Flappy bird style game

* A performant infinite, endless style game by moving the world instead of the player
* Object pooling to create the illusion of many objects while using a few
* Cartoon style unrealistic 2D physics using Unity’s built-in physics system

Note that co-routines are to be avoided unless you only need to run something once, such as fade into the scene. It requires more management. It needs to be started and stop, therefore needs more code. Co-routines also need allocation of memory which leads to garbage collection and therefore performance loss at run time. It is recommended to avoid them during your repetitive gameplay code.

The **Game Controller** decides among things when to display them (Game-Over message, etc.), manages the score, tracks the state of the game and restart the level when it is game over. A Singleton Pattern is usually used for the **Game Controller**, in order to make sure that there is ever only one **Game Controller** that is active. In software engineering, the Singleton Pattern restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists or restricting the instantiation to a certain number of objects.

**Object Pooling** allows one to spawn a set of objects to be used, then reposition them once we need new ones by recycling them once they become invisible to the player. **Object Pooling** is a software creational design pattern that uses a set of initialized objects kept ready to use, rather than allocating and destroying them on demand. The client of the pool will request an object from the pool and perform operations on the returned object. When the client has finished, it returns the object to the pool rather than destroying it. It can be done automatically or manually. It significantly improves performance. It prevents framerate drops from garbage collection which cleans up memory whenever instantiating and destroying objects occurs.

However **Object Pooling** complicates object lifetime as objects obtained from or returned to a pool, are not actually created or destroyed at the this time. Therefore this require care in implementation. Objects are not just going to spawn new. You would need some kind of reset function especially if it is a complicated object, to make sure everything is reset when it goes back to the pool.