפלטפורמה שבחרנו. ונסגור את החלון.
(אין צורך ללחוץ *Build* עדיין, אנחנו נעשה את זה רק בסוף בניית המשחק).

ייבוא תמונות כ-Sprites:

נצטרך לייבא את התמונות כראוי כדי להתאימן לשימוש כ-Sprites. תזכורת: Sprite היא תמונת דו ממד שניתן לבצע עליה מניפולציות כמו הגדלה, הזזה, סיבוב וכדו'.

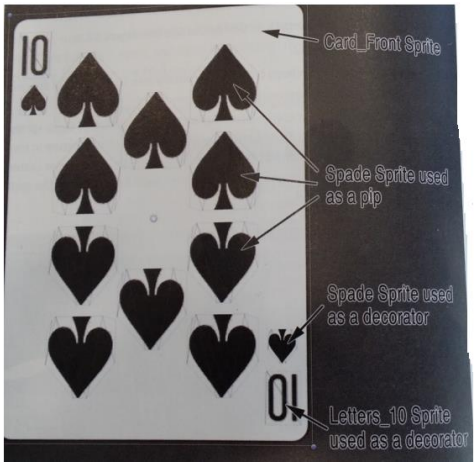
1. נפתח את תיקיית *_Sprties* בחלון הפרויקט, ונבחר את כל התמונות שבו- או ע"י בחירה באחד ואז בליחצה על ctrl + חץ למטה נסמן את כל התמונות עד שנגיע לתחתית התיקייה, או שנבחר את התמונה הראשונה ואז shift + לחיצה על התמונה האחרונה בתיקייה.

2. ניכנס לאינספקטור (אם סימנו את כל התמונות האינספקטור משפיע על כולן). ונשנה את ה-*Texture type* שלהן ל-*Sprite(2D and UI)*. השינוי אמור להשפיע על כלל התמונות שסימנו.
אם נסתכל בחלון הפרויקט ליד כל תמונה מופיע משולש קטן, אם נלחץ נראה sprite שבתוך התמונה שלנו עם אותו השם.

3. בחרו את תמונה *Letters* בחלון הפרויקט. כל התמונות שייבאנו ה-*Sprite Mode* אמור להיות מוגדר כ-*Signle* וזה מתאים כי כל תמונה מייצגת רק sprite אחד . לעומת זאת התמונה *Letter* אמורה לייצג יותר מספרייט אחד (כל אות בו אמורה להיות ספרייט בפני עצמה). לכן נצטרך לשנות את ההגדרות שלה כך: בחלון האינספקטור נשנה את ה-*Sprite Mode* של *Letter* ל-*Multiple* ונלחץ apply כדי ליישם את השינוי.
נבחר ב-*Sprite Editor*, וחלק את התמונה בהתאם לגודל קבוע עבור כל אות. להסבר מורחב ניתן למצוא בסיכום למשחק חלליות ,פרק אנימציות תת-פרק אפקט פיצוץ.
ונשמור את הסצנה בינתיים.

בניית קלף מ-sprite

במהלך הפרויקט אנחנו הולכים לבנות חפיסה שלימה של קלפים מ-21 הקלפים שייבאנו. זה מאפשר לחסוך במקום בבנייה האחרונה של המשחק.
ספויילר: המבנה של קלף כפי שנראה בהמשך יהיה בצורה של כמה אובייקטים מהצורה כשלהי (כגודל המספר אותו הוא מייצג + שני צורות בצדדים ליד המספר, לא כולל הקלפים המיוחדים: ממשפחת המלוכה או ג'וקרים):



שימוש בתבנית XML בקוד-

ראשית ניצור שלושה סקריפטי-#c בתיקייה Script_ עם השמות *Deck*, *Card* ו- *Prospector*.

- **Card-** מחלקה לקלף אינדיבידואלי בחפיסה. הסקריפט מכיל גם את המחלקה CardDefinition (שמכילה את כל המידע על מיקום הספרייטים בקלף לכל מספר קלף). ומחלקת Decorator (שמחזיקה מידע לקובץ xml).
- **Deck-** החפיסה, מפרש את המידע שב- DeckXML.xml ומשתמש במידע הזה כדי ליצור חפיסה.
- **Prospector-** המחלקה מנהלת את כל המשחק במבט על. כשנוצרת חפיסה מהמחלקה deck המחלקה prospector הופכת אותם לקלפי משחק. המחלקה אוספת את הקלפים לכמה ערימות (למשל ערימה שממנה מושכים או ערימה שזורקים אליה קלפים משומשים) ומנהלת את לוגיקת המשחק.

ראשית נפתח את Card.cs ונזין את הקוד הבא:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Card : MonoBehaviour
{
    //this class will be defined later
}

[System.Serializable]
/* A Serializable class is able to be
 edited in the Inspector*/
public class Decorator
{
    //this class srotes information about each decorator or pip from DeckXML
    public string type; //for card pips, tye="pip"
    public Vector3 loc;//the location of the Sprite on the Card
    public bool flip = false;//whether to flip the Sprite vertically
    public float scale = 1f;//the scale of the Sprite
}
```

```
[System.Serializable]
public class CardDefinition
{
    //this class stores information for each rank of card
    public string face;//sprite to use for each card
    public int rank;//the rank(1-13) of thus card
    public List<Decorator> pips = new List<Decorator>();//pips used
}
```

פתחו את הסקריפט Deck.cs והכניסו את הקוד הבא:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Deck : MonoBehaviour
{
    [Header("Set Dynamically")]
    public PT_XMLReader xmlr;
    //InitDeck is called by prospector wwhen it is ready
    public void InitDeck(string deckXMLText)
    {
        ReadDeck(deckXMLText);
    }
    //ReadDeck parses the XML file passed to it into CardDefinition
    public void ReadDeck(string deckXMLText)
    {
        xmlr = new PT_XMLReader();//create a new PT_XMLReder
        xmlr.Parse(deckXMLText);
        //this prints a test line to show you how xmlr can be used
        string s = "xml[0] decorator[0]";
        s += "type=" + xmlr.xml["xml"][0]["decorator"][0].att("type");
        s += " x=" + xmlr.xml["xml"][0]["decorator"][0].att("x");
        s += " y=" + xmlr.xml["xml"][0]["decorator"][0].att("y");
        s += " scale=" + xmlr.xml["xml"][0]["decorator"][0].att("scale");
        print(s); // it's a MonoBehaviour function
    }
}
```

עכשיו פתחו את מחלקת Prospector.cs והזינו את הקוד הבא:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;//will be used later
using UnityEngine.UI; //will be used later

public class Prospector : MonoBehaviour
{
    static public Prospector S;
    [Header("Set in inspector")]
    public TextAsset deckXML;
    [Header("Set Dynamically")]
    public Deck deck;

    void Awake()
    {
        S = this;//set up a singleton for prospector
    }
    void Start()
    {
        deck = GetComponent<Deck>();// get the Deck
        deck.InitDeck(deckXML.text);// pass DeckXML to it
    }
}
```

כמה דברים שכנראה שמתם לב אליהם: הקסטינג header הוא קאסטינג שמוסיף כותרת מעל לאותו משתנה באינספקטור. כמו באובייקט sound שיצרנו במשחק חלליות- הוספנו את הקאסטינג "range" כדי לקבל כמין כפתור הזזה באינספקטור עבור אותו משתנה. בקאסטינג serializable כבר נתקלנו בעבר. הוא מאפשר לנו סנכרן אובייקטים שלא קשורים למנוע הגרפי בכדי

להשתמש בהם באינספקטור. בהמשך נתעמק יותר ב-xml וכיצד נפרסר אותו בדיוק.
לפני שנמשיך וודאו ששמרתם את כל הקבצים והיכנסו ל-unity.
חיברו את הסקריפטים: prospector.cs ו-Deck.cs לאובייקט המצלמה(_MainCamera) בחלון ההיררכיה ע"י גרירה, וודאו, ע"י בדיקה באינספקטור של המצלמה, שהאובייקטים התחברו כראוי.
אם תסתכלו בחלון הפרויקט יש לנו תיקייה בשם resources בתוך התיקייה יש לנו שני קבצי xml: DeckXML ו- LayoutXML
גררו את הקובץ DeckXML לחלון האינספקטור של המצלמה ברכיב prospector היכן שמופיע Deck XML שאמור לקבל text Asset.
עכשיו לאחר שגררנו בואו נבדוק אם הצליח לנו עד כה. אם אתם זוכרים במחלקת Deck עשינו מתודה שמדפיסה לנו כמין "שורת מבחן" להראות כיצד עובד ה-xml שלנו, למתודה קוראים ReadDeck. כדי לבחון אותה נצטרך חלון console. לחצו *Ctrl+Shift+C*) או לכו ל-Windows -> Genaral -> Console) כדי לפתוח את חלון הקונסול והריצו את המשחק.
אם הכל הלך קשורה אנחנו אמורים לראות על המסך של הקונסול את הדבר הבא:

```
xml[0] decorator[0]type=letter x=-1.05 y=1.42 scale=1.25
```

השורה הזו הגיעה, כפי שכבר אמרנו קודם, מהטסט שעשינו דרך המתודה: *Deck:ReaDeck()* ומראה ש-*ReadDeck()* נקרא כראוי את התכונות *type,x,y* ו-*scale* כפי שהם מופיעים בקובץ *DeckXML.xml* שנמצא בתיקיית ה-*resource*.

פִירסוּר אינפורמציה מ-Deck XML:

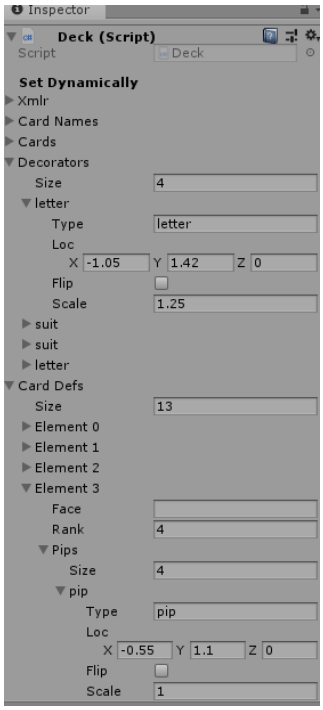
חיזרו לסקריפט Deck ובצעו את השינויים הבאים(די הרבה שינויים):

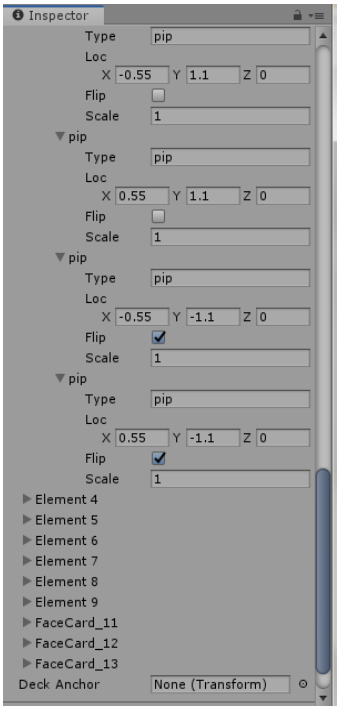
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Deck : MonoBehaviour
{
    [Header("Set Dynamically")]
    public PT_XMLReader xmlr;
    public List<string> cardNames;
    public List<Card> cards;
    public List<Decorator> decorators;
    public List<CardDefinition> cardDefs;
    public Transform deckAnchor;
    public Dictionary<string, Sprite> dictSuits;


    //InitDeck is called by prospector wwhen it is ready
    public void InitDeck(string deckXMLText)
    {
        ReadDeck(deckXMLText);
    }


    //ReadDeck parses the XML file passed to it into CardDefinition
    public void ReadDeck(string deckXMLText)
    {
        xmlr = new PT_XMLReader();//create a new PT_XMLReder
        xmlr.Parse(deckXMLText);
        //this prints a test line to show you how xmlr can be used
        string s = "xml[0] decorator[0]";
        s += "type=" + xmlr.xml["xml"][0]["decorator"][0].att("type");
        s += " x=" + xmlr.xml["xml"][0]["decorator"][0].att("x");
        s += " y=" + xmlr.xml["xml"][0]["decorator"][0].att("y");
        s += " scale=" + xmlr.xml["xml"][0]["decorator"][0].att("scale");
        //print(s); //we done with the test


        //read decorator for all Cards
        decorators = new List<Decorator>();//init the list of decorators
        PT_XMLHashList xDecos = xmlr.xml["xml"][0]["decorator"];
        Decorator deco;
        for (int i = 0; i < xDecos.Count; i++)
```

```
        {
            deco = new Decorator();
            //copy the attributes of the <decorator> to the Decorator
            deco.type = xDecos[i].att("type");
            //bool deco.flip is true if the text io the flip attribute is "1"
            deco.flip = (xDecos[i].att("flip") == "1");
            //floats need to be parsed from the attribute string
            deco.scale = float.Parse(xDecos[i].att("scale"));
            //vector3 loc initializes to [0,0,0],so we need to modify it
            deco.loc.x = float.Parse(xDecos[i].att("x"));
            deco.loc.y = float.Parse(xDecos[i].att("y"));
            deco.loc.z = float.Parse(xDecos[i].att("z"));
            //add temporary deco to the list Decorators
            decorators.Add(deco);
        }

        //read pip location for each card number
        cardDefs = new List<CardDefinition>();
        PT_XMLHashList xCardDefs = xmlr.xml["xml"][0]["card"];
        for(int i=0;i<xCardDefs.Count;i++)
        {
            CardDefinition cDef = new CardDefinition();
            //prase the attibute values and add them to cDef
            cDef.rank = int.Parse(xCardDefs[i].att("rank"));
            //grab an PT_XMLHashList of all the <pip>s on this <card>
            PT_XMLHashList xPips = xCardDefs[i]["pip"];
            if (xPips != null)
            {
                for (int j = 0; j < xPips.Count; j++)
                {
                    //iterate through all the <pip>s
                    deco = new Decorator();
                    //<pip>s on the <card> are handle via the Decorator Class
                    deco.type = "pip";
                    deco.flip = (xPips[j].att("flip") == "1");
                    deco.loc.x = float.Parse(xPips[j].att("x"));
                    deco.loc.y = float.Parse(xPips[j].att("y"));
                    deco.loc.z = float.Parse(xPips[j].att("z"));
                    if (xPips[j].HasAtt("scale"))
                    {
                        deco.scale = float.Parse(xPips[j].att("scale"));
                    }
                    cDef.pips.Add(deco);
                }
            }
            //face cards (Jack,Queen & King) have a face attribute
            if (xCardDefs[i].HasAtt("face"))
            {
                cDef.face = xCardDefs[i].att("face");
            }
            cardDefs.Add(cDef);
        }
    }

}
```

cDef.face הוא שם הבסיס לספרייט של הקלף. למשל *FaceCard_11* הוא שם הבסיס לספרייט של הקלף "נסיך" (Jack) נסיך תלתן (Jack of Clubs) נקרא *FaceCard_11C*, נסיך לב (Heart) נקרא: *FaceCard_11H* וכו'.
עכשיו המתודה ()ReadDeck תפרסר את ה-XML ותהפוך אותו לרשימה של Decorator-ים (הדירוג והורה של כל קלף בצדדים) ו-CardDefinition (מחלקה שמכילה מידע על כל קלף החל מאס ועד למלך).

נחזור ל-unity ונריץ את המשחק. בחרו את המצלמה הראשית והסתכלו באינספקטור שלה על הרכיב Deck, משום שהגדרנו את Decorator ואת CardDefinition כ- [system.Serializable] הרישמות של decorators ו- cardDefs אמורות להופיע.





אילוסטרציה: האינספקטור של המצלמה על הרכיב Deck בזמן ריצת המשחק. ניתן לראות Decorators ו-CardDefs שנקראו מ-DecXML.xml.

עצרו את המשחק ושמרו את הסצנה.

הקצאת ספרייטים לשימוש כקלף-

עכשיו כשה-XML נקרא כראוי ומפורסר לרשימה שניתן להשתמש בה, הגיע הזמן ליצור קלף "מוחשי".
השלב הראשון הוא ליצור רפרנס לכל הספרייטים שעשינו קודם לכן.
1. הוסיפו את הקוד הבא לראש מחלקת Deck כדי שתיהיה לה את המשתנה הבא:

```
public class Deck : MonoBehaviour
{
    [Header("Set in Inspector")]
    //suits
    public Sprite suitClub;
    public Sprite suitDiamond;
    public Sprite suitHeart;
    public Sprite suitSpade;

    public Sprite[] faceSprites;
    public Sprite[] rankSprites;

    public Sprite cardBack;
    public Sprite cardBackGold;
    public Sprite cardFront;
    public Sprite cardFrontGold;

    //prefab
    public GameObject prefabCard;
    public GameObject preabSprite;

  . . .
}
```

נשמור ונחזור ל-unity נוכל לראות שנוסף עכשיו לאינספקטור של המצלמה כמה משתנים שנצטרך להגדיר.

בס"ד