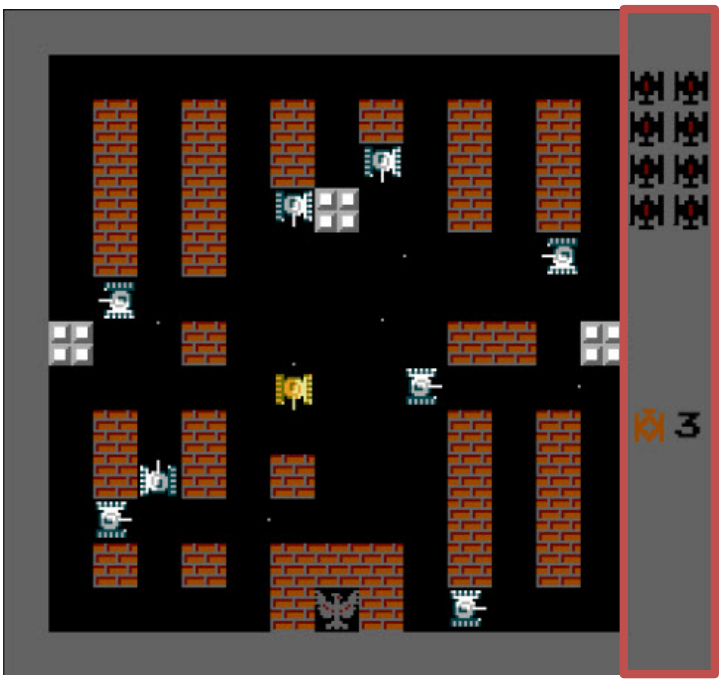
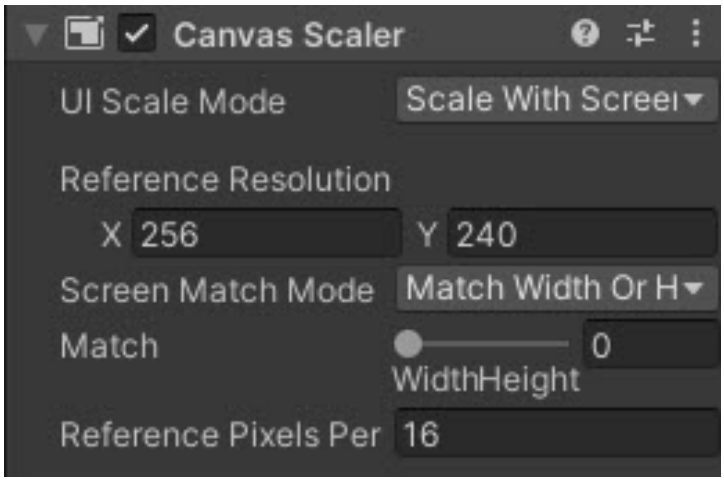
Parte de la UI

Parte del *canvas*



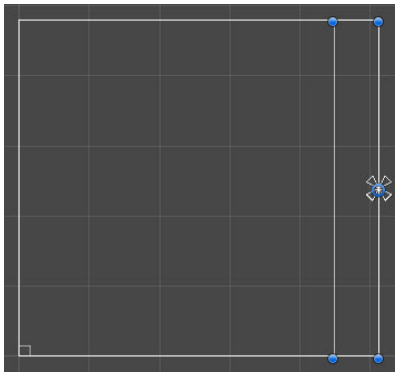
Creación del *canvas*

- Configuramos que el *canvas* escale con respecto a una resolución de referencia (nuestros 16*16 x 16*15)
- Y establecemos como referencia los 16 píxeles por unidad



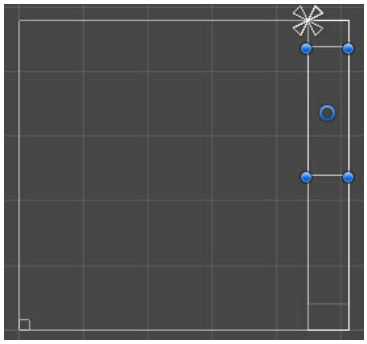
InfoPanel

- Creamos un objeto vacío como hijo del *canvas*
  - Anclas, posición y pivote *middle right*
  - 32 de ancho y 240 de alto
- Le añadimos un componente de *vertical layout*
  - Control de anchura y altura de hijos
  - *Padding* superior e inferior de 20



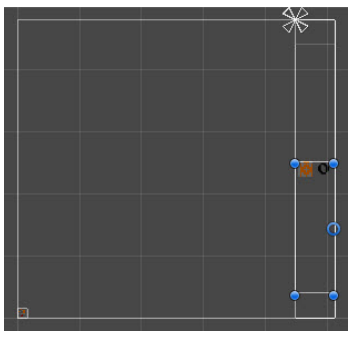
Elementos hijos de *InfoPanel*

- Creamos dos objetos vacíos como hijos del elemento anterior
  - *EnemiesPanel* y *LivesPanel*
- Al primero le añadimos un componente con disposición en rejilla
  - Tamaño de celda 14 x 14
  - 2 de espaciado en X y 1 en Y
- Al 2º le añadimos un componente con disposición horizontal
  - 2 de *padding* a izquierda y a derecha y 2 de espaciado
  - Control de anchura de los hijos



Elementos hijos de *LivesPanel*

- Creamos dos objetos hijos de *LivesPanel*
  - Un objeto *Image* de la UI: *LifeIcon*
  - Y un objeto *Text* de la UI: *LivesText*
- A ambos les establecemos 12 de alto
- *LifeIcon*
  - Le asociamos el *sprite* de vida a la imagen
- *LivesText*
  - Usamos la fuente *ArcadeClassic*
  - Alineación central tanto vertical como horizontal
  - Color negro y 18 de tamaño



EnemiesPanel

- La generación de enemigos en esta parte de la UI la haremos por código
- Necesitamos un *prefab* de icono de enemigo para la UI
  - Creamos un elemento imagen de la UI en el *Canvas*
  - Le asociamos el *sprite* del icono de enemigo
  - Lo hacemos *prefab* y lo llamamos *EnemyIcon*
- Y saber cuántos enemigos hay en cada escena
  - Podríamos confiar en el diseñador
  - Pero es mejor encargarnos de asegurarnos por código del número de enemigos que hay en cada escena

Llevar la cuenta del nº de enemigos

- El *GameManager* recibe un aviso cuando un enemigo es destruido, pero no cuando se crea
- Necesitamos llevar la cuenta en el GM
  - int enemiesInLevel = 0;
  - Método público AddEnemy () para incrementarla en 1
  - EnemyDestroyed (int points) la decrementará en 1
- ¿Quién debería informar al GM de estos sucesos?
  - Hasta ahora Damageable avisaba de la destrucción
  - Pero parece más razonable cambiar esta responsabilidad a un nuevo componente Enemy

Enemy

- Creamos un nuevo *script* que asociaremos al *prefab* *Enemy*
  - Al crearse un enemigo llama al *GameManager* para que añada un nuevo enemigo a la cuenta
  - Al destruirse un enemigo llama al *GameManager* para informarle
    - Éste restará un enemigo de la cuenta
    - Y gestionará los puntos correspondientes a su destrucción
  - Esto implica quitarle esa responsabilidad a Damageable
    - Dejará de tener la información de puntos
    - Y se limitará a destruir al enemigo, dejando la responsabilidad de informar al *GameManager* a este nuevo componente

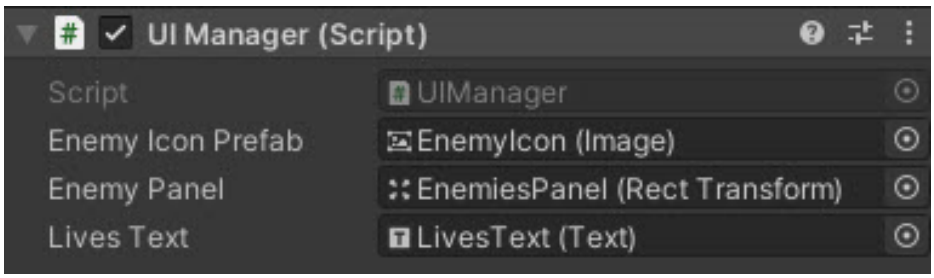
Componentes del *prefab* Enemy

UIManager

- Para que el *canvas* diseñado reciba la información, necesitamos añadirle un nuevo componente
- Añadimos al *canvas* un componente UIManager cuya implementación de partida será la explicada en clase
  - Nuestro GameManager necesitará incluir el método público que permita al UIM presentarse
- Además, necesitamos añadir al UIManager
  - Variables configurables desde el editor
  - Y métodos para ser informado por parte del GameManager de los cambios a mostrar en los momentos pertinentes

UIManager

- Debemos configurar desde el editor lo necesario para que pueda mostrar la información prevista
  - *Prefab* del icono de enemigo
  - Panel de enemigos de la UI
  - Texto de vidas de la UI



UIManager

- Y añadir al UIManager los métodos públicos que permiten al GameManager comunicarse con él
  - Init (int numLives, int numEnemies)
    Al cargar la escena, muestra el número de vidas restantes y crea el número de enemigos de la escena en el panel de enemigos
    - Bucle for que crea numEnemies imágenes, copias del *prefab* del icono de enemigo, como hijas del panel de enemigos de la UI
  - UpdateLives (int numLives) muestra el número de vidas especificado en el texto de vidas de la UI
  - RemoveEnemy () desactiva la última imagen de enemigo del panel de la UI

RemoveEnemy ()

- Desactiva el último hijo del panel de enemigos de la UI

```
public void RemoveEnemy () {
    if (enemiesLeft > 0) {
        if (enemyPanel != null)
            enemyPanel.GetChild (enemiesLeft).gameObject.SetActive (false);
        enemiesLeft--;
    }
}
```

De lo hecho hasta ahora

Resumen

Información resumida

- Por cada *script*, mostramos
  - Nombre exacto del *script*
  - Únicas variables configurables desde el editor
    - Tipo
    - Nombre exacto de la variable
  - Métodos públicos
    - Tipo resultado
    - Parámetros
    - Nombre exacto del método

Resumen

- **PlayerController**
  - float velocityScale
- **Bullet**
  - float velocityScale
- **Shooter**
  - Bullet bulletPrefab
  - float coolingDownSecs
  - bool autoShoot
  - float shootCadenceSecs
  - void Shoot ()
- **DestroyOnCollision**
- **Destructible**
- **Damageable**
  - int maxDamage
  - void MakeDamage ()
- **HeadQuarter**
  - Sprite destroyedSprite
- **Enemy**
  - int points

Resumen

- **GameManager**
  - bool PlayerDestroyed ()
  - void EnemyDestroyed (int points)
  - void FinishLevel (bool playerWon)
  - void ChangeScene (string sceneName)
  - void AddEnemy ()
  - void SetUIManager (UIManager uim)
- **UIManager**
  - Image enemyIconPrefab
  - RectTransform enemyPanel
  - Text livesText
  - void Init (int numLives, int numEnemies)
  - void UpdateLives (int numLives)
  - void RemoveEnemy ()