Using Unity to study properties of a two-dimensional ideal gas

V. Krstić * and I. Mekterović **
* College for Information Technologies, Zagreb, Croatia
** University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
* vladimir.krstic@vsite.hr
** igor.mekterovic@fer.hr

***Abstract* - Unity is a cross-platform game engine developed by Unity Technologies primarily used to develop 2D and 3D video games. Unity features a powerful physics engine and a GUI-based integrated development environment with support for scripting using C#. This way, game developers can easily create objects either by point-and-click operations or programmatically and assign physical properties (such as mass, initial position and velocity, and friction coefficient) to these objects, while the built-in physics engine takes care of the time evolution of the system. We believe that these features make Unity also suitable for various educational simulations and exercises. In this paper, we will use Unity and its physics engine to study properties of a two-dimensional ideal gas. In the Unity simulation, gas particles will be represented as discs with some physical properties and placed within a container. Unity's physics engine will be used to evolve the system in time, and the state of the system will be observed. Finally, we will provide an analysis of various scenarios and comparisons of observed system properties to the theoretical framework.**

Keywords* - *Unity game engine, physics engine, physics simulation, two-dimensional ideal gas

I. INTRODUCTION

Today's computer games are incredibly realistic due to powerful computers and sophisticated software used to create these games. The Unity game engine is one such software [1]. Its powerful physics engine is responsible for all calculations in the game regarding physics. Since Unity is a relatively simple, intuitive, user-friendly game engine to start with, we came to the idea of exploring the possibility of using Unity as a virtual physics laboratory in physics education. We expect that this approach might be interesting to some students and educators because it combines programming, virtual data acquisition, and subsequent data analysis.

In this paper, we present how we used Unity to study the following properties of a two-dimensional ideal gas: speed distributions, fluctuations in the number of gas particles, and equipartition of energy. These properties are probably presented in every book that discusses the kinetic theory of gases [2]. Properties like the evolution of the speed distribution are interesting enough to be studied using computer simulations even back in the '70s and '80s [3, 4]. In these computer simulations, particle trajectories are never actually calculated due to the lack of computer power. Two particles collide not because their trajectories cross but because they are randomly chosen from the set of all particles. However, in this work, we use the Unity physics engine, which calculates for us "behind the scene" the trajectories of all particles in the system, detects collisions, and works them out.

II. UNITY AS A VIRTUAL PHYSICS LABORATORY

The two-dimensional ideal gas is a problem that consists of a two-dimensional box and non-interacting gas particles (they do not attract or repel each other except when they collide, and in these collisions, momentum and energy are conserved).

This problem is easy to simulate in Unity because the Unity physics engine handles gas particles movement and collisions. One only needs to:

(a) create the container (a two-dimensional box in our case, though any other shape will do) and gas particles,
(b) set the initial state.

A. Setting the scene: container and gas particles

In Unity, objects are represented as images (e.g., png, gif, etc.) called sprites. For sprites to interact with each other, they have to be equipped with at least a collider object. For instance, the gas container can be created by joining four "wall" sprites equipped with the corresponding collider objects in a rectangle. More elegantly, an arbitrary container (e.g., ring) can be drawn in any image editing software (or even MS Word) with transparent areas set and imported in Unity, such as a png image. Unity can create and fit a (multi-polygon) collider to such images with transparent backgrounds.

In our simulation, we need only one container and this container can be manually created as described. However, we also need hundreds of gas particles and obviously, this calls for a programmatic approach. Luckily, Unity features a *Prefab* concept – a way to create an object with many features, and then bundle it as a single template object from which one can create new object instances. In our simulation, we represent gas particles as discs equipped with *Circle Collider 2D* and *RigidBody 2D* with a material friction set to 0 and bounciness set to 1. Hence, all gas particle collisions (either with particles or walls of the container) are elastic ("1 indicates a perfect bounce with no loss of energy" – Unity manual [1]). The single gas particle is created manually and stored as a *Prefab* in order to be

used as a template during runtime when gas particles are instantiated in the container at the beginning of the simulation.

B. Setting the initial state

Unity enables programming (scripting) in C# and Javascript. In this work, C# is used to programmatically instantiate gas particles (from the gas particle *Prefab*) and set their initial velocities and positions. In all simulations, particles are initially uniformly distributed within a container, while each gas particle is assigned the same speed in the direction chosen from the randomized set of uniformly distributed directions. Figure 1 shows an OO style pseudocode of the script used in this work. The script makes use of the *Start()* method to instantiate the objects and set the initial state, and the *LogVelocity()* method to periodically log the velocities of gas particles.

```
public class RunTimeCreator : MonoBehaviour {
  const int numParticles = 1000;
  GameObject[] gasParticles;
  void Start() {
    double scale = calcScale(numParticles);
    ScaleContainer(scale);
    SetCammeraZoom(scale);
    gasParticles = new GameObject[numParticles];
    Vector2[] velocities = RandomDirVectors(numParticles, 4.0f);
    Vector2[] positions = UniformPositions(numParticles, scale);
    for (i = 0; i < numParticles) {
      gasParticles[i] = InstantiateFromPrefab();
      gasParticles[i].setVelocity(velocities[i]);
      gasParticles[i].setPosition(positions[i]);
    }
    InvokeRepeating("LogVelocity", 0.0f, 5.0f);
  }
  private void LogVelocity() {
    static int i = 0;
    DumpVelocitiesToFile(gasParticles, "log" + i++);
  }
  private Vector2[] RandomDirVectors(int n, double vSize){
    Vector2[] vectors = new Vector[n];
    for (i = 0; i < numParticles; ++i) {
      vectors[i] = new Vector2(vSize * cos(2π/n*i),
                               vSize * sin(2π/n*i));
    }
    ShuffleArray(vectors) // Fisher-Yates algorithm
    return vectors;
  }
}
```

Figure 1. Setting the scene and initial state pseudocode

In general, to implement the desired behavior in Unity, a user has to extend the *MonoBehaviour* class – the base class from which every Unity script derives and overrides certain methods. The following methods are interesting in this context:

- *Start()* – called by Unity once on startup,
- *FixedUpdate()* – called by Unity in regular (fixed) time intervals (can run once, zero, or several times per frame).

FixedUpdate() is a part of the "physics cycle" in the execution order in Unity scripting, commonly used when one has to apply forces on rigid bodies [5]. To evolve in time the system with many particles of an ideal gas, it is not necessary to specify or calculate forces that are present in each collision. Hence, we do not use the *FixedUpdate()* method, and Unity progresses to the next part of the "physics cycle" called "*Internal physics update*" [5]. During this stage, the Unity physics engine evolves the system in time for one fixed time step. During this fixed time step, many collisions take place and the Unity physics engine handles all of them for us.

In the running Unity simulation, frame 0 corresponds to the initial state of our physical system. Figure 2 (left) shows the initial state for a container with 20 gas particles. It is apparent that particles are uniformly distributed within the container. The state at frame 100 is shown in Fig. 2 (right) where it is visible that particles have started to move in various directions. The size of the container equals 18.0×22.0 in Unity space units. This raises the following questions: How does Unity space units relate to the real-world units? To what real-world time does frame 100 correspond? In general, how does any Unity unit relate to the real-world unit?

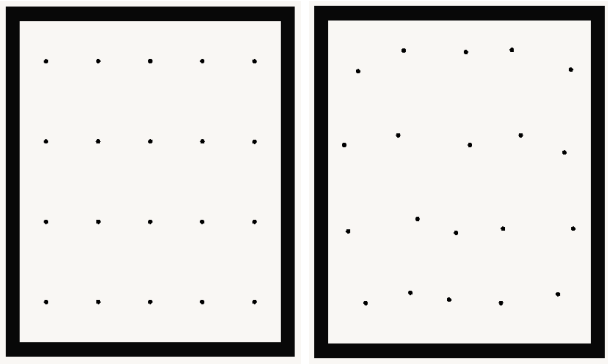

Figure 2. Container with 20 gas particles: initial state at frame 0 (left) and state at frame 100 (right)

C. Unity units vs. real-world units

One Unity space unit is a distance traveled by an object moving with the velocity of 1 Unity velocity unit for a time of 1 Unity time unit. If we choose 1 Unity velocity unit to correspond to 10^2 m/s and 1 Unity time unit to correspond to 10^{-11} s, then 1 Unity space unit corresponds to $10^2\ \mathrm{m/s} \cdot 10^{-11}\ \mathrm{s} = 10^{-9}\mathrm{m}$. Since we will not work with gravity or any other forces present in the analyzed physical system, we can choose for 1 Unity mass unit any real-world mass unit.

D. Some Unity physics engine fine-tuning

Unity exposes to users some of the physics engine parameters so users can optimize the physics engine according to their requirements. We have changed the default values for three parameters.

In our first simulations, we observed that particles with low speeds (less than 1 Unity velocity unit) stick to the walls instead of bouncing off the walls. This is probably a desirable property in games, but it is not in physics simulations. Luckily, Unity allows users to change this *Velocity Threshold* through GUI: *Edit* → *Project Setting* → *Physics 2D*. We set this value at 0.0001, which is the minimal value Unity allows for this property.

The physics engine is called at regular time intervals *Fixed Timestep* with the default value 0.02 (Unity time units). One can change this value through GUI: *Edit* → *Project Settings* → *Time*. In our simulations, we used a 10 times smaller value.

Physics engine handles collision detection between two objects with a collider. One of the parameters of the *Collider 2D* component is *Collision Detection*. We have changed the value of this parameter from *Discrete* to *Continuous* to prevent objects passing through each other

during an update. The drawback: additional computational demands on the physics engine.

III. RESULTS

To start the simulation, we needed some real-world data. We decided to work with neon gas because neon atoms are spherical and hence have no rotational degrees of freedom. Therefore, in our two-dimensional simulations, neon atoms have only two translational degrees of freedom. Van der Waals radius for a neon atom is 0.154 nm, and this value is used as the radius of the disk representing a neon atom in the Unity simulations. The mass of a neon atom is $3.35 \cdot 10^{-26}$ kg. The mass of an atom is not important in our simulations with only one type of atom. In simulations with several types of atoms, only the ratio of the masses of atoms is important, not the exact values.

In this work, we will perform three different simulations to study three different important properties of an ideal gas. In all three simulations, the state of the system (all particle velocities and positions) is saved into files in regular time intervals, and these data are analyzed using software for data analysis, in our case SageMath [6]. Notice that this procedure resembles real experiments: the experiment is performed, data are collected, and later these data are analyzed using the appropriate software.

A. Speed distribution for a two-dimensional ideal gas

One of the results of the kinetic theory of gases is the distribution of speeds in a two-dimensional ideal gas in an equilibrium named the Maxwell-Boltzmann distribution

$$f(v) = \frac{2v}{b^2} e^{-v^2/b^2}; \quad b^2 = \frac{2kT}{m}, \tag{1}$$

where $k = 1.38 \cdot 10^{-23}$ J/K is the Boltzmann constant, m is the mass of the gas particles, v is the speed of the gas particles, and T is the gas temperature. If the initial distribution of speeds is not Maxwell-Boltzmann, the system is not in equilibrium, but as time progresses, the system will reach equilibrium and the distribution of speeds will be Maxwell-Boltzmann.

To demonstrate the process of reaching equilibrium, we started our simulation with 500 gas particles uniformly distributed in the two-dimensional box with the concentration of $0.05\ 1/\text{Unity space units squared}$; each gas particle having the speed of 4 Unity velocity units in a random direction. As time progresses, the initial distribution of speeds widens due to collisions between gas particles. After 15 Unity time units, the peak around 4 Unity velocity units (400 m/s) is still present (Fig. 3), but after 25 Unity time units, the speed distribution is close to the Maxwell-Boltzmann distribution for two-dimensional gas with two translational degrees of freedom at the temperature T obtained using equipartition theorem:

$$2\frac{1}{2}kT = \frac{m\langle v^2\rangle}{2} \tag{2}$$

B. Fluctuations around average values for the system in equilibrium

If we count the number of gas particles in one half of the two-dimensional box from the previous virtual experiment, we expect that number to be about half the number of all particles in the box with some fluctuations in time. Let us denote by N_+ the number of gas particles in one half of the box and by N_- in the other half. The total number of particles in the box N equals $N = N_+ + N_-$. Using the kinetic theory of gases, one can show that these (relative) fluctuations when the system is in equilibrium depend on the number of gas particles in the box [2]:

$$r = \frac{\sqrt{\langle (N_+ - \langle N_+ \rangle)^2 \rangle}}{\langle N_+ \rangle} = \frac{1}{\sqrt{N}} \tag{3}$$

We used Unity to check this result. The size of the two-dimensional box is the same in all simulations. At the beginning of each simulation, particles are uniformly distributed in the two-dimensional box with each gas particle having the speed of 4 Unity velocity units in

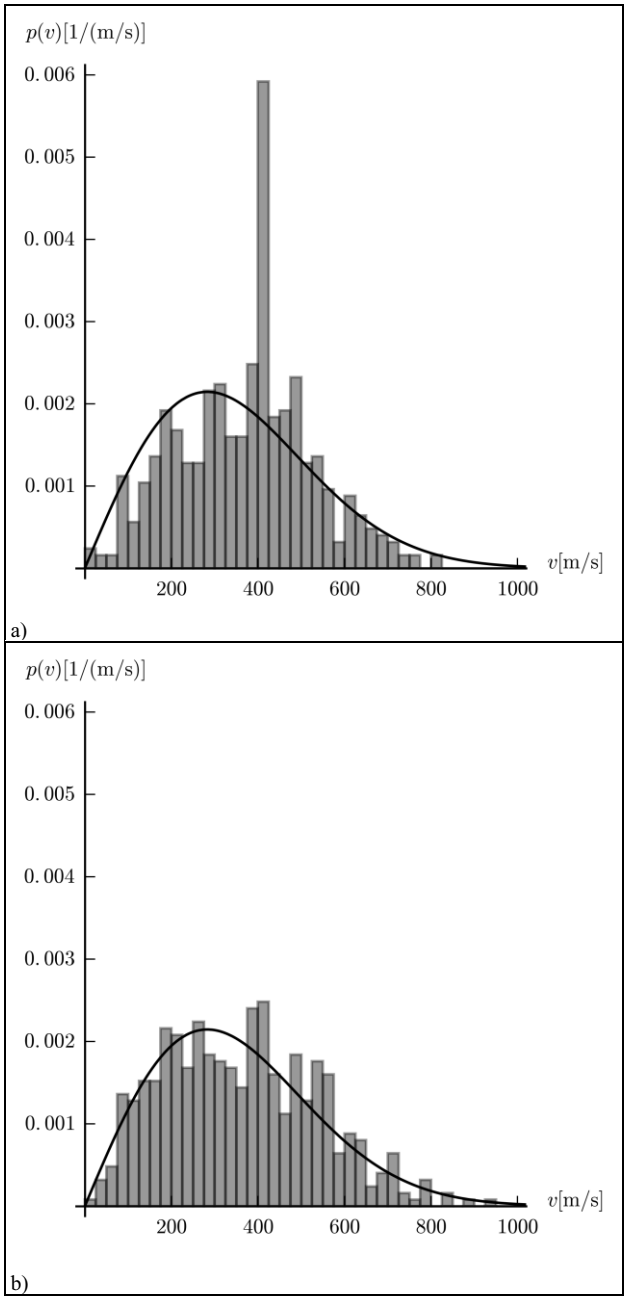

Figure 3. Normalized speed histograms for 500 gas particles at times a) 15 Unity time units, and b) 25 Unity time units. Initially, all gas particles have the speed of 4 Unity speed units (400 m/s). The full line corresponds to the Maxwell-Boltzmann distribution (1) for the two-dimensional ideal gas at the temperature T obtained using equation (2).

random direction. In regular time intervals (5 Unity time units), the positions of all particles are saved into files. Data are analyzed in SageMath, and results of this analysis are shown in Fig. 4.

The results from the Unity simulations are in agreement with the theory (Fig. 4). The nonlinear fit with the model function $a \cdot N^{-b}$ to these results (full line in Fig. 4) yields $b = 0.52 \pm 0.01$, which is close to theoretical $1/2$.

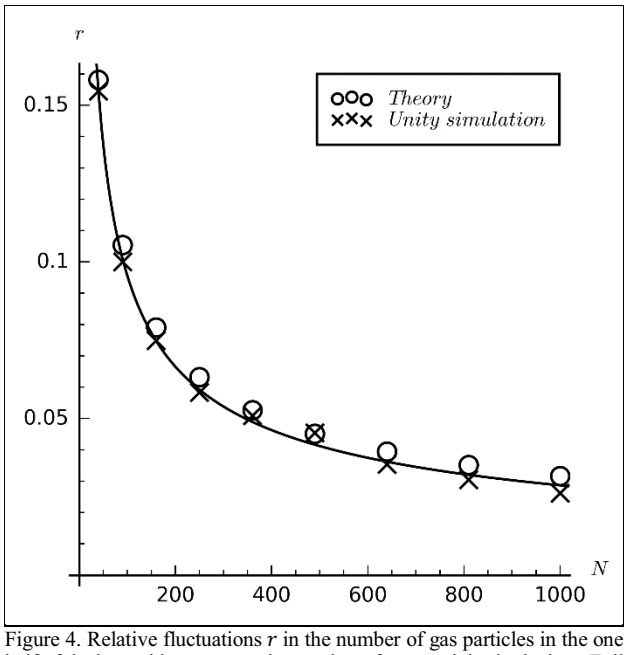

Figure 4. Relative fluctuations r in the number of gas particles in the one half of the box with respect to the number of gas particles in the box. Full line corresponds to the fit of Unity simulations data to the function $a \cdot N^{-b}$.

C. *Theorem of equipartition of energy*

The theorem of equipartition of energy (or simply equipartition theorem) says that energy in the system in thermal equilibrium is shared equally among all its independent parts (degrees of freedom). Hence, in the two-dimensional ideal gas with no rotational degrees of freedom in equilibrium, energy is shared among two translational degrees of freedom:

$$\langle \tfrac{1}{2} m_A v_{A,x}^2 \rangle = \langle \tfrac{1}{2} m_A v_{A,y}^2 \rangle, \qquad (4)$$

where m_A is the mass of gas particles, and $v_{A,x}$ and $v_{A,y}$ x and y components of the velocity vector, respectively. If we add to the system another type of gas particle with mass m_B (we are working now with the two-component two-dimensional ideal gas), the equipartition theorem predicts that in thermal equilibrium, the average kinetic energy of gas components are equal:

$$\langle \tfrac{1}{2} m_A v_A^2 \rangle = \langle \tfrac{1}{2} m_B v_B^2 \rangle. \qquad (5)$$

To see the equipartition theorem in action, we simulated in Unity two-component two-dimensional ideal gas with 500 gas particles of mass m_A equals to the mass of a neon atom and 500 gas particles of mass $m_B = 2 \cdot m_A$. Initially, the system is out of equilibrium: all gas particles of mass m_A have a speed of $400\ \mathrm{m/s}$ in the random direction while all gas particles of mass m_B are stationary. As time progresses, energy is exchanged between all gas particles, and the system approaches equilibrium where Eq. (5) is satisfied with some fluctuations in time (Fig. 5).

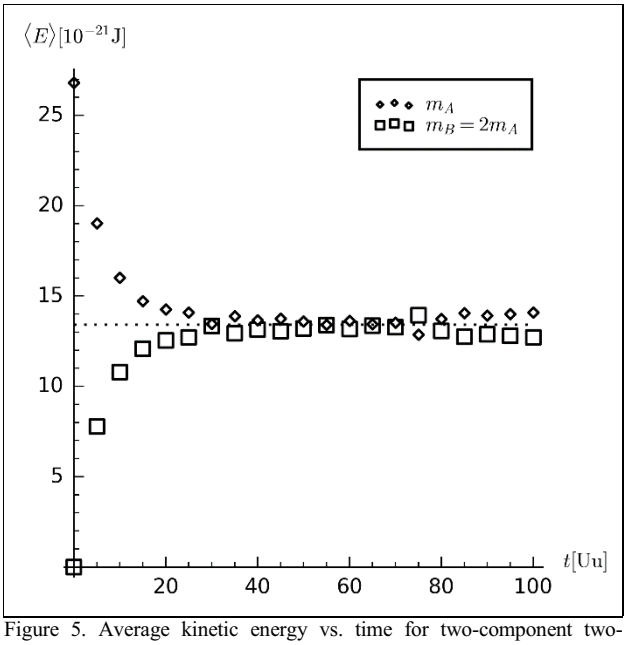

Figure 5. Average kinetic energy vs. time for two-component two-dimensional ideal gas with only translational degrees of freedom. Initially, all 500 gas particles of mass m_A have the same speed of $400\ \mathrm{m/s}$ in the random direction, and 500 gas particles of mass $m_B = 2 \cdot m_A$ are stationary. Initially, gas particles are uniformly distributed in the two-dimensional rectangular box. The dotted line shows the average kinetic energy that gas particles of both components should have in equilibrium according to the theorem of equipartition of energy. Uu stands for Unity units.

IV. Conclusion

We have shown that the Unity game engine, with its physics engine, can be used as a virtual physics laboratory in which students perform virtual experiments to study physics and deepen their understanding of physical laws. Unity features a graphical user interface and scripting support, and is easy to use and learn. Also, Unity is free for students. As shown in this work, with an understanding of just a few fundamental concepts (such as prefabs and unit conversion), students can create powerful and realistic simulations. This approach is probably best suited for ICT students. However, since in modern natural sciences, such as physics and biophysics, computer simulations of physical processes are very important, they might also find this approach of some interest.

References

[1] Unity Technologies, Unity®, Version 2017.3.0, 2017.

[2] F. Reif, Statistical Physics, vol. 5, McGraw-Hill, 1967.

[3] J. Novak and A. B. Bortz, "The evolution of the two-dimensional Maxwell-Boltzmann distribution", Am. J. Phys., vol 38, pp. 1402–1406, 1970.

[4] R. P. Bonomo and F. Riggi, "The evolution of the speed distribution for a two-dimensional ideal gas: A computer simulation", Am. J. Phys., vol. 52, pp. 54–55, 1984.

[5] Unity Technologies, Unity Manual: Execution Order of Event Functions, Unity®, Version 2017.3.0, 2017.

[6] The Sage Developers, SageMath, the Sage Mathematics Software System (Version 8.0), 2017, http://www.sagemath.org.