WHAT WILL BE EVALUATED

- Organisation of classes and relationships.
- Quality and readability of the code.

TASK DEFINITION

- The end result should at least compile & run on Unity 2017.4.
- Your task is to write a simulation where red and blue units (represented by a circle shape) fight each other.
- Parse the json file “data.txt” placed in a folder and store it in the memory. It contains game configuration. (Parsing should be done with an unity’s own serialization classes.)

```
{
        "GameConfig":
        {
"gameAreaWidth": 50,
"gameAreaHeight": 50,
"unitSpawnDelay": 500,
"numUnitsToSpawn": 50,
"minUnitRadius": 0.5f,
"maxUnitRadius": 1, “minUnitSpeed”:
5f, “maxUnitSpeed”: 10f, } }
```

- The battle simulation should have an area defined in the GameConfig. (gameAreaWidth, gameAreaHeight.) Draw this area with a rectangle sprite. Position camera dynamically to see whole game area.
- In the beginning of the simulation, units should be spawned one by one at random

positions in this game area. (define unit's radius by a random value between GameConfig -> minUnitRadius & maxUnitRadius) Delay between unit spawns is defined in the GameConfig. (unitSpawnDelay)

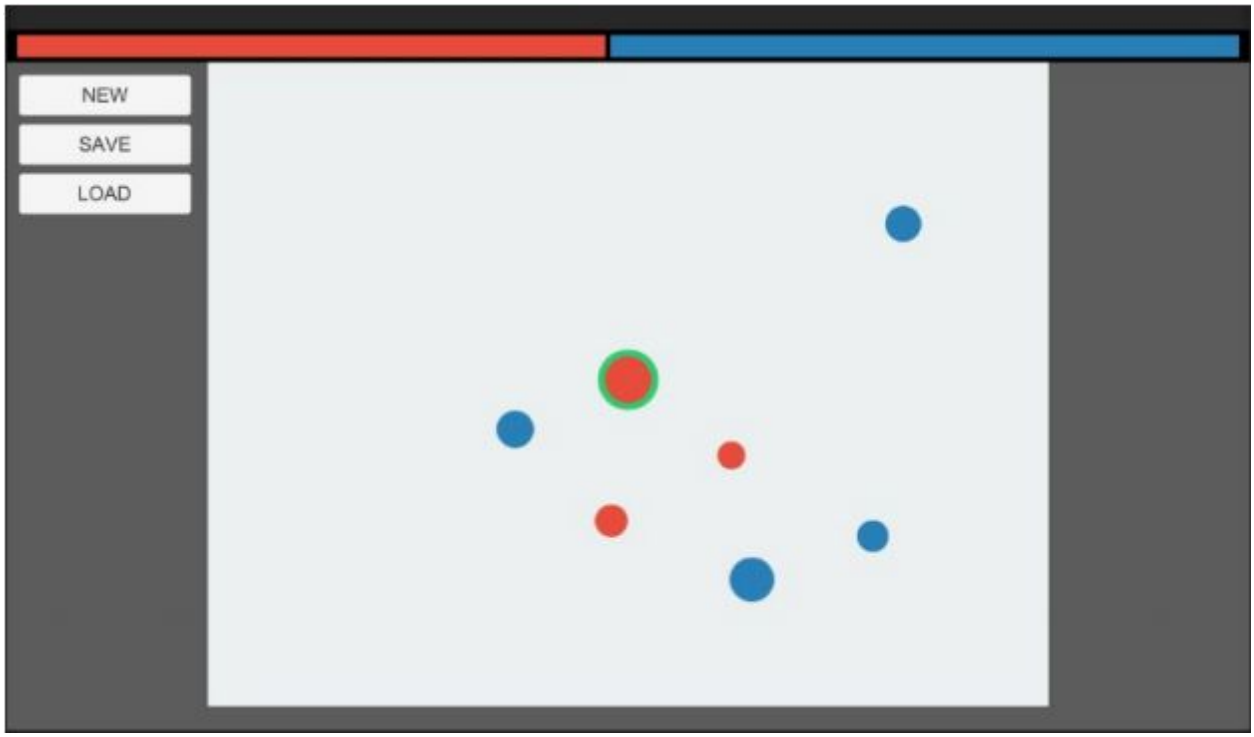
● When all the units are spawned (GameConfig -> numUnitsToSpawn), they should start moving at the same time where each unit has a different initial speed (GameConfig -> minUnitSpeed, maxUnitSpeed) and different initial random angle. (0 - 360)

● Units bounce back when they collide with the boundary of the game area.

● Units bounce back when they collide with same color units.
● When different colored units collide, both of their radius should shrink equal amount to prevent the collision. If any of the colliding units' radius become smaller than 0.2, both units should be destroyed.

● Unity physics shouldn't be used. Custom collision detection for the circles (and edge detection for bouncing back from boundary) should be enough for physics calculations.

● At the top of the screen, draw GameHud ui showing which color is winning.



● User can save and load the state of simulation any time using UI buttons. Simulation state should be saved as a json string in PlayerPrefs. ● When one of the color doesn't have any more units, show a simple popup which shows:

- ○ The winner color.
- ○ How long the simulation took.
- ○ Play again button.

● (Bonus) Implement ui to be able to change the speed of the simulation.

● Don't use any 3rd party frameworks.

● Write a short summary explaining organisation & structure of the code you've written. Explain the reasonings if you used any design patterns.