

Informe de proyecto

Desarrollo de juego inicial

Luis Garcia | Unity Project | 2019/06/03

# Introduccion

Este documento describe el plan de proyecto de desarrollo de juegos del equipo de Pardal Developers. Tiene como objetivo describir claramente el plan de trabajo a seguir, así como también explicar de forma clara y concisa el código fuente de la aplicación y los diseños utilizados a la hora de la implementación.

Este documento servirá como guía para la implementación de los siguientes proyectos que se lleven a cabo bajo esta metodología.

## Indice

Temario del documento

1. Introducción a la visión y objetivos del proyecto
2. Introducción a las herramientas utilizadas
3. Diseño inicial de la aplicación
   1. Diagrama conceptual de app
4. Diagrama de actividades
5. Implementación
   1. Scripts definidos

# Introducción a la visión y objetivos del proyecto.

La idea de este proyecto es ganar confianza en el desarrollo de juegos de video, logrando integrar el equipo, evaluar performance numerando las tareas y retos que se presenten durante el ciclo de diseño, implementación y publicación de proyecto.

Se tomará nota del listado de tareas, se ponderará sobre el proyecto completo y se evaluaran costos y diversas maneras de implementar o llegar al mismo objetivo.

Como objetivo principal del proyecto, se busca:

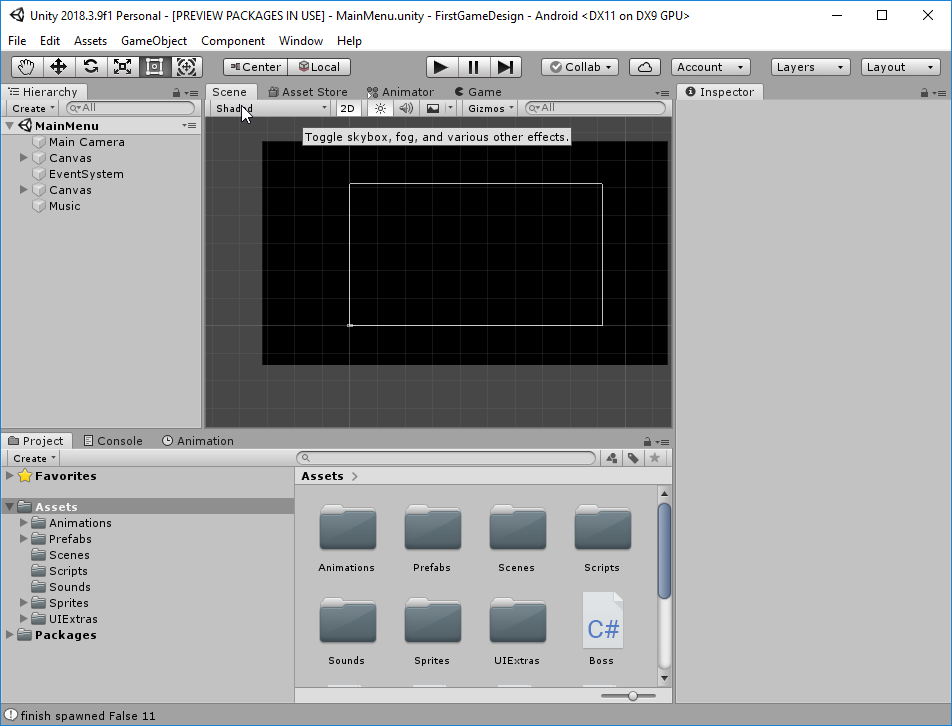
* Implementar un juego 2D en perspectiva TOP-DOWN con sprites simples con un solo lado.
* Completar la dinámica del personaje
* Completar dinámica de MOBs
* Diseñar método de spawneo de MOBs
* Diseñar distintos patrones de movimiento para MOBs
* Diseñar sistema de UI básico
* Publicar

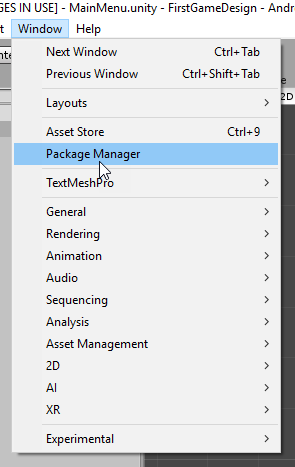
Adicionales opcionales:

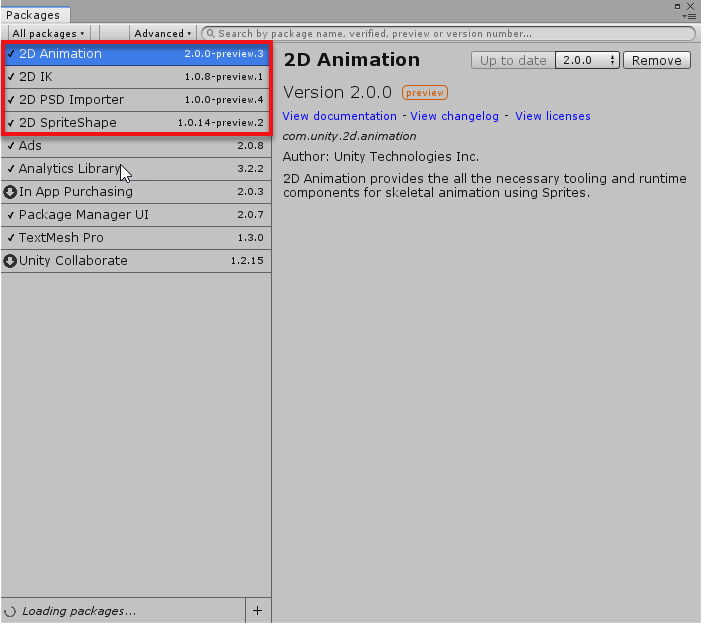
* Diseñar Boss
* Diseñar metodología de movimientos
* Diseñar drop y pickup system
* Implementar swap de escenas
* Implementar sonidos
* Implementar música
* Impementar mapa de juego
* Implementar historia
* Implementar secuencia de escenas con historia
* Implementar inventario
* Implementar score
* Implementar integración con Facebook, G+, google
* Implementar juego en red
* Implementar sistema de compra de objetos
* Implementar publicidades

# Introducción a las herramientas utilizadas.

Para el desarrollo del proyecto se utilizó Unity 2018 en su versión 3.3f1, adicionando los siguientes componentes agregados desde la sección Package Manager:







Estos componentes son:

* **2D Animation**
  + Permite obtener los valores de posicionamiento de los esqueletos (rigging) de los sprites 2D. Esto facilita la animación de los elementos.
* **2D IK**
  + Habilita el movimiento inverso de las coyunturas de lor Riggs, eso se utiliza para lograr movimientos mas fluidos a la hora de mover el esqueleto del personaje 2d
* **2D PSD Importer**
  + Permite importar archivos PSD y trabajarlos como sprite, permitiendo una rápida edición desde Photoshop.
* **2D Sprite Shape**
  + Para modelado de ambientación 2D.

Adicionalmente se utilizó:

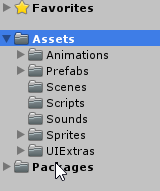
Photoshop

Audacity

Visual Studio 2019

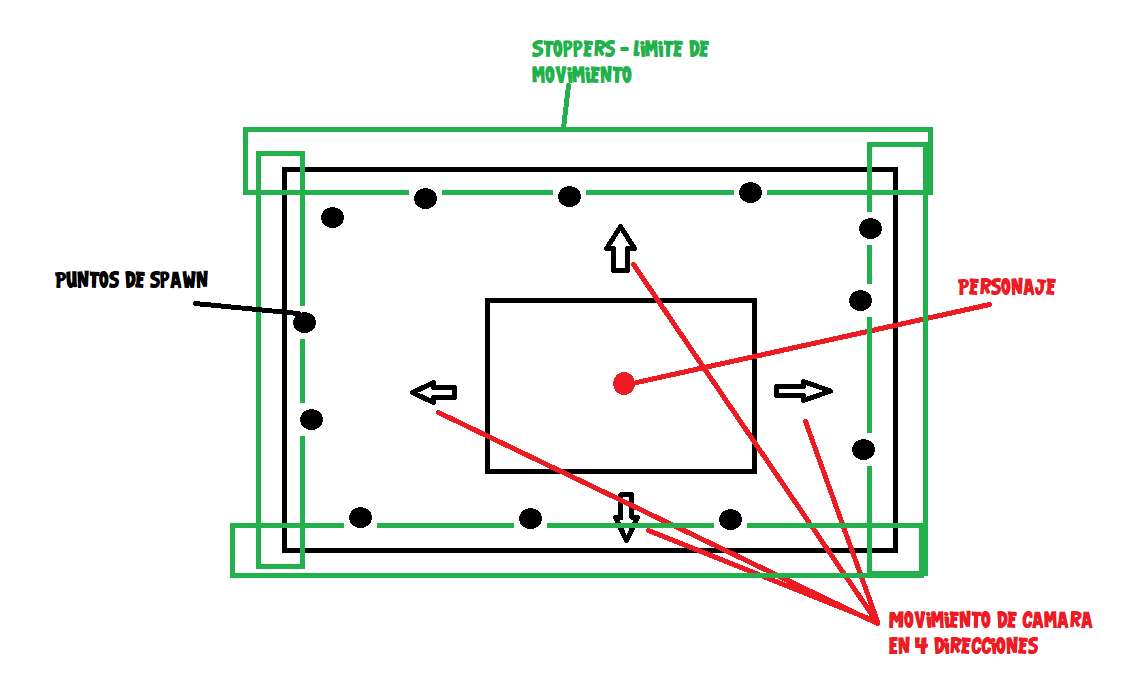
# Diseño de aplicacion.

Vista de árbol de jerarquía de componentes:



* Carpeta de Animaciones
  + Esta carpeta contendrá animaciones de los distintos componentes del juego, la nomenclatura de las animaciones será:
    - NombrePrefab\_motion.anim
    - Se puede incluir subfolder con nombre prefab, lo que no debería modificar el nombre de la animación, la cual deberá seguir siendo la mencionada anteriormente.
* Carpeta de Prefabs
  + Esta carpeta contendrá todos los elementos prefabricados y se deberá priorizar la utilización/creación de los mismos ante la creación de objetos sueltos.
* Scenes
  + Folder que contendrá todas las escenas contenidas en el juego.
  + Las mismas deberán ser secuenciadas por números identificadores
* Scripts
  + Esta carpeta contiene todos los scripts del juego, los mismos deberán siempre comenzar con letra mayúscula y referenciar el prefab, objeto y / o funcionalidad lo más clara posible
* Sounds
  + Carpeta destinada a los archivos de sonido
    - Esta carpeta deberá contener los sonidos a ejecutarse en el juego, deberán seguir la misma nomenclatura que los prefabs:
      * NombrePrefab\_Function\_Sound.extencion
* Carpeta Sprite
  + Esta carpeta contiene las imágenes ingresadas en el juego, la misma deberá organizarse en subfolders indicando el tipo de sprite que se inserta en el juego.
  + Se deberán purgar los sprites que no se utilicen luego de que la funcionalidad a implementar finaliza.
* UIExtras
  + Paquete de interfaz de usuario adicional, la cual habilita el uso del joystick
* Packages
  + Paquete importado por defecto en Unity

# Diagrama conceptual de app.



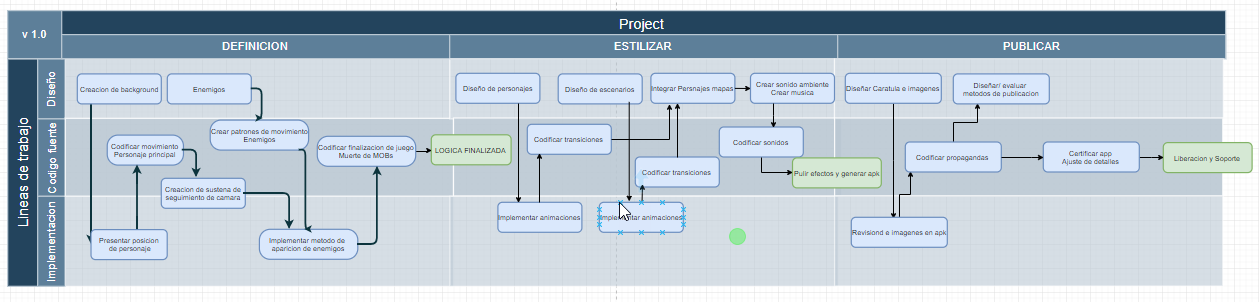
Objetivo:

* Sobrevivir oleadas de MOBs sobre un único escenario

Adicionales:

* Agregar Boss al final de cada orda
* Agregar mas de una escena de juego y en secuencia
* Agregar métricas de juego (scoring y high score)

# Diagrama de actividades



# Implementación

* Se genera escena de Game
* Se situa GameObject = Ground con tamaño mayor a la cámara
* Se situa personaje principal
* Se genera Script C# = Player

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

//Se agrega interaccion con Interfaz de usuario, para poder transiccionar la escena en caso de muerte, o bien utilizar efectos mediante paneles de colores al momento de recibir daño

public class Player : MonoBehaviour

{

//Variable editable desde el inspector, para modificar velocidad de pj

public float speed;

//Acceso a las propiedades de la física del objeto, lo necesito para mover el objeto, además se debe freezar rotación en Z y setear gravedad = 0

private Rigidbody2D rb;

//Variable que contendrá dirección y velocidad de movimiento.

private Vector2 moveAmount;

//Variable que para instanciar y acceder al joystick análogo prefab agregado en la escena.

protected Joystick joystick;

//Variable que para acceder a las variables de animación, para setear triggers y/o valores de transición de las animaciones definidas.

private Animator anim;

//Variable publica que permite definir vida desde el inspector.

public int health;

//Vector de imágenes representativas de la barra de vida del personaje.

//\*Esto podría cambiarse por una barra de vida y mana (ver script Boss)

public Image[] hearts;

//Sprite agregado desde el inspector….el cual sirve para identificar una vida

public Sprite fullHeart;

//Sprite agregado desde el inspector….el cual sirve para identificar una vida faltante

public Sprite emptyHeart;

//Slot para agregar desde el inspector el componente animator de otro objeto, en este caso se agrega un panel con efecto rojo a transparente, el cual flashea cuando te pegan

public Animator gethurt;

//Variable de tipo SceneTransition para poder llamar el método Load(“nombreEscena”) para mostrar la escena dead / lose

private SceneTransition scene;

//Funcion de inicialización de parametros

private void Start()

{

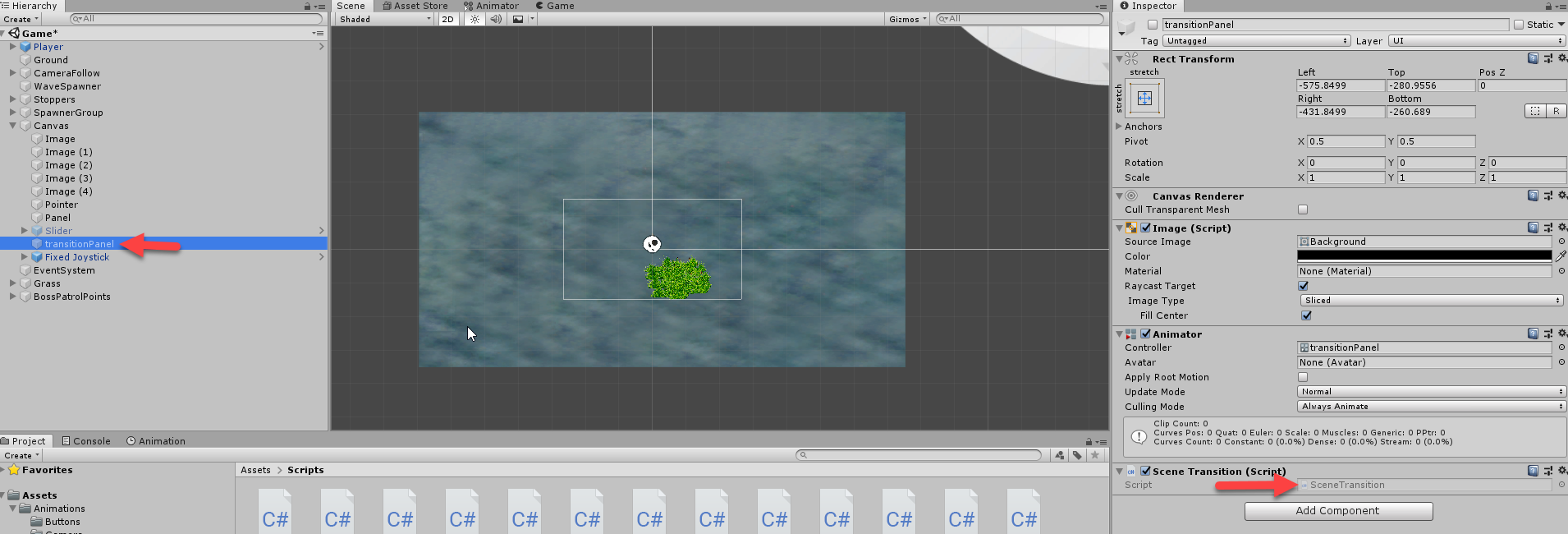
//se toman los componentes animator y rigidBody del gameObject al que se le agrego el script, en este caso el player

anim = GetComponent<Animator>();

rb = GetComponent<Rigidbody2D>();

//Se toma desde la escena el Objeto que tenga componente SceneTransition en el

scene = FindObjectOfType<SceneTransition>();



//En la escena existe un panel que tiene dicho compoenente. Este script o componente, permite el cambio de escena utilizando una animación de transición de fade out y fade in disponibles en el objeto transitionPanel del Canvas.

//Se inicializa el Joystick.

joystick = FindObjectOfType<Joystick>();

}

//Funcion ejecutada en cada frame  
 private void Update(){

//Se calcula la dirección de movimiento, ya sea por axis (asdw o flechas) o por joystick

Vector2 moveImput = new Vector2(Input.GetAxisRaw("Horizontal")+ joystick.Horizontal, Input.GetAxisRaw("Vertical") + joystick.Vertical);

//Se calcula desplazamiento en función de dirección y velocidad del gameObject que tenga este script attachedo (El Player)

moveAmount = moveImput.normalized \* speed;

//Chequeo si se está moviendo y como se esta moviendo….con el shift se corre (funcionalidad no en uso)

if (moveImput != Vector2.zero)

{

if (Input.GetKeyDown("left shift")) {

anim.SetBool("isWalking", true);

anim.SetBool("isAlsoRunning", true);

}

else {

anim.SetBool("isWalking", true);

anim.SetBool("isAlsoRunning", false);

}

}

else {

anim.SetBool("isWalking", false);

anim.SetBool("isAlsoRunning", false);

}

}

//Funcion ejecutada en cada frame pero a tiempo fijo….evita saltos y fija frame rate en función de hardware, esto hace que el game corra igual en todos lados.

private void FixedUpdate()

{

//Mover el objeto con las variables calculadas antes, pero fijo al tiempo de procesador

rb.MovePosition(rb.position + moveAmount \* Time.fixedDeltaTime);

}

//función publica que se puede llamar en una colision por ejemplo….usando collider.takeDamage(10) para dañar en 10 al Player

public void takeDamage(int damageAmount)

{

gethurt.GetComponent<Animator>().SetTrigger("hurt");

health -= damageAmount;

UpdateHealthUI(health);

if (health <= 0)

{

//Si la vida baja a 0, se muere, se destruye el objeto de la escena y se carga la escena “lose”

Destroy(gameObject);

scene.LoadScene("lose");

}

}

//Funcion para el cambio de arma, recibe un arma (GameObject que tenga tag = NewWeapon en layer NewWeapon y con script WeaponScript attached) por parámetro.

public void ChangeWeapon(WeaponScript weapontoEquip) {

//Se obtiene la posición donde se encuentra la Weapon actual

Transform weaponPlace = GameObject.FindGameObjectWithTag("Weapon").transform;

//Se destruye el arma actual

Destroy(GameObject.FindGameObjectWithTag("Weapon"));

//Se le setea el tag Weapon a el NewWeapon recibido por parametro

weapontoEquip.tag = "Weapon";

//Se le cambia la escala

weapontoEquip.transform.localScale += new Vector3(0.7F, 0.7F, 0);

//Se instancia la nueva arma donde estaba la vieja

Instantiate(weapontoEquip, weaponPlace.position, weaponPlace.rotation, transform);

}

//Funcion interna para actualizar la barra de vida en función de la variable currentHealth recibida por parámetro, utiliza los corazones para mostrar el estado de la barra. Cabe destacar que el tamaño máximo de hearts es el tamaño máximo de las imágenes agregadas en el canvas en este caso 5.

void UpdateHealthUI(int currentHealth) {

//Basicamente recorro el vector de corazones y me fijo si la variable i es menor que el valor de HP actual, si lo es, entonces eso quiere decir que desde ahí….en adelante hasta heats.length….todos los sprites deben ser corazones vacios

for (int i = 0; i < hearts.Length; i++) {

if (i < currentHealth)

{

hearts[i].sprite = fullHeart;

}

else {

hearts[i].sprite = emptyHeart;

}

}

}

//Funcion para curar, recibe por parámetro el monto de curación recibido….recordemos que el tope en este caso es 5, dado que nuestro vector de corazones = 5.

public void Heal(int healAmount) {

if (health + healAmount > 5)

{

health = 5;

}

else {

health += healAmount;

}

//Post curación, se llama al método de recalculo de imágenes en barra de vida.

UpdateHealthUI(health);

}

}

* Se genera Script C# = CameraFollow

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

// Este script se agrega a la camara

public class CameraFollow : MonoBehaviour

{

// Se pone variable publica, para poder agregar el Player a seguir desde el inspector

public Transform playerTransform;

// Velocidad de movimiento

public float speed;

// Moviendo la cámara a los bordes del escenario, podemos obtener esto valores mencionado a continuación. Este será el rango de movimiento de la cámara en dicho escenario.

public float minX;

public float maxX;

public float minY;

public float maxY;

// Start is called before the first frame update

void Start()

{

// Se inicializa el Player

transform.position = playerTransform.position;

}

// Update is called once per frame

void Update()

{

// Si el player existe, calculo el movimiento y llevo la camara a ese lugar.

if (playerTransform != null) {

float clampedX = Mathf.Clamp(playerTransform.position.x, minX, maxX);

float clampedY = Mathf.Clamp(playerTransform.position.y, minY, maxY);

transform.position = Vector2.Lerp(transform.position, new Vector2(clampedX, clampedY), speed);

}

}

}

* Se genera Script C# = Enemy