# **Zoo World Documentation**

# **Functionalities**

* Top Down Camera
* Any Graphics
* Animals
  + Every 1-2 seconds one animal appears
  + Animal moves randomly
  + Animal checks collision using physics
  + If move out of screen it changed movement direction to return back to screen
  + Animal Types
    - Frog (Prey)
      * Collisions
        + Collide with Frog → Fly apart
        + Collide with Snake → Disappear from screen
      * Movement
        + Every X seconds make a jump for fixed distance
    - Predator(Snake)
      * Collisions
        + Collide with Frog → Eat Frog → Display “Tasty”
        + Collide with Snake → One wins and the other dies (Oldest one wins)
      * Movement
        + Linear movement

## **Design Patterns**

* **Singleton**: for the main Managers, I used the Singleton Pattern, so I can easily create decoupled code and avoid “spaghetti code”.
* **Composite Pattern**: Instead of creating the classes that include several different functionalities, I opted for the Composition. With this way of designing the application, I can create new objects and make them assume specific behaviors just by adding the very specific scripts I need, just like the original components of unity.
* **Factory**: By using the ObjectSpawner I can have a list of the possible objects (In this case the animals) and instantiate them on the scene inside of a specific area and stage limits. I could
* **Pooling**: In order not to keep instantiating and destroying objects all the time, If I have enough amount of them on the screen, I just deactivate and activate the objects I need, to avoid memory peaks and improve the performance.

## **Scripts**

* Game Management
  + **GameManager**.cs:
    - Handles the dead animals counters
    - Handles spawning groups
    - Can handle main Game Logic
* UI
  + **UIManager**.cs:
    - Update dead counters on Canvas
    - Display the “Tasty” message when an animal dies
* Handling Animals Collision
  + **Animal**.cs
    - Define the animal type (Prey or Predator)
    - Can name the specific animal for the prefab
    - Places the animal as a child in the specific group according to type
    - Compares which animal it is colliding with to destroy preys or the weakest predators (based on spawnIndex)
* Spawning Animals
  + **ObjectSpawner**.cs As generic as it can be, it has a list of possible prefabs and instantiates them randomly on the stage according to parameters such as (minWait and maxWait). It also uses a SpawnAreaChecker before placing the animal, this way we can ensure it won’t spawn an animal on top of the other and cause undesired behaviors. The object is instantiated in between the bounds areas that can be changed straight on the editor, making it easy for the game designers to design levels without coding.
  + **SpawnAreaChecker**.cs: used to detect if the future spawning spot has room for the new spawned animal and if it cannot spawn there, the **ObjectSpawner.cs** calculates a new position.
* Movement Types (Keeping the logic very decoupled, I can simply add those scripts to any object and just by adjusting parameters such as speed and rotation speed, it will work)
  + **LinearMovement**.cs: every directionChangeInterval seconds the object selects a new direction to move and smoothly rotates towards the new direction according to the possible directions the object can walk based on the movementDirection selected. If the object collides with the “StageBounds” it starts moving towards the center until it meets the PlayableArea again.
  + **JumpMovement**.cs: every jumpIntervalMin to jumpIntervalMax seconds the object selects a new direction to move and smoothly rotates towards the new direction according to the possible directions the object can walk based on the movementDirection selected. If the object collides with the “StageBounds” it makes a jump towards the center.
* Colliders Visualization
  + **BoxColliderGizmoDrawer**.cs: used for the invisible walls, bounds and the **SpawnAreaChecker,** to visualize how the collisions work better.