An Interactive Experience on the Transformation of Energy into Motion.

Ricardo Lagunes Tejeda, [1], Surya N. Ramírez Hernández[2]

[1] Lead Gameplay and Physics Programmer. Editorial Department of Knosphere, PRISCONE. email: ricardo.lagunes@priscone.com.

[2] Physics and Mathematics Analyst. Editorial Department of Knosphere, PRISCONE. email: surya.ramirez@priscone.com.

Abstract

This article examines the physical concepts underlying Kinerto, with particular emphasis on elastic potential energy, impulse, and the transformation of stored energy into motion. By connecting the game's computational mechanics with their physical interpretation, it aims to promote conceptual understanding, develop physical intuition, and support the recognition of these phenomena in real-world situations.

Keywords: elastic potential energy; impulse; velocity; energy transformation, physics intuition.

1. Introduction

Energy, impulse, and velocity are fundamental concepts for describing and understanding the motion of bodies. Teaching these concepts often requires students to relate abstract quantities to phenomena that must be explored through observation and experimentation.

Physical intuition can begin to develop at any age through experiences that allow individuals to observe, anticipate, and compare the effects of different actions. Gamification offers a means of introducing these phenomena to a broad audience without requiring, as an initial step, mastery of their complete mathematical formulation.

Kinerto was designed as a gamified interactive experience in which the length of time a button is held down determines the accumulated charge and, subsequently, the initial velocity of a ball. The interaction therefore functions as a simplified representation of the accumulation of elastic potential energy, the application of an impulse, and the generation of motion.

The main objective of this article is to analyze the physical and computational foundations of Kinerto's core mechanic and examine its potential to establish connections between the in-game experience and observable real-world situations, thereby providing a conceptual foundation for the subsequent formal study of these phenomena.

2. Theorical Foundations

2.1. Elastic Potential Energy

Potential energy is the energy associated with the configuration of a system. In an elastic system, energy can be stored when an object—such as a spring—is compressed or stretched from its equilibrium position. For an ideal spring that obeys Hooke's law, the elastic potential energy is given by

$$U_e = \frac{1}{2}kx^2,$$

[1, 2].

where U_e is the elastic potential energy measured in joules, k is the spring constant measured

in newtons per meter, and x is the displacement from the equilibrium position measured in meters.

Kinerto does not contain a physical spring. Instead, holding down the control button represents the compression of an idealized elastic mechanism. The longer the button is held, the greater the virtual displacement and the amount of energy stored by the system.

In the original implementation, this accumulation is represented by the following instructions:

```
_totalAccumulatedEnergy += _deltaPotentialEnergy;
_totalAccumulatedEnergy =
    Mathf.Min(_totalAccumulatedEnergy,
     _maxEnergyCap);
```

This code increases an abstract charge during every physics update and restricts it to a maximum value. Although the variable is named `_totalAccumulatedEnergy`, it does not yet calculate physical energy in joules. To establish a more direct correspondence with the physical model, the code can represent the button-hold duration as a virtual compression:

```
_compression += _compressionRate * Time.
     fixedDeltaTime;
_compression = Mathf.Min(_compression,
     _maxCompression);

_elasticPotentialEnergy =
    0.5f * _springConstant * _compression *
     _compression;
```

This implementation follows the mathematical expression

$$U_e = \frac{1}{2}kx^2.$$

The maximum compression,

```
_compression = Mathf.Min(_compression,
     _maxCompression);
```

also constitutes a game-design decision. It prevents indefinite energy accumulation and defines the range within which the player must estimate the appropriate charge.

2.2. Transformation of Energy into Motion

When the button is released, the energy represented as stored in the virtual elastic mechanism is transformed into the motion of the ball. The kinetic energy of an object of mass m and speed v is

$$K = \frac{1}{2}mv^2.$$

For an ideal system without friction or other dissipative effects, conservation of mechanical energy allows the elastic potential energy to be equated with the resulting kinetic energy:

$$U_e = K,$$

and therefore

$$\frac{1}{2}kx^2 = \frac{1}{2}mv^2.$$

Solving for the speed gives

$$v = x\sqrt{\frac{k}{m}},$$

or, when the stored energy is already known,

$$v = \sqrt{\frac{2U_e}{m}}.$$

This relation can be represented computationally as follows:

```
float speed = Mathf.Sqrt(
    2f * _elasticPotentialEnergy / _rigidbody.mass
);
```

The square root is necessary because kinetic energy depends on the square of velocity. Consequently, energy cannot be converted directly into velocity through a simple linear multiplication.

2.3. Linear Momentum

Linear momentum describes the quantity of motion of an object and is defined as the product of its mass and velocity:

$$\vec{p} = m\vec{v}.$$

Momentum is a vector quantity because it has both magnitude and direction. Two objects with the same mass and speed can therefore have different momenta if they move in different directions.

Once the launch speed and direction of the ball have been determined, its momentum can be calculated in the game as

```
Vector3 velocity =
    direction.normalized * speed;

Vector3 momentum =
    _rigidbody.mass * velocity;
```

In this representation, ‘direction.normalized‘ is a unit vector that specifies the direction of motion, while ‘speed‘ specifies its magnitude.

2.4. Impulse

Impulse measures the effect of a force acting during a time interval. For a force that may vary with time, impulse is defined as

$$\vec{J} = \int_{t_i}^{t_f} \vec{F}(t)\, dt.$$

If an average force is used, the expression becomes

$$\vec{J} = \vec{F}_{\text{avg}} \Delta t.$$

The impulse–momentum theorem establishes that the impulse applied to an object is equal to its change in momentum:

$$\vec{J} = \Delta\vec{p} = \vec{p}_f - \vec{p}_i.$$

[2]

If the ball is initially at rest, then $\vec{p}_i = 0$, and therefore

$$\vec{J} = \vec{p}_f = m\vec{v}.$$

This relation can be represented in code by calculating the required impulse from the ball's final velocity:

```
Vector3 initialMomentum =
    _rigidbody.mass * _rigidbody.velocity;

Vector3 finalVelocity =
    direction.normalized * speed;

Vector3 finalMomentum =
    _rigidbody.mass * finalVelocity;

Vector3 impulse =
    finalMomentum - initialMomentum;
```

The impulse can then be applied through Unity's physics system:

```
_rigidbody.AddForce(
    impulse,
    ForceMode.Impulse
);
```

This implementation creates a direct correspondence with the impulse–momentum theorem:

$$\vec{J} = \vec{p}_f - \vec{p}_i.$$

When the ball begins from rest and the stored elastic energy is transformed entirely into kinetic energy, energy and impulse can also be related through

$$U_e = \frac{1}{2}mv^2$$

and

$$J = mv.$$

Substituting the velocity obtained from the energy equation gives

$$J = \sqrt{2mU_e}.$$

This expression does not imply that energy and impulse are the same physical quantity. Energy is measured in joules, whereas impulse is measured in newton-seconds. Instead, it shows how the amount of stored energy determines the impulse required to produce the corresponding change in the ball's momentum.

2.5. Velocity and Vector Direction

Velocity is a vector quantity that combines speed with direction. In three-dimensional space, it can be expressed in component form as

$$\vec{v} = v_x\hat{\imath} + v_y\hat{\jmath} + v_z\hat{k}.$$

Kinerto restricts the player's selection to four predefined directions. The original code represents them with Unity's ‘Vector3‘ structure:

```
case Direction.NorthEast:
    return new Vector3(1f, 0f, 0f);

case Direction.SouthEast:
    return new Vector3(0f, 0f, -1f);

case Direction.NorthWest:
    return new Vector3(0f, 0f, 1f);

case Direction.SouthWest:
    return new Vector3(-1f, 0f, 0f);
```

Each vector establishes the direction of motion, while the previously calculated speed determines its magnitude:

```
Vector3 velocity =
    direction.normalized * speed;
```

Mathematically, this operation is represented as

$$\vec{v} = v\hat{d},$$

where v is the speed and $\hat{d}$ is a unit vector indicating the selected direction. This separation between magnitude and direction allows the same accumulated energy to produce motion toward different regions of the game environment.

Referencias

[1] D. Halliday, R. Resnick, and J. Walker, *Fundamentals of Physics*, 12th ed. Hoboken, NJ, USA: John Wiley & Sons, 2021.

[2] W. Moebs, S. J. Ling, and J. Sanny, *University Physics, Volume 1*. Houston, TX, USA: OpenStax, Rice University, 2016. Available: https://openstax.org/details/books/university-physics-volume-1