Ravensburger

Un juego de cartas en 2D

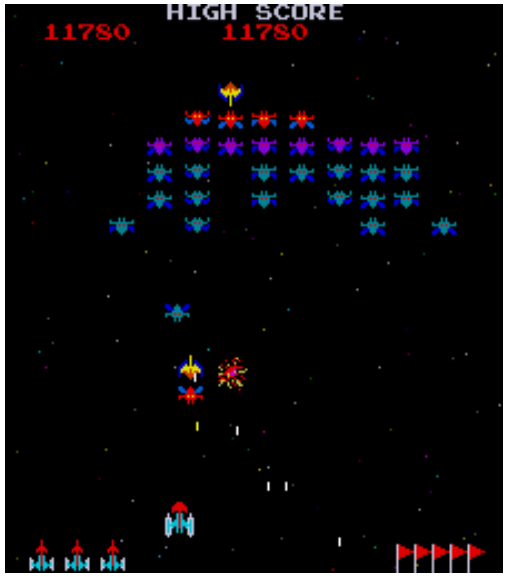
Índice

- Introducción
- Diseño del juego
- Creando el proyecto
- Atlas de texturas
- Ajustando la cámara
- Configurando las cartas
- Lógica de las parejas
- Interfaz de usuario

Introducción

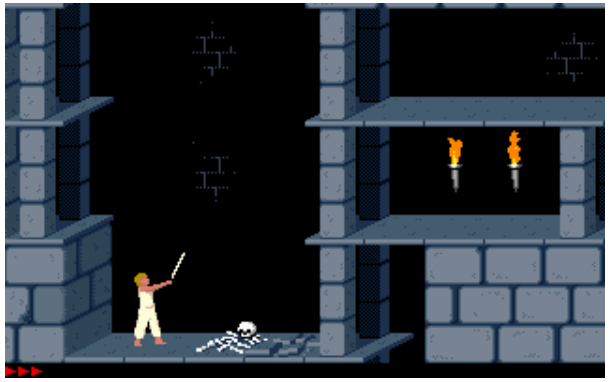
- En esta unidad empezaremos a estudiar las herramientas de Unity para construir juegos 2D
- Unity empezó como plataforma de juegos 3D, pero ha ido incorporando herramientas para facilitar la implementación de juegos 2D
  - Y de hecho, la forma de trabajar es muy parecida
- Implementaremos un juego de memoria:
  - Inicialmente, todas las cartas se encuentran boca abajo
  - El objetivo del juego es emparejar todas las cartas
  - El jugador destapa dos cartas y, si no coinciden, se vuelven a voltear

Introducción



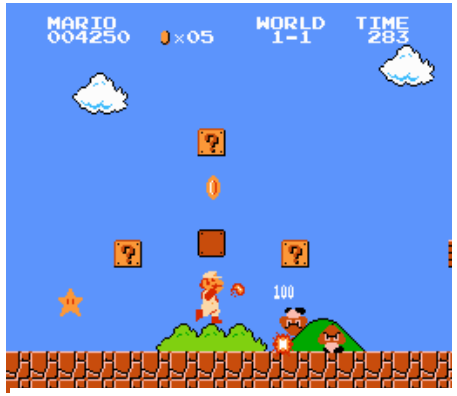
Galaxian, Namco. 1979

Gauntlet, Atari Games. 1985



Prince of Persia,Broderbund. 1989

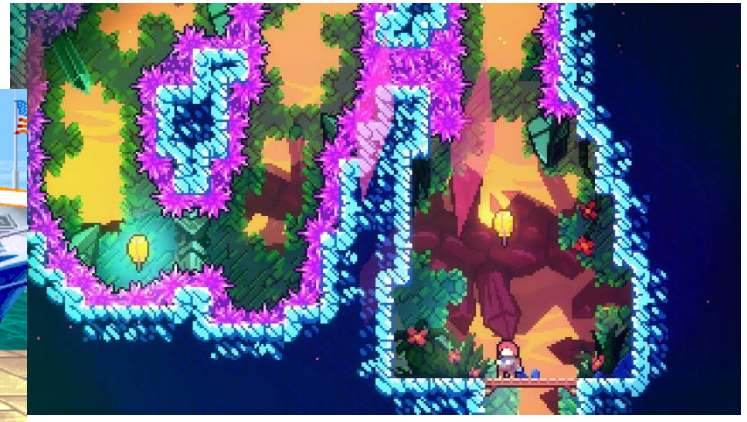
Stardew Valley, ConcernedApe. 2016



Super Mario Bros, Nintendo. 1985

Castlevania, Konami. 1986

Street Fighter II: The World Warrior, Capcom. 1991



Celeste, Matt Makes Games. 2018

Introducción

- El elemento principal de los juegos 2D son los *sprites*:
  - Son bitmaps bidimensionales usados para construir el juego. Incluyen color y transparencia
  - Los elementos interactivos de un juego 2D suelen ser sprites
  - Los sprites animados están formados por una secuencia de imágenes
  - Las consolas y arcades antiguas tenían soporte hardware para dibujar sprites. La CPU sólo tenía que definir su posición (y, a veces, escalado y rotación)

Introducción

- Los sprites se almacenan en ficheros de tipo imagen

Formato		Alfa
PNG	Lossless	SI
GIF	Lossy (color)	NO
TIFF	No/Lossless	NO
JPG	Lossy	NO
TGA	No/Lossless	SI
PSD	No	SI
BMP	No	NO

Formato HDR
EXR
HDR

Importando modelos 3D

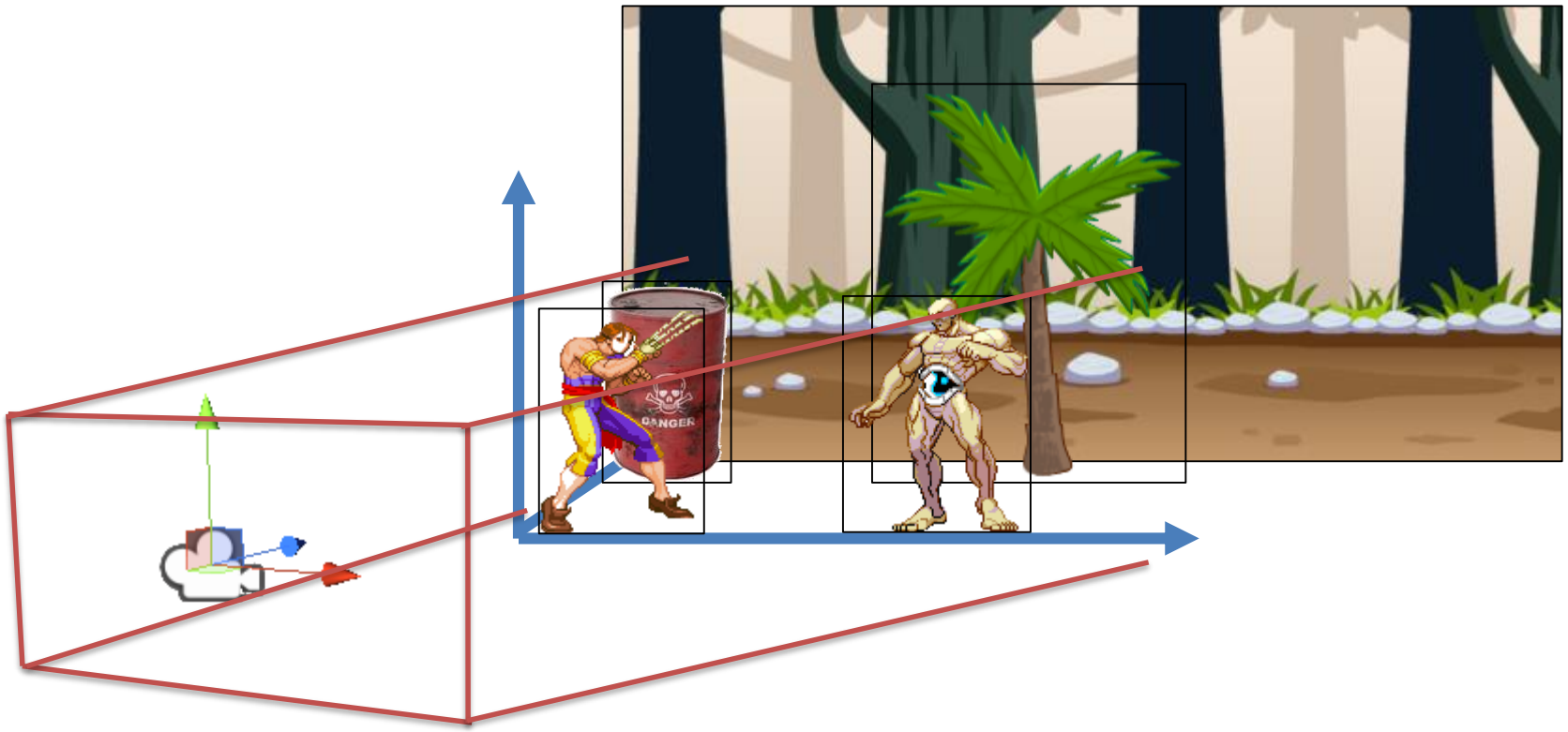
- Repositorios de *tilesets*

Sitio	Pago	Gratis
Unity Asset Store	X	X
OpenGameArt		X
Kenney		X
itch.io	X	X

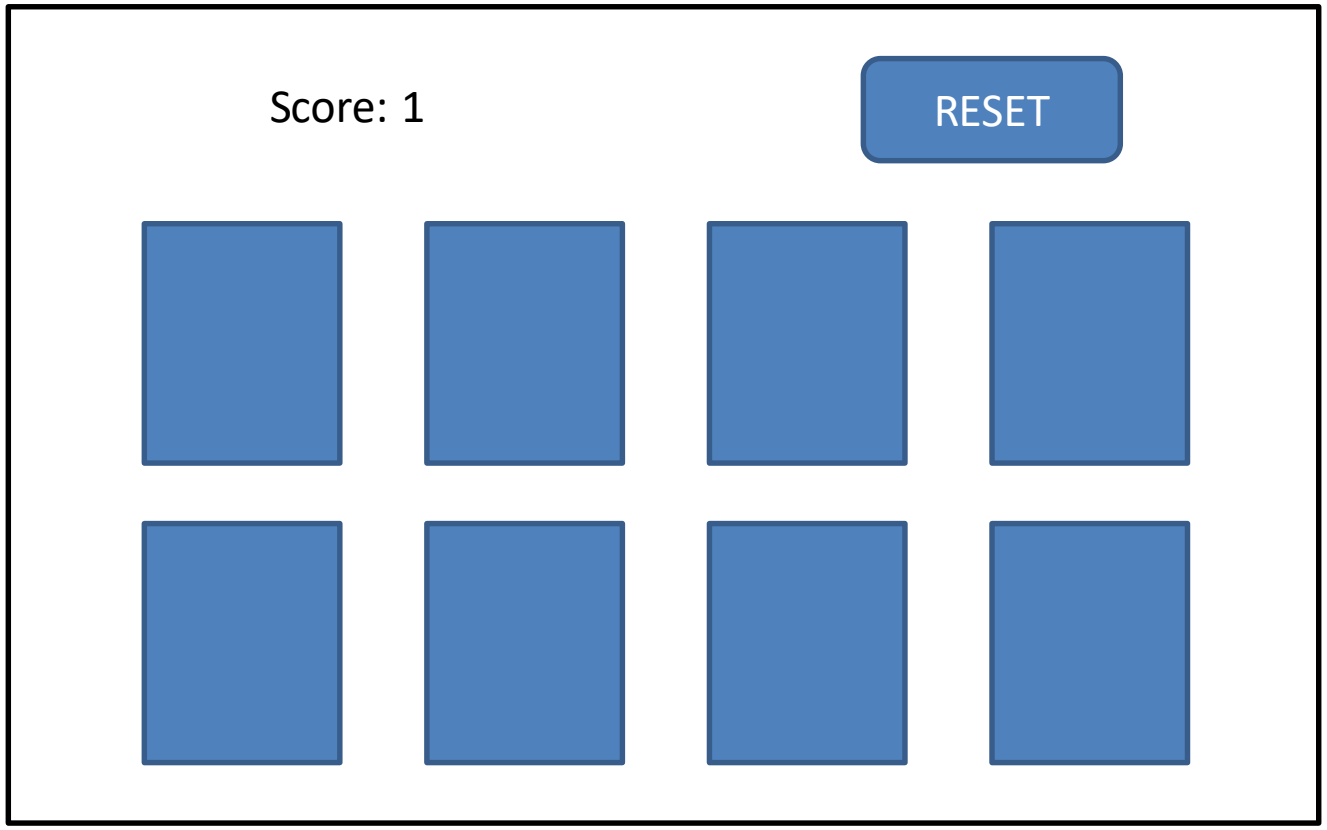
- Generadores de *tilesets*
  - https://sanderfrenken.github.io/Universal-LPC-Spritesheet-Character-Generator
  - https://0x72.itch.io/pixeldudesmaker

Introducción

- Aunque estemos construyendo un juego 2D a partir de sprites, tenemos una escena 3D

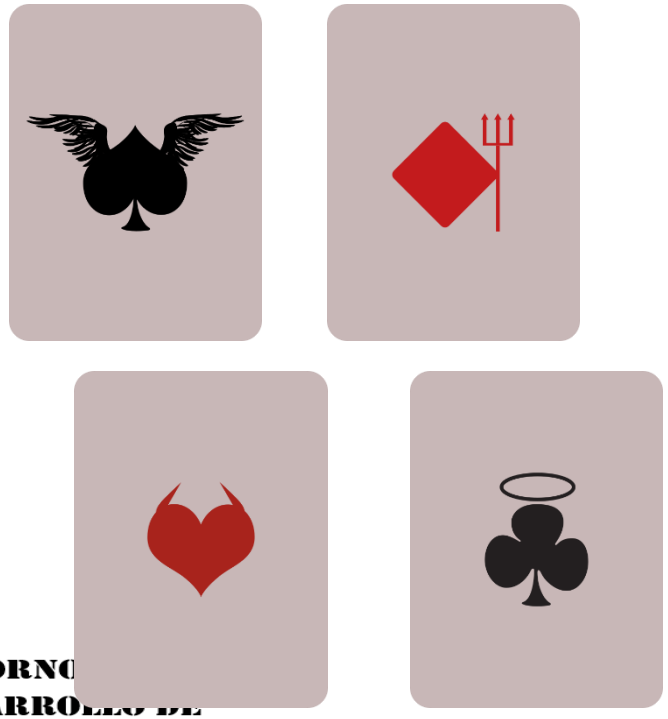


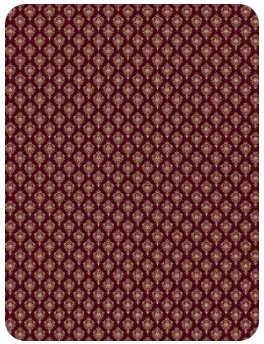
Diseño del juego

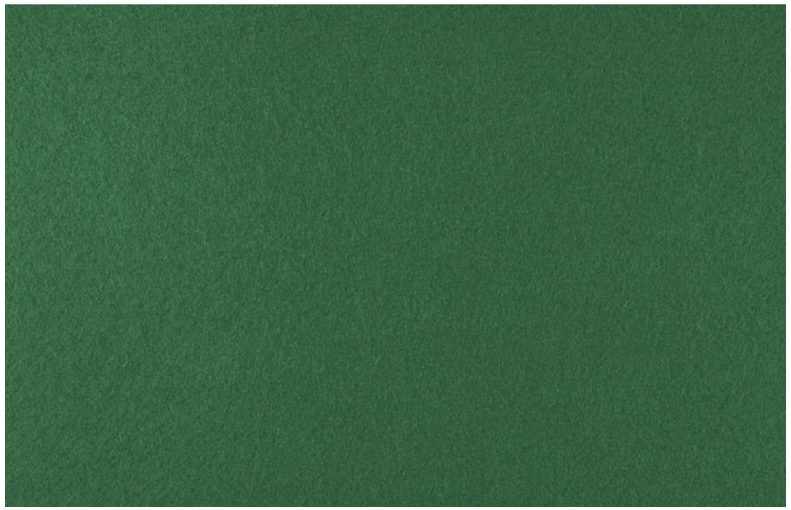


Diseño del juego

- Consigue los elementos que usaremos para construir el juego:
  - Un fondo, el botón de reset, la cara trasera de las cartas y las frontales



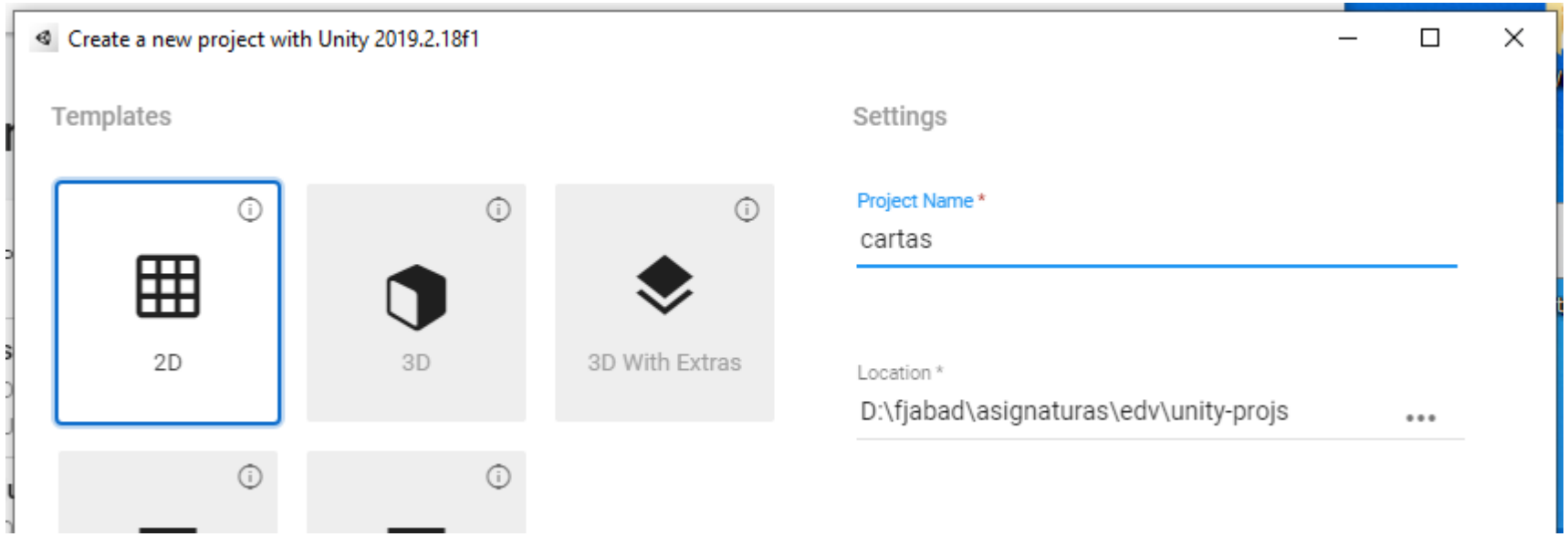




https://openclipart.org/detail/169165/card-suits-angelic-or-devilish

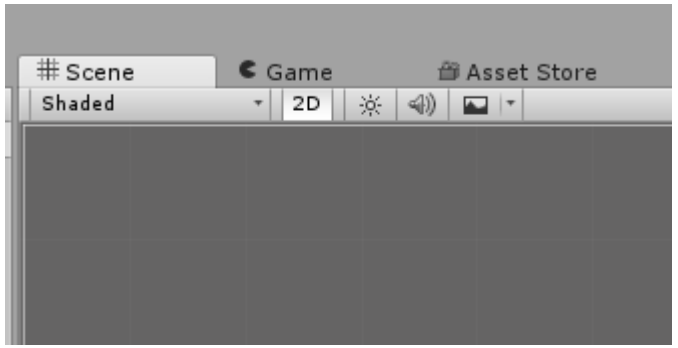
Creando el proyecto

- Crea un nuevo proyecto en Unity
- En el tipo de proyecto, elige 2D



Creando el proyecto

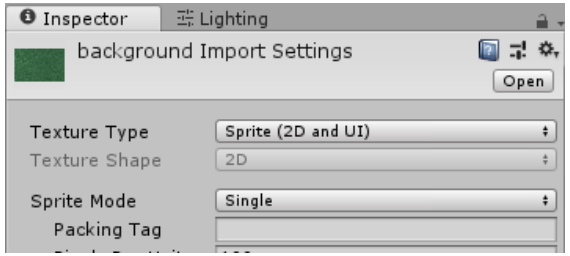
- En realidad, el tipo de proyecto únicamente cambia un par de configuraciones del editor, por lo demás da igual elegir 2D o 3D
  - Al importar una imagen, por defecto será tipo sprite (en vez de textura)
  - No hay iluminación por defecto
  - La cámara por defecto es ortográfica

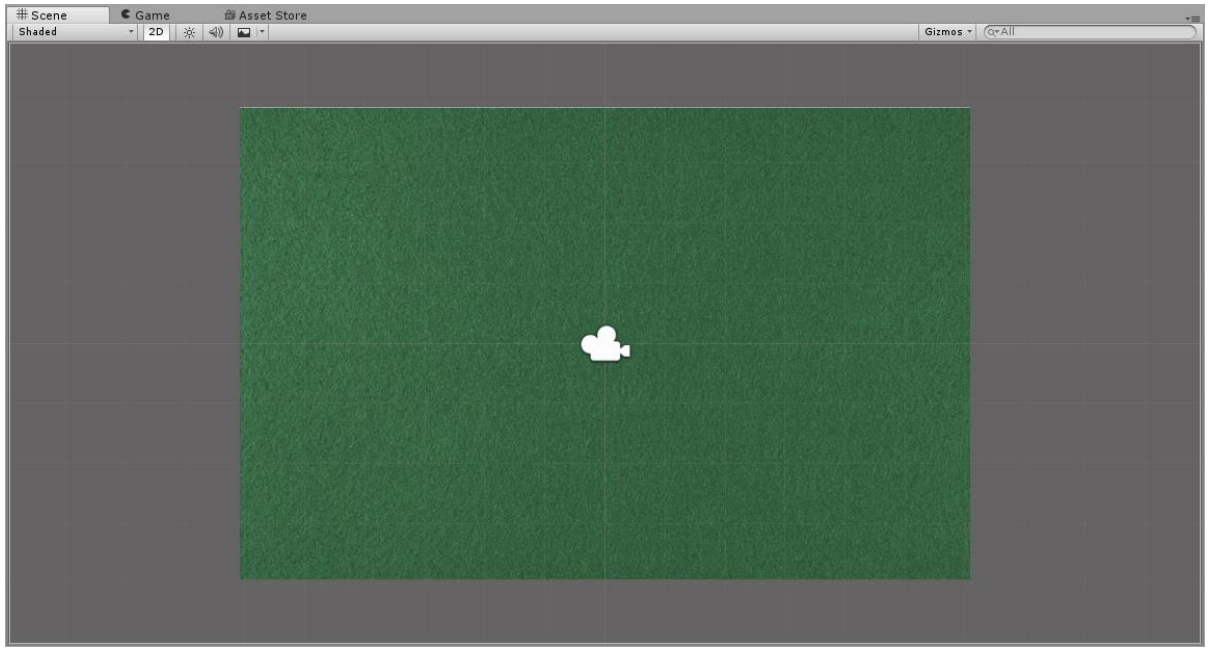


El editor está configurado en modo 2D (no se muestra el eje Z)

Importa los assets

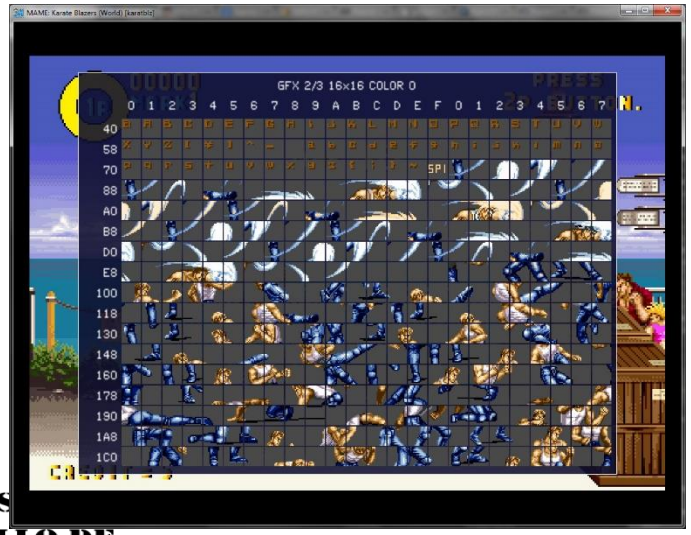
- Importa los assets en una carpeta del proyecto (asegúrate de que se han importado como sprites)
- Sitúa la imagen de fondo en (0, 0, 5), para asegurarnos que sale detrás de el resto de la escena

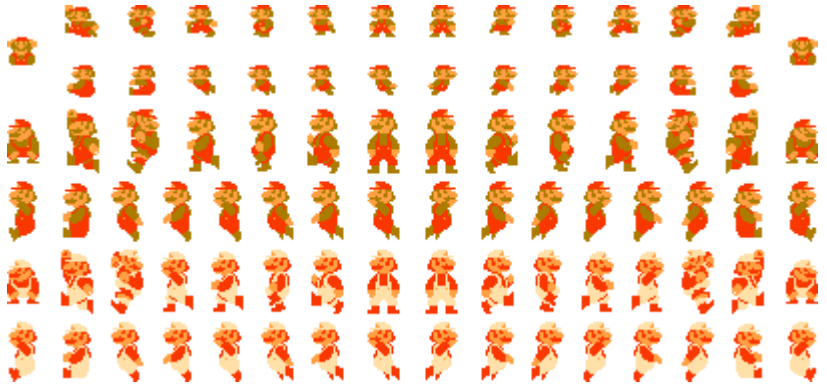




Atlas de texturas

- Un juego 2D puede contener centenares de imágenes (especialmente cuando hay sprites animados)
- Tener los sprites separados en ficheros es útil durante el desarrollo, pero trae problemas en ejecución
  - Espacio perdido, muchas llamadas de dibujo, muchos cambios de texturas, etc.
- Por ello es común agrupar sprites en una sola imagen (atlas o *sprite sheets*)



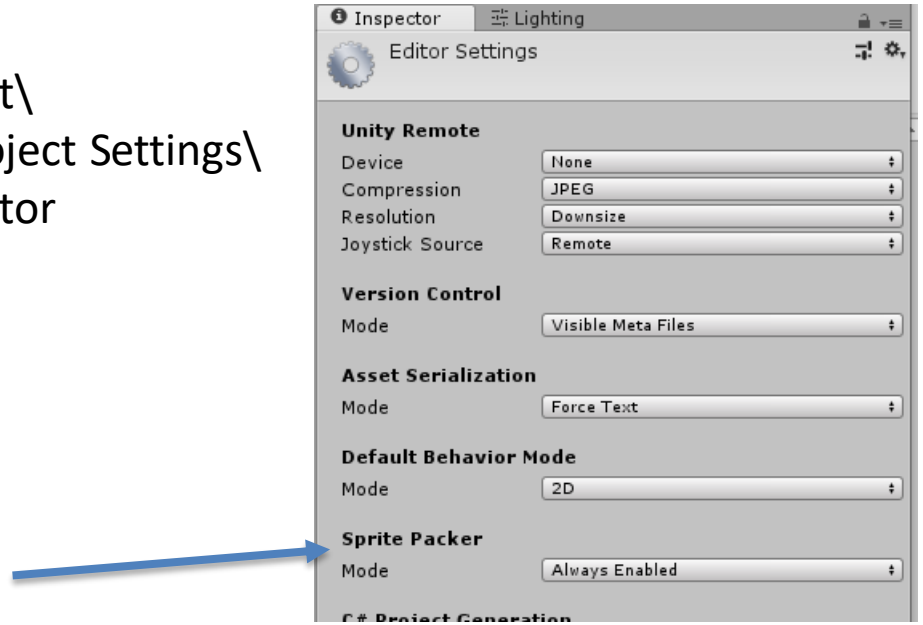


http://www.chronocrash.com/forum/index.php?topic=813.0

Atlas de sprites

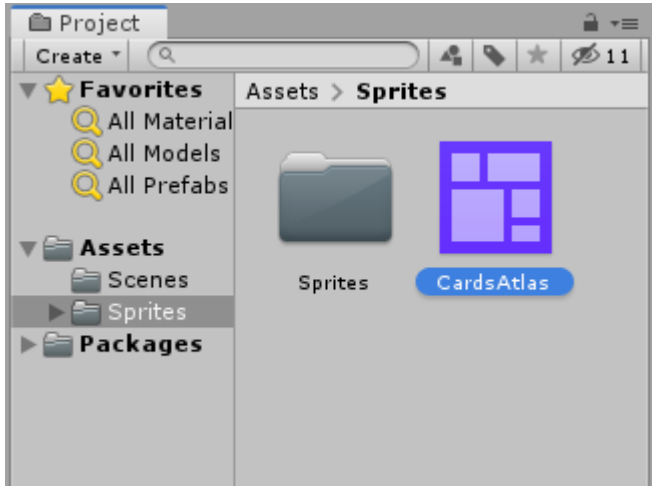
- Unity permite trabajar con imágenes individuales, para luego empaquetarlas al compilar

Edit\
Project Settings\
Editor

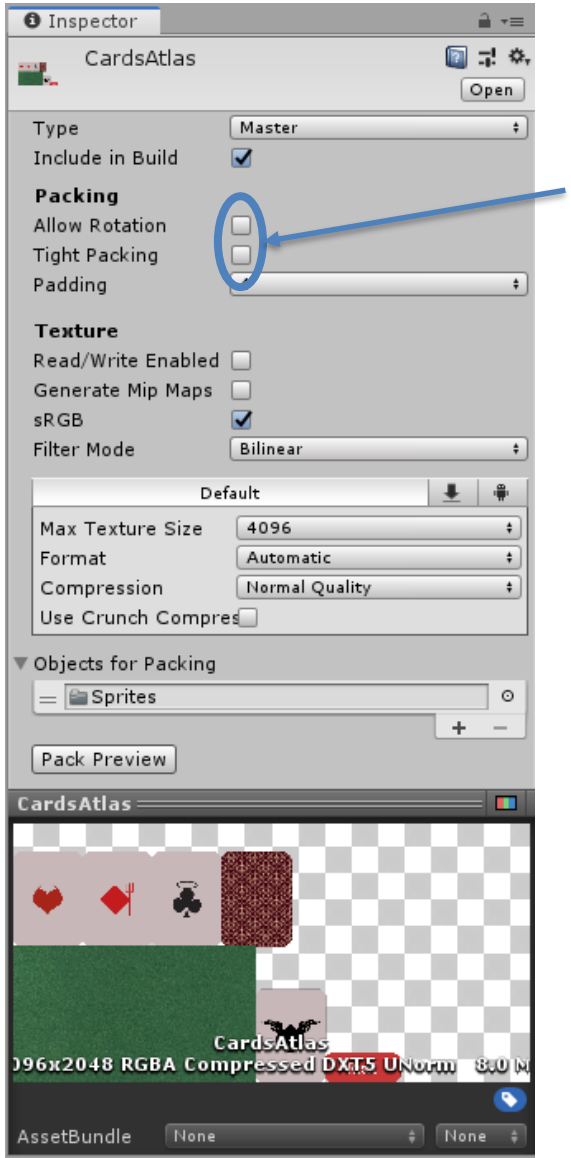


Organiza todos los sprites que quieras empaquetar en un directorio

Create\Sprite Atlas



Atlas de sprites



Para el tipo de juego que vamos a hacer, desmarca Allow Rotation y Tight Packing

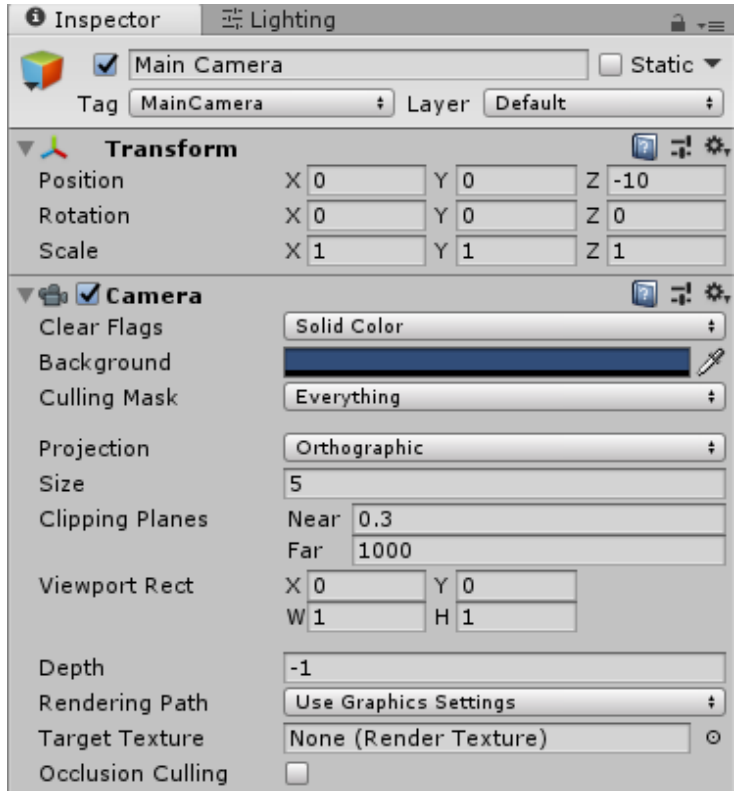
Elige un tamaño máximo de textura suficiente para almacenar todos los sprites

Selecciona todos los sprites a empaquetar (también acepta directorios)

Pulsa Pack Preview para generar visualizar el atlas

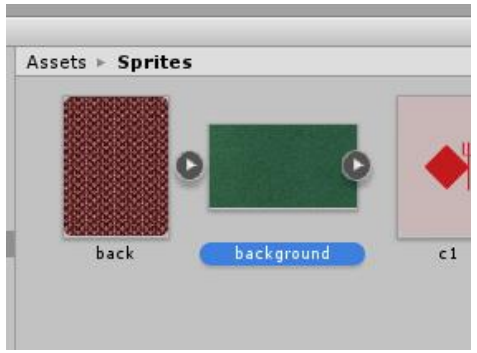
Ajustando la cámara

- Al crear el proyecto en modo 2D, la cámara ya estará en modo ortográfico, pero aún tenemos que ajustar el tamaño y otros parámetros
- Selecciona la cámara (Main Camera)
  - Se puede ajustar el color de fondo
  - La proyección debe ser ortográfica
  - Size es la mitad de la altura del volumen de la cámara
  - Puedes dejar la cámara en 0, 0, -10

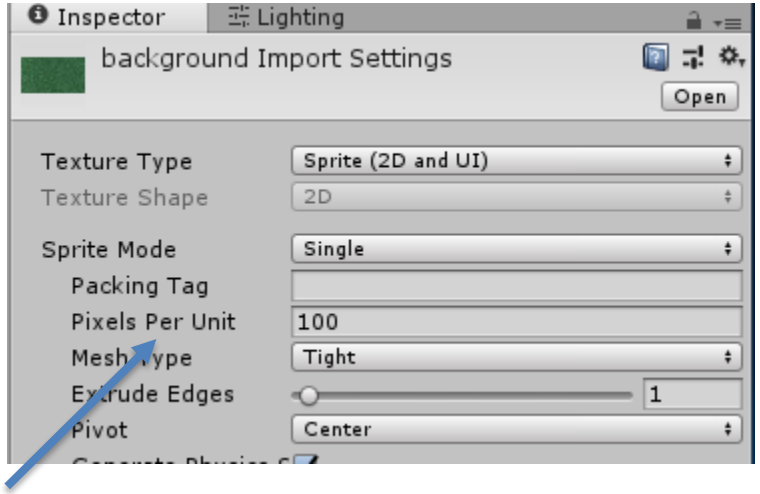


Ajustando la cámara

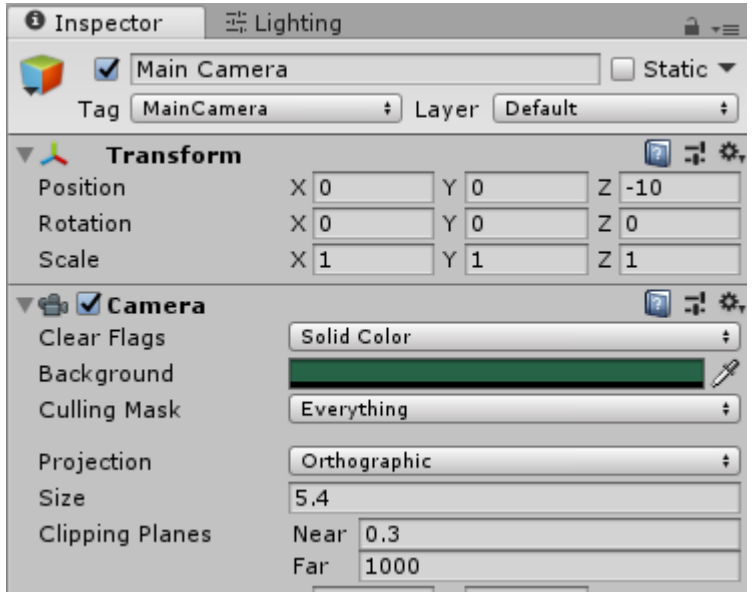
- Queremos ajustar la cámara de tal forma que la imagen de fondo ocupe toda la ventana



Al importar un sprite…



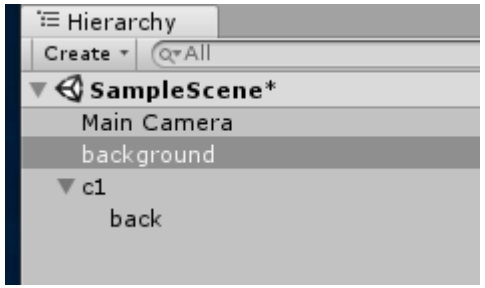
…especificamos la conversión entre píxeles y unidades de Unity. En este caso, la imagen tiene 1920x1080 píxeles, por lo que mide 19.2x10.8 unidades

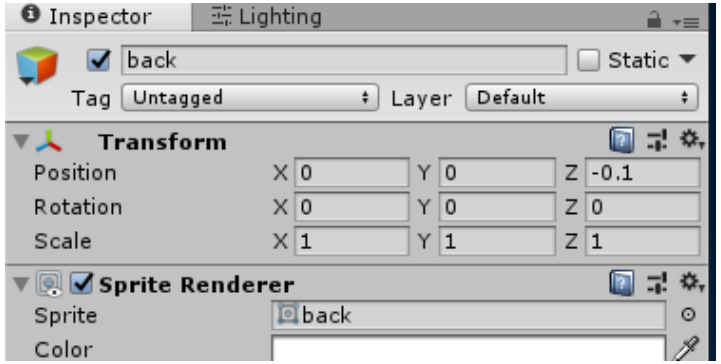


Por lo tanto, 10.8/2=5.4 hace que la cámara se ajuste a la altura de la imagen

Configurando las cartas

- Vamos a construir la carta, con la funcionalidad necesaria para que responda al usuario
- Arrastra una carta (la cara frontal) sobre la escena
  - Ajusta su tamaño para quede bien en la ventana (por ejemplo, ajustando los píxeles por unidad)
- Arrastra la parte trasera de la carta sobre la anterior
  - Haz que sea hija de la frontal en la jerarquía
  - Sitúala en la posición 0, 0, -0.1, para que quede por encima de la anterior





Configurando las cartas

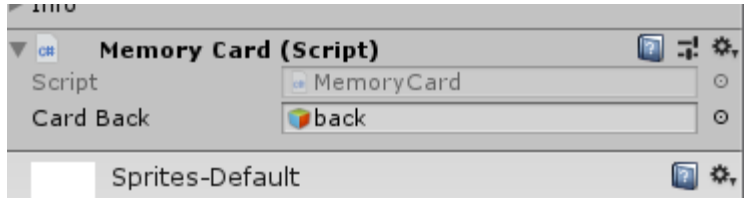
- La carta debe responder a los clics del usuario
  - Para ello, necesita un collider:
    - Selecciona la cara frontal
    - Añade un componente
    - Selecciona Box Collider 2D (¡no Box Collider!)
  - Crea un script llamado MemoryCard.cs
    - Añádelo a la cara frontal y prueba que funciona

```
public class MemoryCard : MonoBehaviour {
  public void OnMouseDown() {
    Debug.Log("clic");
  }
}
```

Configurando las cartas

- Vamos a hacer que la parte trasera de la carta desaparezca/aparezca con el clic:

```
public class MemoryCard : MonoBehaviour {
  [SerializeField] private GameObject cardBack;

  public void OnMouseDown() {
    if (cardBack.activeSelf)
      cardBack.SetActive(false);
    else
      cardBack.SetActive(true);
  }
}
```



- En el editor, asocia la parte trasera de la carta que está en la escena con la referencia cardBack del script

Configurando las cartas

- Podemos cambiar el sprite asociado a un GameObject cambiando la propiedad Sprite del componente Sprite Renderer:

```
GetComponent<SpriteRenderer>().sprite = image;
```

- Así será posible crear cartas por código en tiempo de ejecución
- Sólo hará falta un prefab carta, que podremos instanciar y cambiar la cara frontal
- Crea un prefab y borra la carta de la escena

Configurando las cartas

- Vamos a crear un controlador de escena para gestionar la creación de cartas
- Crea un GameObject vacío (llamado `GameController`) y asóciale el siguiente script:

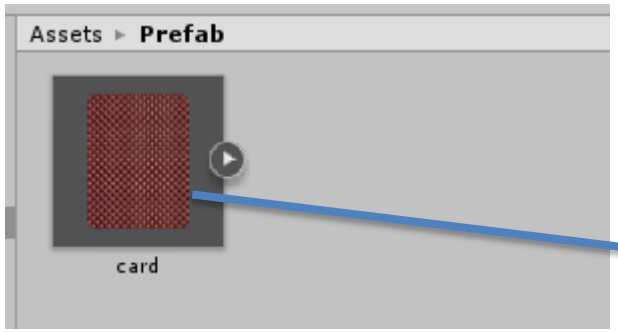
```
public class SceneController : MonoBehaviour {
  [SerializeField] private MemoryCard originalCard;
  [SerializeField] private Sprite[] images;

  void Start () {
    int id = Random.Range(0, images.Length);
    originalCard.SetCard(id, images[id]);
  }
}
```

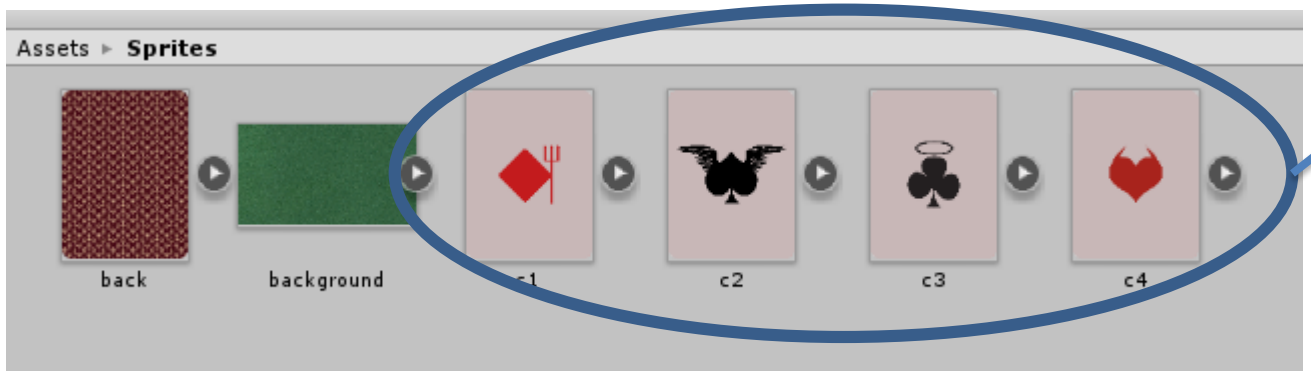
- Y añade a MemoryCard.cs:

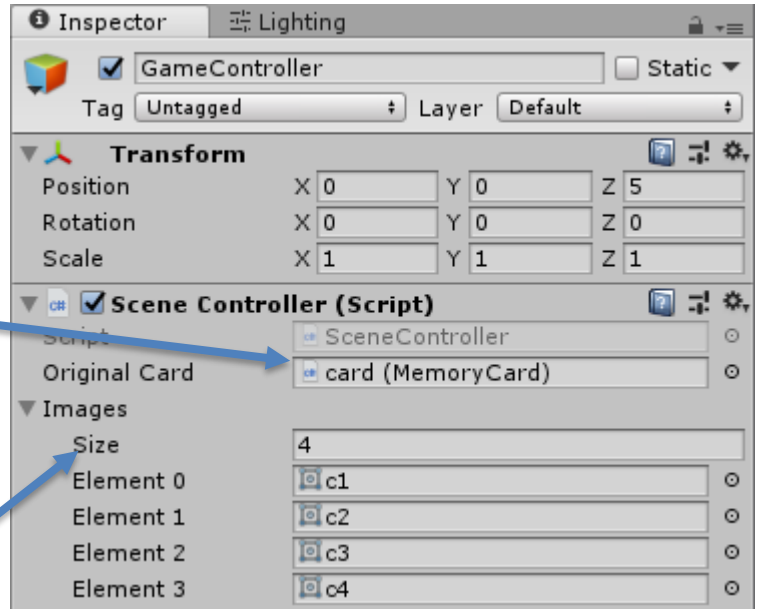
```
private int _id;

public int id {
  get { return _id; }
}

public void SetCard(int id, Sprite image) {
  _id = id;
  GetComponent<SpriteRenderer>().sprite = image;
}
```


Configurando las cartas

- Asocia los GameObjects a las propiedades de Scene Controller:

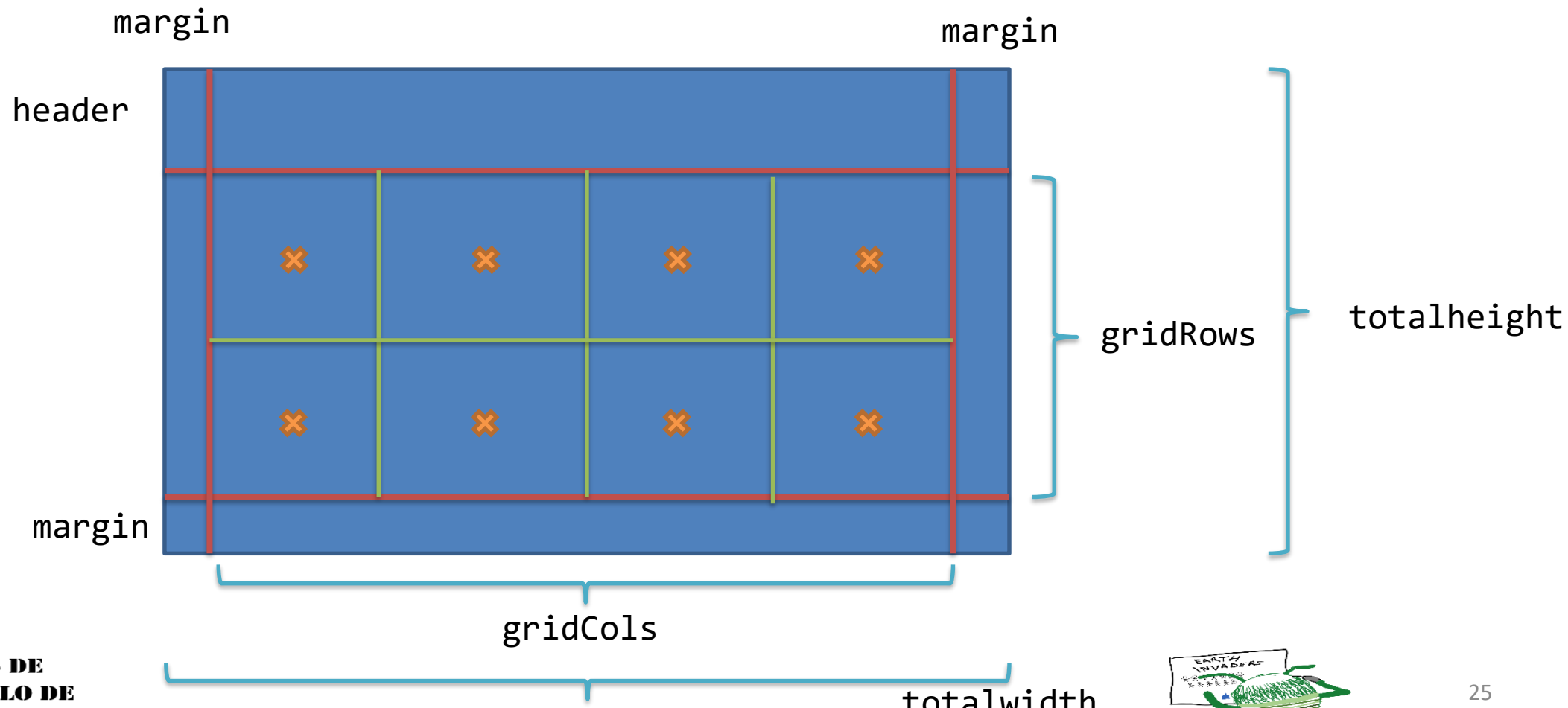






Configurando las cartas

- Vamos a hacer que el controlador reparta las cartas en la mesa automáticamente:



Configurando las cartas

```
public int gridRows = 2;
public int gridCols = 4;
public float header = 2f;
public float margin = 1f;

void Start () {
  float totalheight = Camera.main.orthographicSize * 2;
  float totalwidth = totalheight * Camera.main.aspect;
  float cardWidth = (totalwidth - 2 * margin) / gridCols;
  float cardHeight = (totalheight - header - margin) / gridRows;
  for (int j = 0; j < gridRows; j++) {
    for (int i =0; i < gridCols; i++) {
      MemoryCard card = Instantiate(originalCard) as MemoryCard;
      int id = Random.Range(0, images.Length);
      card.SetCard(id, images[id]);
      float posX = (i + 0.5f) * cardWidth - totalwidth / 2 + margin;
      float posY = (j + 0.5f) * cardHeight - totalheight / 2 + margin;
      card.transform.position = new Vector3(posX, posY, originalCard.transform.position.z);
    }
  }
}
```


Configurando las cartas

- El código anterior no asegura que sólo haya parejas en el tablero. Añade al método Start del controlador el siguiente código:

```
int[] ids = new int[gridRows * gridCols];
for (int i = 0; i < ids.Length; i++) {
  ids[i] = i / 2;
}
ids = ShuffleArray(ids);

…
int index = j * gridCols + i;
int id = ids[index];
card.SetCard(id, images[id]);
```

- Y define el método `ShuffleArray`

```
private int[] ShuffleArray(int[] numbers) {
  int[] newArray = numbers.Clone() as int[];
  for (int i = 0; i < newArray.Length - 1; i++) {
    int r = Random.Range(i, newArray.Length);
    int tmp = newArray[i];
    newArray[i] = newArray[r];
    newArray[r] = tmp;
  }
  return newArray;
}
```


Lógica de las parejas

- Para implementar la lógica de las parejas vamos a implementar las siguientes funciones:

SceneController:

```
MemoryCard _firstRevealed, _secondRevealed; // cartas boca arriba
bool canReveal; // ¿se puede girar otra carta?
void CardRevealed(MemoryCard c) // se ha girado la carta c
```

MemoryCard:

```
void OnMouseDown() // si estoy boca abajo y se puede girar una carta,
                   // girar y notificar

public void Unreveal()  // volver a poner la carta boca abajo
```

Lógica de las parejas

- SceneController:

```
private MemoryCard _firstRevealed;
private MemoryCard _secondRevealed;

public bool canReveal {
  get { return _secondRevealed == null; }
}

public void CardRevealed(MemoryCard card) {
  if (_firstRevealed == null)
    _firstRevealed = card;
  else {
    _secondRevealed = card;
    Debug.Log(_firstRevealed.id == _secondRevealed.id ? "¡Pareja!" : "Inténtalo otra vez");
  }
}
```

Lógica de las parejas

- MemoryCard:

```
private SceneController controller;

public void Start() {
  controller = GameObject.Find("GameController").GetComponent<SceneController>();
}

public void OnMouseDown() {
  if (cardBack.activeSelf && controller.canReveal) {
    cardBack.SetActive(false);
    controller.CardRevealed(this);
  }
}

public void Unreveal() {
  cardBack.SetActive(true);
}
```


Lógica de las parejas

- Al dar la vuelta a dos cartas, puede ocurrir dos cosas:
  - No coinciden
    - Dejaremos pasar un tiempo, y volveremos a dejarlas boca abajo
  - Coinciden
    - Las dejaremos boca arriba, incrementaremos la puntuación, y dejaremos al jugador que siga destapando cartas
    - Si no quedan más parejas, terminar con un mensaje de felicitación

Lógica de las parejas

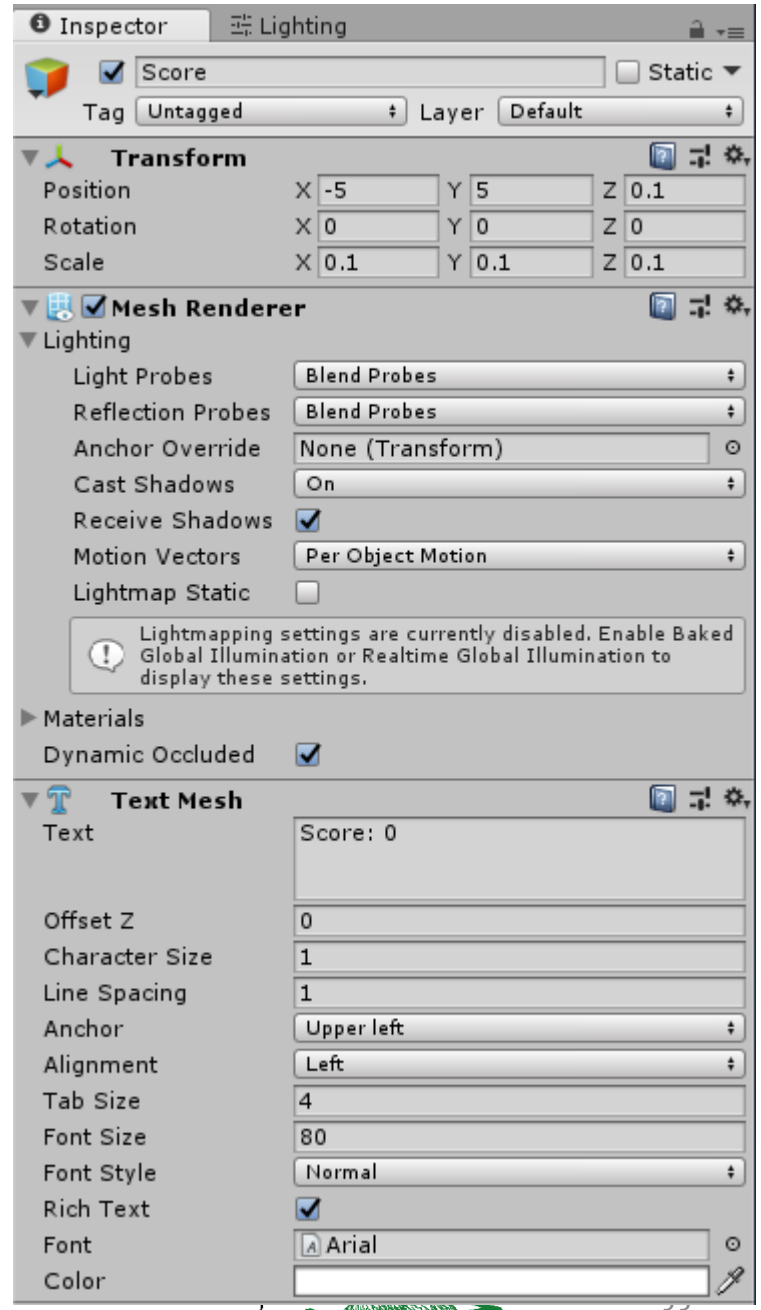
- SceneController:

```
private int _score = 0;

public void CardRevealed(MemoryCard card) {
  if (_firstRevealed == null)
    _firstRevealed = card;
  else {
    _secondRevealed = card;
    StartCoroutine(CheckMatch());
  }
}
```

```
private IEnumerator CheckMatch() {
  if (_firstRevealed.id == _secondRevealed.id) {
    _score++;
    Debug.Log("Score: " + _score);
  } else {
    yield return new WaitForSeconds(.5f);
    _firstRevealed.Unreveal();
    _secondRevealed.Unreveal();
  }
  _firstRevealed = _secondRevealed = null;
}
```

Implementando el marcador

- Vamos a implementar una interfaz de usuario rudimentaria para mostrar el marcador y para resetear el juego
- Crea un Empty en la escena y añádele un componente Text Mesh
- Ajusta la posición del texto y establece las propiedades necesarias
- Puedes importar una fuente True Type al proyecto y usarla



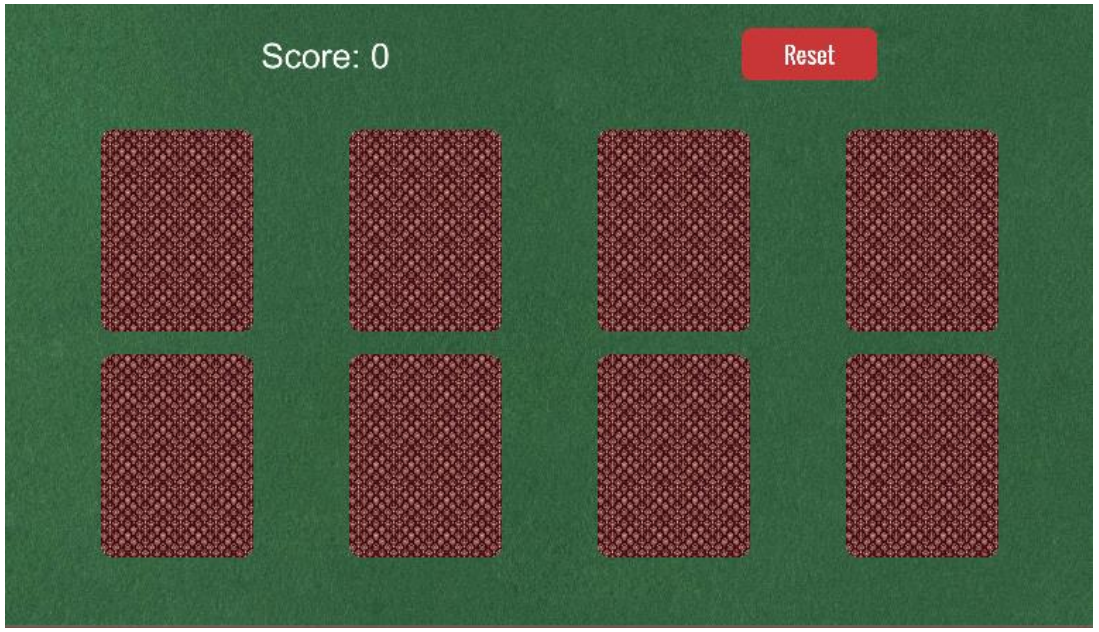
Implementando el marcador

- Para mostrar el marcador, en el controlador añadimos:

```
[SerializeField] private TextMesh scoreLabel;

private IEnumerator CheckMatch() {
  if (_firstRevealed.id == _secondRevealed.id)
  {
     _score++;
     scoreLabel.text = "Score: " + _score;
  } else
     …
```

Añadiendo la opción de reiniciar el juego

- Arrastra el botón de Reset a la escena
- Sitúalo en la parte superior de la ventana y ajusta su tamaño
- Acuérdate de situar el botón más cerca que el fondo con respecto a la cámara
- Añádele un Box Collider 2D para poder responder a los clicks



Añadiendo la opción de reiniciar el juego

- Crea un script para el botón:

```
public class UIButton : MonoBehaviour {
  [SerializeField] private GameObject targetObject;
  [SerializeField] private string targetMessage;
  public Color highlightColor = new Color(.8f, .8f, .8f);
  private Vector3 originalScale;

  private void OnMouseEnter() {
    SpriteRenderer sprite = GetComponent<SpriteRenderer>();
    if (sprite != null)
      sprite.color = highlightColor;
  }
  private void OnMouseExit() {
    SpriteRenderer sprite = GetComponent<SpriteRenderer>();
    if (sprite != null)
      sprite.color = Color.white;
  }
```

Nos permitirá avisar a otro objeto cuando se pulse el botón

Añadiendo la opción de reiniciar el juego

```
  private void OnMouseDown()
  {
    originalScale = transform.localScale;
    transform.localScale = originalScale * 0.9f;
  }

  private void OnMouseUp()
  {
    transform.localScale = originalScale;
    if (targetObject != null)
    {
      targetObject.SendMessage(targetMessage);
    }
  }
}
```

GameObject.SendMessage

Leave feedback

SWITCH TO MANUAL

public void **SendMessage**(string **methodName**, object **value** = null, SendMessageOptions **options** = SendMessageOptions.RequireReceiver);

Parameters

methodName	The name of the method to call.
value	An optional parameter value to pass to the called method.
options	Should an error be raised if the method doesn't exist on the target object?

Description

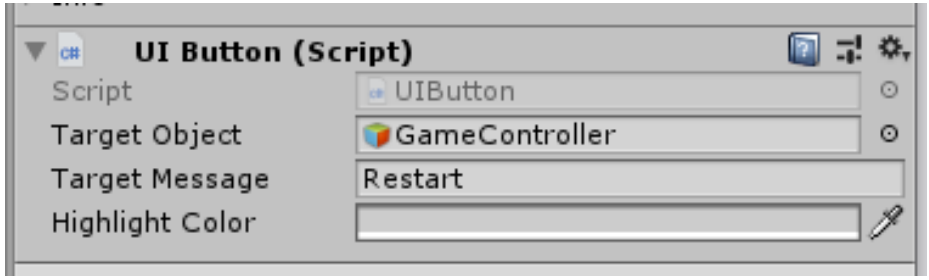
Calls the method named `methodName` on every MonoBehaviour in this game object.

Añadiendo la opción de reiniciar el juego

- Añade la función `Restart` a `SceneController`:

```
using UnityEngine.SceneManagement;
public void Restart() {
  SceneManager.LoadScene("SampleScene");
}
```

- Conecta el campo Target Object con GameController y haz que se llame a Restart con la pulsación del botón

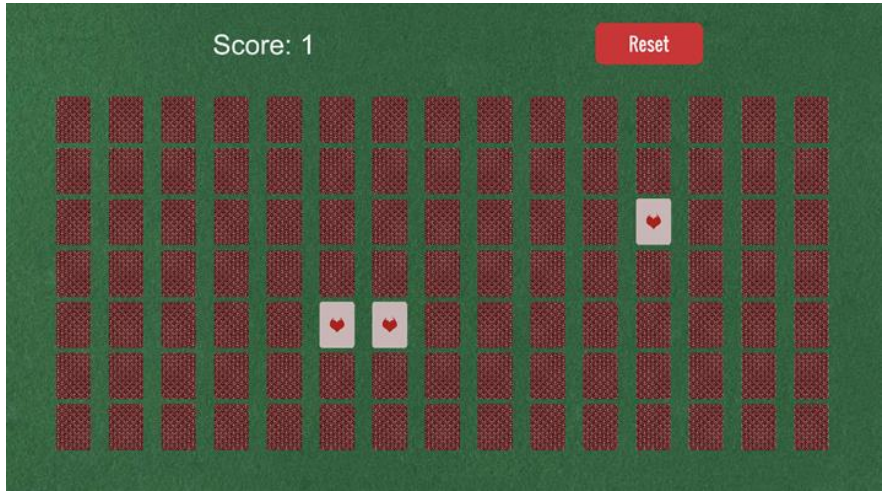


Añadiendo la opción de reiniciar el juego

- La función `SceneManager.LoadScene`:
  - Carga la escena almacenada en el fichero cuyo nombre se le pasa por parámetro
  - Todos las instancias cargadas en memoria (gameobjects, scripts, etc.) se destruyen
  - El método `DontDestroyOnLoad()` permite especificar que un objeto no se debe destruir al cargar otra escena (por ejemplo, para llevar el marcador en un juego con varias fases)

Ejercicios

- Añade algún diseño de carta adicional
- Complica el juego haciendo que el jugador tenga que formar tríos en vez de parejas
- Modifica el controlador de escena para que ajuste el escalado de las cartas dependiendo del número de filas y columnas
- Crea una nueva escena con la pantalla final del juego y muéstrala cuando el jugador descubra todas las parejas



Bibliografía

- Joseph Hocking. Unity in Action. 2nd edition. Manning, 2018.
  - Capítulo 6
- Assets:
  - https://openclipart.org/detail/169165/card-suits-angelic-or-devilish