El proyecto desde el punto de vista de los ficheros está dividido en las siguientes carpetas, incluidas en el directorio “Assets”:

* **PhysicsMaterials**: contiene el material usado por la bola.
* **Plugins**: tiene los archivos correspondientes al plugin “UnityAds”.
* **Prefabs**: tiene el prefab utilizado por las pelotas.
* **Resources**: contiene una carpeta con los mapas en fichero de texto de los niveles y otra carpeta con los sprites utilizados en el juego.
* **Scenes**: esta carpeta contiene las tres escenas de la aplicación: SelectLevel, Game y Shop.
* **Scripts**: carpeta con los distintos scripts del proyecto, subdividida en:
* **Ball**:

1. **Ball.cs**: componente usado por cada bola del juego para moverse, moverse a una posición, pararse y continuar con su última velocidad. Para lanzar bolas por parte del BallSpawn se llama al método “StartMoving” de la pelota instanciada. Cuando una bola llega a la DeathZone se para su velocidad “Stop” y se establece, en el LevelManager-> “BallIntoDeathZone”: se llama al “GoTo” de la bola, que la bola vaya hacia la posición del BallStacker.
2. **BallSpawn.cs**: componente usado por el objeto que lanza bolas, para ello, cuenta con un método “SpawnBall” para lanzar las pelotas, llamado desde el AimController cuando es posible disparar, también en ese momento en el AimController se hace visible.
3. **BallStacker.cs** (BallSink): este componente lo tiene el recolector de pelotas. Que hace que se muestre cuando llega la primera pelota al DeathZone -> LevelManager-> ”BallIntoDeathZone”: se le da la posición de la pelota y se hace visible el recolector -> “ProcessPlay”: se suma una bola al contador de pelotas que contiene.

* **Bricks**:

1. **Bricks.cs**: clase padre para todos los Bricks del juego. Tiene un método para bajar el brick una posición en el juego-> “Fall”, otro para ver si pueden bajar-> “canFall”. Los dos se llaman en GameField en el método “MoveBlocks” para actualizar las posiciones de los bricks. Tiene un atributo del LevelManager inicializada cuando se llama a su método “Init” en el “Init” del componente LevelManager.
2. **OneBallBrick.cs**: si es tocado el atributo de LevelManager que se encuentra en la clase padre llama al método “AddBall” y \_levelManager.ballStacker.AddBall.
3. **RayHorizontalBrick.cs**: sobrescribe el método “Fall” de la clase base, haciendo que caiga y si hay sido tocado se destruya.
4. **SolidBrick.cs**: bloque del cual se le restan toques cuando es tocado y se suman puntos cuando es destruido. Con el atributo del LevelManager se llama a los métodos para sumar puntos. “OnCollisionEnter2D”-> \_levelManager.AddPoints()…

* **GameZone**:

1. **DeathZone.cs**: cuando una pelota colisiona con ella el atributo que tiene del LevelManager llama al método “BallIntoDethZone”
2. **GameField.cs** (BoardManager):

* **Managers**:

1. **AdsManagerGame.cs:** “ShowBanner” es llamado en LevelManager cuando se abre el menu de pausa y “HideBanner” cuando se cierra. “ShowAds” se llama en LevelManager cuando se clickea en el botón para añadir puntos cuando completas un nivel.
2. AdsManagerShop.cs:
3. **AimController.cs**: se inicializa en el “Init” de LevelManager y en el FixedUpdate realiza lo perteneciente al lanzamiento de bolas.
4. **GameManager.cs**: seleccionar el mapa (“SetMapLevel”) se llama en SelectLevelManager -> “LoadLevel” método listener que tiene cada botón del nivel correspondiente, y en LevelManager para cargar el siguiente nivel cuando completas el actual-> “OnClickNextEndMenu” método asociado a un botón del canvas “WinLevel”
5. **Levelmanager.cs:** en este componente cuando una bola ha colisionado con la DeathZone el método “BallIntoDeathZone” mira si ha sido la primera bola o si es otra anterior y procesa la jugada “ProcessPlay”, es decir, añade una bola al BallStacker y si ha sido la última el GameField actualiza las posiciones de los bloques y comprueba si hay que activar los warnings, el final de la partida o nivel completo.
6. **ResizeManager.cs**: se inicializa y se llama al método “Resize” en el “Init” de LevelManager.
7. **SelectLevelManager.cs**: contiene los botone, del menú inicial, con la carga del nivel correspondiente. Los botones tienen como listener delegado el método “LoadLevel” con el nivel asociado. Se puede ir a la escena de Shop-> “OnClickShopScene” y salir de la aplicación en caso de estar en Android -> “OnClickPower”.
8. **ShopManager.cs**: gestiona los botones de la escena menú: ver anuncio, comprar power up, volver al menu anterior.
9. **UIManager.cs:** gestiona la aparición de los canvas de la escena de juego según el estado de la partida. Los métodos son llamados en el LevelManager en el lugar correspondiente. Por ejemplo, “Pause” es llamado en LevelManager->OnClickPauseMenu: uiManager.Pause(). uiManager es un atributo de la clase LevelManger. En el “Init” de LevelManager se inicializa-> uiManager.Init().

* **Tools**:
* **MapReader.cs:** en GameField se lee el mapa correspondiente para crear el tablero.
* **SaveAndLoad.cs:** se usa para guardar y cargar datos. En GameField al crear un rayo se llama al método “Save” para guardar que se ha consumido un power up. Se llama también en LevelManager para guardar las estrellas y que el nivel siguiente esta desbloqueado y en ShopManager para guardar que se ha gastado rubies al comprar un power up. Por último, también se llama en AdsManagerShop y AdsManagerGame cuando ves un anuncio y consigues rubies. El método “Load” se llama en el “Awake” de GameManager para cargar los datos que hayan sido guardados anteriormente.

Características del juego:

* **Adaptar correctamente la visualización en pantalla en función de la relación de aspecto (ancho-alto) del dispositivo**: para el menú de inicio se ha hecho mediante los anchors y el CanvasScaler del canvas en Unity. Y para la escena de juego se ha hecho por código, cogiendo la altura de la cámara y su altura en pixeles para calcular la relación de pixeles por unidades de Unity. Después se coge la altura del canvas superior e inferior para determinar la altura del espacio de juego. Y por último calculamos el nuevo tamaño de la cámara según la altura del espacio de juego en pixeles, la altura deseada del espacio de juego y la altura ortográfica anterior de la cámara.
* **Leer los mapas de ficheros de texto:** se parsea el fichero de texto con el mapa, consiguiendo enteros para el tipo y el número de toques que se guardan en una lista. Después esa lista se usa en el GameField para inicializar el tablero.
* **Incluir al menos un tipo de bloque especial, más allá de los sólidos (1. . . 6) que se deben destruir:** hemos incluido un bloque rayo en horizontal y un bloque que añade una bola. El rayo cuando le choca una pelota activa un objeto hijo alargado (el rayo) que al ser activado colisiona con los bloques de esa fila y según el método OnCollisionEnter2D que tienen los bloque se les resta un toque de sus totales. El de la bola adicional simplemente cuando es colisionado suma uno al contador de bolas del LevelManager y del BallStacker.
* **Mantener el progreso del usuario (niveles desbloqueados) de tal forma que sea difícil romperlo y modificarlo externamente accediendo a los ficheros en el dispositivo:** hecho por la serialización proporcionada por Unity. En el script SaveAndLoad.
* **Incorporar al menos un potenciador. Para ello será necesario incorporar moneda virtual, por lo que también deberá guardarse, de forma segura, la cantidad de rubíes acumulados. El modo de conseguir esos rubíes puede escogerse como se desee:** se ha incorporado un potenciador para que aparezcan bricks de rayo horizontal al comprarlos con rubies. Los rubies se consiguen en la tienda al ver anuncios.
* **Mostrar anuncios al usuario (con UnityAds) para monetizar el juego:** se pueden ver anuncios en la escena Shop. Se han hecho mediante UnityAds.

Partes opcionales:

* Banners: aparecen cuando estás en el menú de pausa.
* Permitir al usuario solicitar la visualización de anuncios a cambio de rubíes.