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.

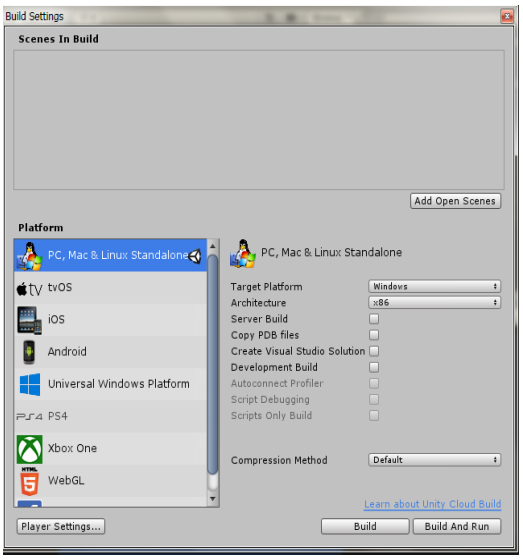
בפרק נבנה משחק סוליטר.
בסוף פרק זה יהיה לנו, לא רק משחק קלפים לטלפון, אלא גם אחלה תבנית למשחקי קלפים אחרים שנרצה לבנות בעתיד.
נראה כמה טכניקות חדשות, הכוללות: שימוש בקונפיגורציות XML ועיצוב משחקים לטלפונים סלולריים.

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,-6),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'.

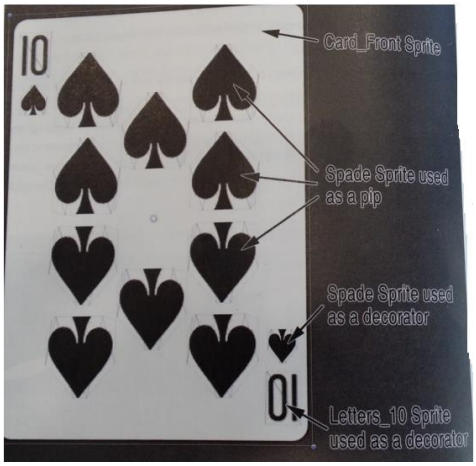
1. נפתח את תיקיית *_Sprties* בחלון הפרויקט, ונבחר את כל התמונות שבו- או ע"י בחירה באחד ואז בליחצה על ctrl + חץ למטה נסמן את כל התמונות עד שנגיע לתחתית התיקייה, או שנבחר את התמונה הראשונה ואז shift + לחיצה על התמונה האחרונה בתיקייה.

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()* קראה כראוי את התכונות *type,x,y* ו-*scale* כפי שהם מופיעים בקובץ *DeckXML.xml* שנמצא בתיקיית ה-*resource*.

פִירסוּר אינפורמציה מ-Deck XML:

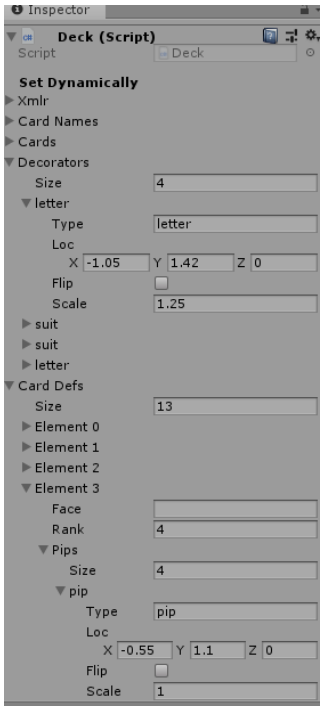
חיזרו לסקריפט Deck ובצעו את השינויים הבאים(די הרבה שינויים):

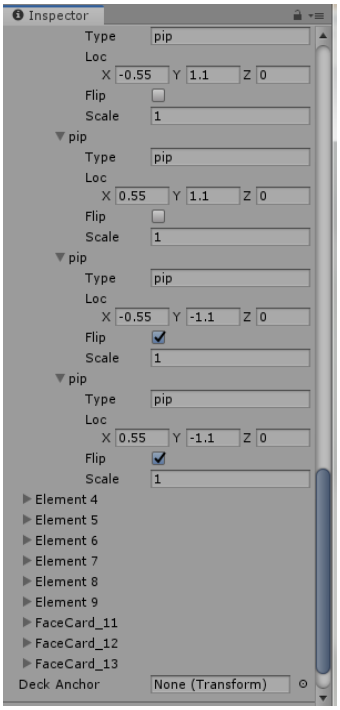
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

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 when 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 המשתנים הבאים:

```
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.

2. גררו את הטקסטורות:*Heart*, *Diamond*, *Club* ו-*Spade* מהתיקייה Sprite_ בחלון הפרויקט לתוך המשתנה התואם לו בDeck (*suitHeart*, *suitDiamond*, *suitClub* ו-*suitSpade*). Unity אוטומטית תקצאה את הספרייטים לתוך המשתנים.

3. ניכנס לאינספקטור של המצלמה, אם נשים לב יש לנו מעל למצלמה תמונה קטנה של מנעול. לחצו עליו בכדי לנעול את האובייקט. הנעילה דואגת שהוא לא ישנה איזה אובייקט יוצג מתי שנשנה למשהו חדש.

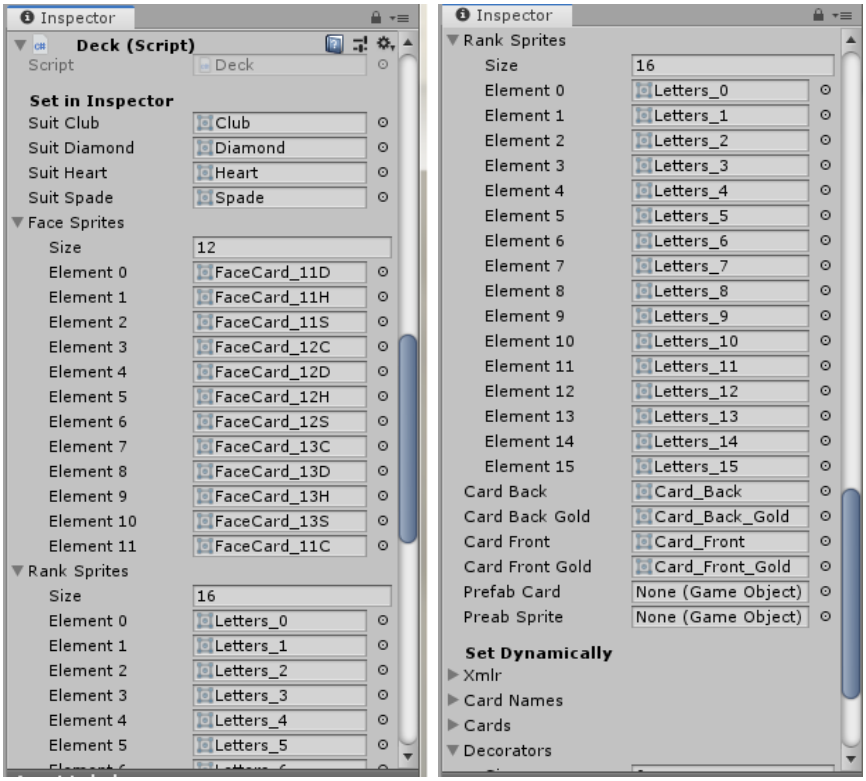
4. הקצאה כל אחד מהספרייטים שמתחילים ב-*FaceCard_* לאלמנט של המערך faceSprites ברכיב Deck:

- בחרו ב-*FaceCard_11C* בתיקייה *sprite_* ולחצו shift ובעוד אתם לוחצים עליו תבחרו *FaceCard_13S*, זה אמור לבחור בכל הספרייטים ביחד.
- גררו את קבוצת הספרייטים שבחרנו למערך *faceSprites* זה אמור להכניס את כל האובייקטים לתוך המערך. סה"כ המערך אמור להיות בגודל 12 (4ספרייטים לכל דרגה של קלף ויש שלושה:11,12,13).
- במידה ולא הצלחנו ניתן להגדיר את גודל(ע"י שינוי size) המערך ולהעביר ספרייט ספרייט לתוכו.

5. לחצו על המשולש הקטן ליד המולטי-ספרייט *Letters* בתיקייה Sprites_ עתה כמו התהליך שעשינו בשלב הקודם (עם הגרירה של הספרייטים למערך *faceSprites*)- סמנו את כל הספרייטים ב-*Letters* וגררו אותם למערך *rankSprites*.
עתה המערך אמור להיות בגודל 16 (יש שש-עשרה דרגות 0-15),ומלא בכל הספרייטים של *Letters* .
וודאו שהספרייטים ממוינים בסדר הנכון (Letters_0 במקום ה-0 במערך,Letters_1 במקום ה-1במערך,...,Letters_15 במקום ה-15 במערך) . אם לא סדרו אותם ע"י השמה במקום הנכון.

6.גררו את הספרייטים *Card_Front*, *Card_Back_Gold*, *Card_Back* ו- *Card_Front_Gold* מחלון הפרויקט למקום המתאים להם באינספקטור (ברכיב Deck) .

האינספקטור שלנו אמור להיראות ככה:



אילסוטרציה: הרכיב Deck באינספקטור של האובייקט MainCamera_ לאחר השמה נכונה מהשלבים הקודמים.

7. בטלו את נעילת האובייקט באינספקטור(המנעול הקטן למעלה) ושמרו את השינויים.

יצירת קלפים בקוד-

בדיוק כמו כל דבר אחר על המסך, ספרייטים צריכים להיות סגורים ב-GameObject . לפרויקט זה אנחנו צריכים שני אובייקטי משחק- אובייקט *PrefabSprite* שישמש לכל ה-decorators וה-pips (שייבאנו ב-asset הראשוני של המשחק),*PrefabCard* שתוחם את הבסיס לכל הקלפים בחפיסה.
כדי לייצור את ה-PrefabCard נעשה את הדברים הבאים:

- מהתפריט הראשי בחרו GameObject -> 2D Object -> Sprtie קראו לאובייקט זה בשם *PrefabCard*.
- גררו את *Card_Front* מחלון הפרויקט למשתנה הספרייט של ה-*Sprite Renderer* ב-PrefabCard. עתה אנחנו אמורים לראות את ה Card_Front בחלון הסצנה.
- גררו את הסקריפט *Card.cs* ל-PrefabCard.
- באינספקטור של ה-PrefabCard לחצו על הכפתור *Add Component* חיפשו *Physics* -> *Box Collider*. גודל ה-Box Collider אמור להיות מאותחל ל-[2.56,3.56,0.2] במידה ולא שנו אותו שיתאים.
- הפכו את האובייקט PrefabCard ל-*prefab* ע"י גרירת האובייקט לחלון הפרוייקט(לתיקייה *_Prefab*).
- מחקו את האינסטנס של PrefabCard מחלון ההיררכיה ושמרו את הסצנה.

עתה אנחנו צריכים להתאים את ה-prefabCard וה-PrefabSprite למקום המתאים להם ברכיב Deck של המצלמה הראשי:

- בחרו במצלמה הראשית וגררו את הprefabs שייצרנו מחלון הפרויקט למקום המתאים להם באינספקטור.
- שמרו את הסצנה.

בניית הקלפים בקוד-

לפני שממש נוסיף את המתודה למחלקת Deck כדי ליצור את הקלפים, אנחנו צריכים להוסיף משתנים למחלקת Card:
1. שנו את ההערה בראש מחלקת Card (`//this class will be defined later`) לקוד הבא:

```
public class Card : MonoBehaviour
{
    [Header("Set Dynamically")]
    public string suit;//(C,D,H,or S)
    public int rank;//(1-14)
    public Color color = Color.black;//color to tint pips
    public string colS = "Black";//or Red. Name thr color

    //this list hold all of thre Decorator GameObjects
    public List<GameObject> decoGOs = new List<GameObject>();
    //this list holds all thr pip GameObject
    public List<GameObject> pipGOs = new List<GameObject>();
    public GameObject back;//the GameObject of the back of the card
    public CardDefinition def;//parsed from DeckXML.xml
}
```

2. הוסיפו את הקוד הבא ל-Deck במתודה *InitDeck()* שנו לקוד הבא:

```
//InitDeck is called by prospector when it is ready
    public void InitDeck(string deckXMLText)
    {
        //this creates an anchor for all the Card GameObjects in the Hierarchy
        if (GameObject.Find("_Deck") == null)
        {
```

```
            GameObject anchorGO = new GameObject("_Deck");
            deckAnchor = anchorGO.transform;
        }
        //initialize the Dictionary of SuitSprites with necessary Sprites
        dictSuits = new Dictionary<string, Sprite>()
        {
            {"C",suitClub},
            {"D",suitDiamond },
            {"H",suitHeart },
            {"S", suitSpade }
        };

        ReadDeck(deckXMLText);//this will preexisting line from earlier
        MakeCards();
    }
```

אחרי *ReadDeck()* הוסיפו את המתודות הבאות:

```
//get the proper CardDefinition based on Rank(1 to 14)
    public CardDefinition GetCardDefinitionByRank(int rnk)
    {
        //search through all of the CardDefinition
        foreach (CardDefinition cd in cardDefs)
        {
            //if the rank is correct, return this definition
            if (cd.rank == rnk)
            {
                return (cd);
            }
        }
        return null;
    }

    //make the card GameObject
    public void MakeCards()
    {
        //cardName will be the names of crds to build
        //each suit goes from 1 to 14 (e.g., C1 to C4 for Clubs)
        cardNames = new List<string>();
        string[] letters = new string[] { "C", "D", "H", "S" };
        foreach (string s in letters)
        {
            for (int i = 0; i < 13; i++)
            {
                cardNames.Add(s + (i + 1));
            }
        }
        //make list to hold all  the cards
        cards = new List<Card>();

        //iterate through all of the card names that were just made
        for (int i = 0; i < cardNames.Count; i++)
        {
            //makethe cards and add it to the cards Deck
            cards.Add(MakeCard(i));
        }
    }


    private Card MakeCard(int cNum)
    {
        //create a new Card GameObject
        GameObject cgo = Instantiate(prefabCard) as GameObject;
        //set the transform.parent of the new card to the anchor
        cgo.transform.parent = deckAnchor;
        Card card = cgo.GetComponent<Card>();//get Card component
        //this line stacks the cards so that they're all in nice rows
        cgo.transform.localPosition = new Vector3((cNum % 13) * 3, cNum / 13 * 4, 0);

        //assign basic values to the card
        card.name = cardNames[cNum];
        card.suit = card.name[0].ToString();
        card.rank = int.Parse(card.name.Substring(1));
        if (card.suit == "D" || card.suit == "H")
```

```
        {
            card.colS = "Red";
            card.color = Color.red;
        }

        //pull the CardDefinition for this card
        card.def = GetCardDefinitionByRank(card.rank);

        AddDecorators(card);

        return card;
    }
    //temporary veriables will be reused several times in helper methods
    private Sprite _tSp = null;
    private GameObject _tGO = null;
    private SpriteRenderer _tSR = null;

    private void AddDecorators(Card card)
    {
        //Add Decorators
        foreach(Decorator deco in decorators)
        {
            if (deco.type == "suit")
            {
                //instantiate a Sprite GameObject
                _tGO = Instantiate(prefabSprite) as GameObject;
                //get the spriteRenderer Component
                _tSR = _tGO.GetComponent<SpriteRenderer>();
                //set the Srite to the proper suit
                _tSR.sprite = dictSuits[card.suit];
            }
            else
            {
                _tGO = Instantiate(prefabSprite) as GameObject;
                _tSR = _tGO.GetComponent<SpriteRenderer>();
                //get the proper sprite to show this rank
                _tSp = rankSprites[card.rank];
                //assign this rank sprite to the SpriteRenderer
                _tSR.sprite = _tSp;
                //set the color of the rank to match the suit
                _tSR.color = card.color;
            }
            //make the deco Sprites render above the Card
            _tSR.sortingOrder = 1;
            //make the decorator Sprites render above the Card
            _tGO.transform.SetParent(card.transform);
            //set the localPosition based pn the location from DeckXML
            _tGO.transform.localPosition = deco.loc;
            //flip the Decorator if needed
            if (deco.flip)
            {
                // an Euler rotation of 180 around the Z-axis will flip it
                _tGO.transform.rotation = Quaternion.Euler(0, 0, 180);
            }
            //set the scale to keep deco from being too big
            if (deco.scale != 1)
            {
                _tGO.transform.localScale = Vector3.one * deco.scale;
            }
            //name this GameObject so it's easy to see
            _tGO.name = deco.type;
            //add this deco GameObject to the List card.decoGOs
            card.decoGOs.Add(_tGO);
        }
    }
```

MakeCard() ו- *AddDecorator* הן מתודות פרטיות שעוזרות למתודה *MakeCards()*. הן מקצרות את הכתיבה של המתודה.

3. שמרו את הסקריפטים, וחזרו ל-unity. הריצו את המשחק, אתם אמורים לראות 52 קלפים מסודרים לפי הצורה בחלון הסצנה ובהיררכיה. אין להם עדיין סימונים באמצע(pips), אבל הקלפים בכל זאת מופיעים עם הצבע וה-*Decorators* הנכונים.

4. עכשיו נוסיף את ה-pips (הצורות במרכז הקלף)וה-faces (הקלפים שיש להם ציור כמו מלך, מלכה וכו')דרך עוד שלוש מתודות עזר במחלקת Deck:

```
private void AddPips(Card card)
    {
        //for each of the pips in the definition..
        foreach (Decorator pip in card.def.pips)
        {
            //instantiate a Sprite GameObject
            _tGO = Instantiate(prefabSprite) as GameObject;
            //set the parent to be the card GameObject
            _tGO.transform.SetParent(card.transform);
            //set the position to that specified in the XML
            _tGO.transform.localPosition = pip.loc;
            //flip it if necessary
            if (pip.flip)
            {
                _tGO.transform.rotation = Quaternion.Euler(0, 0, 180);
            }
            //scale it necessary
            if (pip.scale != 1)
            {
                _tGO.transform.localScale = Vector3.one * pip.scale;
            }
            //give it a name
            _tGO.name = "pip";
            //get the SpriteRenderer componenet
            _tSR = _tGO.GetComponent<SpriteRenderer>();
            //set the sprite to the proper suit
            _tSR.sprite = dictSuits[card.suit];
            //set storingOrder so the pip is rendered above the Card_Front
            _tSR.sortingOrder = 1;
            //Add this  to the Card's list of pips
            card.pipGOs.Add(_tGO);
        }
    }

    private void AddFace(Card card)
    {
        if (card.def.face == "")
        {
            return; //no need to run
        }

        _tGO = Instantiate(prefabSprite) as GameObject;
        _tSR = _tGO.GetComponent<SpriteRenderer>();
        //generate the right name annd pass it to GetFace()
        _tSp = GetFace(card.def.face + card.suit);
        _tSR.sprite = _tSp;
        _tSR.sortingOrder = 1;
        _tGO.transform.SetParent(card.transform);
        _tGO.transform.localPosition = Vector3.zero;
        _tGO.name = "face";
    }

    //Find the proper face card Sprite
    private Sprite GetFace(string faceS)
    {
        foreach (Sprite _tSP in faceSprites)
        {
            if (_tSP.name == faceS)
            {
                return (_tSP);
            }
        }
        return null;
    }
```

המתודה הראשונה(*AddPips*) אמורה להוסיף את ה-pips לקלפים, השנייה (*AdFace*) אמורה להוסיף את התמונה המתאימה לקלפים "המיוחדים" עם הפנים (מלך, מלכה ונסיך) ולמתודה עשינו מתודת עזר (*GetFace*) שמחפשת את התמונה המתאימה לקלף בהתאם לסוג שלו.
עכשיו נוסיף קריאה לשני המתודות לעיל מ- *MakeCard()* (נקרא להן אחרי הקריאה ל- *AddDecorator*) :

```
private Card MakeCard(int cNum)
    {
    . . .

      AddDecorators(card);
      AddPips(card);
      AddFace(card);
     }
```

5. נשמור את הסקריפטים ונריץ את המשחק. אנחנו אמורים לראות את כל הקלפים מפורסרים כראוי עם הצורות בקלפים ועם הציור המתאים לקלפים המיוחדים.

השלב הבא שנעשה הוא להוסיף לקלפים "גב" , כלומר עכשיו נעבוד על מה שקורה אם הפכנו את הקלף על פניו.
למעשה לקלף לא יהיה ממש גב, אלא פשוט נוסיף לכל קלף ספרייט שיהיה בשכבה מעל לפרונט הקלף, ומתי ש"נהפוך" את הקלף, פשוט נגרום לאותו ספרייט להיות נראה (Visible), אחרת הוא יוגדר כלא נראה(Invisible).

6. כדי ליצור את האפקט של "גב" הקלף הוסיפו לקוד של מחלקת Card את המשתנה הבולייאני faceUp שבתוכו יש get ו- set כך:

```
public class Card : MonoBehaviour
{
. . .

public bool faceUp
    {
        get
        {
            return (!back.activeSelf);
        }
        set
        {
            back.SetActive(!value);
        }
    }

}
```

7. שמרו וחיזרו לקלפים במחלקת Deck . הוסיפו את השדות והמתודות עזר הבאות:
בראש המחלקה הוסיפו את המשתנה הבולייאני startFaceUp :

```
public class Deck : MonoBehaviour
{
```

```
    [Header("Set in Inspector")]
    public bool startFaceUp = false;

    //suits
    public Sprite suitClub;
     .   .   .
```

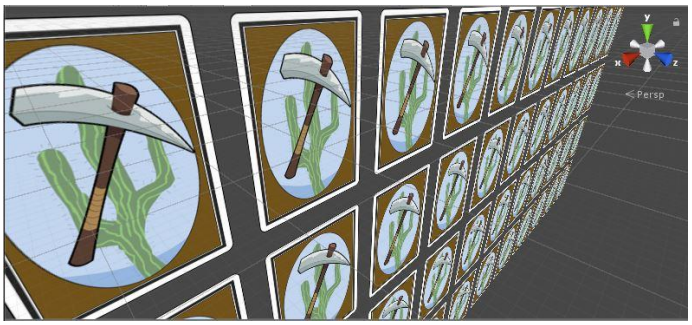
צרו את המתודה באה בתחתית המחלקה:

```
private void AddBack(Card card)
    {
        //add Card Back
        //the Card_Back will be able to cover everything else on the Card
        _tGO = Instantiate(prefabSprite) as GameObject;
        _tSR = _tGO.GetComponent<SpriteRenderer>();
        _tSR.sprite = cardBack;
        _tGO.transform.SetParent(card.transform);
        _tGO.transform.localPosition = Vector3.zero;
        //this us a higher sortingOrder than anything else
        _tSR.sortingOrder = 2;
        _tGO.name = "back";
        card.back = _tGO;
        //Default to face-up
        card.faceUp = startFaceUp;//Use the property of faceUp of Card
    }
```

לכו למתודה *MakeCard(int cNum)* וקראו מתודה שיצרנו כרגע ביחד בסוף המתודה:

```
private Card MakeCard(int cNum)
    {
        .   .   .
  AddDecorators(card);
        AddPips(card);
        AddFace(card);
        AddBack(card);

        return card;
    }
```

שמרו את הסקריפטים וחזרו ל-unity . אם תריצו את המשחק אתם אמורים לראות את הקלפים מסודרים כמו מקודם רק שפניהם כלפי מטה (הפוכים):



הפסיקו את ההרצה של המשחק, שנו את המשתנה startFaceUP ברכיב Deck של האינספקטור של המצלמה הראשית ל-true והריצו שוב את המשחק. עתה כל הקלפים כלפי מעלה שוב.
שמרו את הסצנה.

ערבוב החפיסה-

עכשיו כשהקלפים יכולים להיבנות ולהיות מוצגים על המסך, הדבר האחרון שנצטרך ממחלקת Deck הוא היכולת לערבב את החפיסה.

1. הוסיפו את המתודה הבאה לסוף המחלקה:

```
public class Deck : MonoBehaviour
{
. . .
static public void Shuffle(ref List<Card> oCards)//ref means sendind referance to the function
    {
        //Create a temporary List to hold the new shuffle order
        List<Card> tCard = new List<Card>();

        int ndx; //this will the index of the card to be moved
        tCard = new List<Card>();
        //repeat as long as there are cards in the original List
        while (oCards.Count > 0)
        {
            //pick index of a random card
            ndx = Random.Range(0, oCards.Count);
            //add that card to the temporary List
            tCard.Add(oCards[ndx]);
            //and remove that card from the original List
            oCards.RemoveAt(ndx);
        }
        //replace the original List with the temporary List
        oCards = tCard;
        /*because oCards is a referance (ref) parameter,
          the original argument that was passed in is changed as well
        */
    }
}
```

המילה השמורה ref, בפרמטרים של המתודה, מוודא ש-List<Card> שעבר ל- List<Card> oCards יועבר כרפרנס ולא יועתק למשתנה oCards. מה שאומר שמה שיקרה ל-oCards קורה למשתנה של המתודה. במילים אחרות, אם הקלפים של Deck מועברים דרך המתודה, הקלפים הם האלה הם אלו שיעורבבו ולכן אין צורך ב-return מהפונקציה.

2. הוסיפו את השורות ל- Prospector.Start():

```
public class Prospector : MonoBehaviour
{
. . .
  void Start()
    {
        deck = GetComponent<Deck>();// get the Deck
        deck.InitDeck(deckXML.text);// pass DeckXML to it
        Deck.Shuffle(ref deck.cards);//this shuffles the deck by reference
        Card c;
        for(int cNum=0; cNum<deck.cards.Count; cNum++)
        {
            c = deck.cards[cNum];
            c.transform.localPosition = new Vector3((cNum % 13) * 3, cNum / 13 * 4, 0);
        }
    }
}
```

אתם חייבים גם כאן להשתמש ב-ref מתי שקוראים למתודה.
הלולאת for מציגה את הקלפים על המסך בסדר החדש שלהם.
3. אם נשמור את הסקריפט ונריץ את המשחק נוכל לראות כי הקלפים מעורבבים כרצוי.
עתה כשמחלקת Deck יכולה לערבב כל רשימת קלפים, יש לנו את הכלים הבסיסיים לשימוש בחפיסת קלפים: