Development of a model of the solar system in AR and 3D

Valentyna V. Hordiienko[1[0000-0003-1271-9509]], Galyna V. Marchuk[1[0000-0003-2954-1057]], Tetiana A. Vakaliuk[1[0000-0001-6825-4697]] and Andrey V. Pikilnyak[2[0000-0003-0898-4756]]

[1] Zhytomyr Polytechnic State University, 103, Chudnivska Str., Zhytomyr, 10005, Ukraine
rafiktgebestlove@gmail.com, pzs_mgv@ztu.edu.ua, tetianavakaliuk@gmail.com

[2] Kryvyi Rih National University, 11 Vitalii Matusevych Str., Kryvyi Rih, 50027, Ukraine
pikilnyak@gmail.com

Abstract. In this paper, the possibilities of using augmented reality technology are analyzed and the software model of the solar system model is created. The analysis of the available software products modeling the solar system is carried out. The developed software application demonstrates the behavior of solar system objects in detail with augmented reality technology. In addition to the interactive 3D model, you can explore each planet visually as well as informatively – by reading the description of each object, its main characteristics, and interesting facts. The model has two main views: Augmented Reality and 3D. Real-world object parameters were used to create the 3D models, using the basic ones – the correct proportions in the size and velocity of the objects and the shapes and distances between the orbits of the celestial bodies.

Keywords: augmented reality, virtual reality, ARCore, planet, solar system.

1 Introduction

1.1 Problem statement

The development of technology in the 21st century is extremely fast. One of these technologies is Augmented Reality (AR). This technology has a direct vector in the future. The urgency of the introduction of augmented reality technology, especially in the educational process, is that the use of such a new system will undoubtedly increase students' motivation, as well as increase the level of assimilation of information due to the variety and interactivity of its visual presentation.

Augmented reality is a concept that describes the process of augmenting reality with virtual objects. Virtual reality communication is performed on-line, and only the camera is required to provide the desired effect – images that will be complemented by virtual objects. The relevance of the introduction of augmented reality technology in the educational process is that the use of such a new system will undoubtedly increase students' motivation, as well as increase the level of assimilation of information due to the variety and interactivity of its visual presentation [12]. ARCore is a state-of-the-art cross-platform kernel for application creation, designed for the entire development process to take place in the bundled integrated development environment.

1.2 Literature review

Vladimir S. Morkun, Natalia V. Morkun, and Andrey V. Pikilnyak view augmented reality as a tool for visualization of ultrasound propagation in heterogeneous media based on the k-space method [9].

Mariya P. Shyshkina and Maiia V. Marienko considering augmented reality as a tool for open science platform by research collaboration in virtual teams. Authors show an example of the practical application of this tool is the general description of MaxWhere, developed by Hungarian scientists, and is a platform of aggregates of individual 3D spaces [13].

Tetiana H. Kramarenko, Olha S. Pylypenko and Vladimir I. Zaselskiy view prospects of using the augmented reality application in STEM-based Mathematics teaching, in particular, the mobile application 3D Calculator with Augmented reality of Dynamic Mathematics GeoGebra system usage in Mathematics teaching are revealed [7].

Pavlo P. Nechypurenko, Viktoriia G. Stoliarenko, Tetiana V. Starova, Tetiana V. Selivanova considering development and implementation of educational resources in chemistry with elements of augmented reality, and as a result of the study, they were found that technologies of augmented reality have enormous potential for increasing the efficiency of independent work of students in the study of chemistry, providing distance and continuous education [10].

Anna V. Iatsyshyn, Valeriia O. Kovach, Yevhen O. Romanenko, Iryna I. Deinega, Andrii V. Iatsyshyn, Oleksandr O. Popov, Yulii G. Kutsan, Volodymyr O. Artemchuk, Oleksandr Yu. Burov and Svitlana H. Lytvynova view the application of augmented reality technologies for the preparation of specialists in the new technological era [4].

Lilia Ya. Midak, Ivan V. Kravets, Olga V. Kuzyshyn, Jurij D. Pahomov, Victor M. Lutsyshyn considering augmented reality technology within studying natural subjects in primary school [8].

Aw Kien Sin and Halimah Badioze Zaman [14] researching the Live Solar System (LSS), which is a learning tool to teach Astronomy. They are user study was conducted to test on the usability of LSS, in findings of the study concluded that LSS is easy to use and learn in teaching Astronomy.

Vinothini Kasinathan, Aida Mustapha, Muhammad Azani Hasibuan and Aida Zamnah Zainal Abidin in the article [6] presents an AR-based application to learn Space and Science. They are concluded that the proposed application can improve the ability of children in retaining knowledge after the AR science learning experience.

There are currently several software products that simulate the solar system. Consider some of them.

The AR Solar System application [1] uses AR technology, which was implemented using the Vuforia service. This application provides the opportunity to see the planets of the solar system directly in front of you. At the same time, there is no interaction with the user, except for the ability to choose the screen orientation: horizontal or vertical. When launched, the application automatically selects the location of the planets and this is not always successful. Besides, if you change your location, the location of the planets does not change. The planets revolve around the Sun and its axis, but this is the whole functionality of the application.

The Solar System AR application [11] involves printing or opening a special card on another device, which will then display the planets. At startup, you need to place an image in front of the phone's camera, which consists of squares in which the first letters of the name of the planet are written. As a result, the planets begin to appear along with the soundtrack. However, with the slightest movement of the phone's camera, all the planets disappear and begin to reappear, as does the soundtrack. Similarly, as in the first application, there is no interaction with the user.

The next analog is the Solar System (AR) [17]. The application has much more functionality than the previous two: there is a menu where you can go to the introduction, which tells some interesting facts; the very reflection of the solar system; quiz, which contains questions about the structure, characteristics, features of all objects. By selecting the transition to the solar system, the application places the Sun in the center of the screen, all the others – on their axes, which, incidentally, are located inconsistently with reality. By clicking on any planet, you can see minimal information about it with sound.

Another application that is worth noting is the Solar System Scope [5]. This application has good graphics: from the download bar to display the planets themselves. There are also constellations and a large number of stars that are not usually reflected in the solar system. The application has music in space style; there are enough settings to make the application convenient for each user. In particular, the planets can be enlarged, reduced, rotated 360°. Having chosen any planet or star, you can see it in section, read about it quite detailed information. There is a function to search for planets, stars, comets, constellations, etc., display in real-time or in the past or future, and accelerate the pace of rotation. That is, this application has many advantages and is very convenient for studying all space objects. However, it does not use augmented reality.

2 Methods

2.1 Why ARCore?

The *purpose* of the article is to develop an application that implements the Solar System model in AR and 3D.

Today AR is developing rapidly. There are already many platforms where you can create applications with this technology: ARToolKit, Cudan, Catchoom, Augment, Aurasma, Blippar, InfifnityAR, Layar SDK, Vuforia and more. This work will use ARCore, a platform developed by Google as a tool for creating augmented reality applications. Its advantages over others are: it is well-developed, provides many features, has content documentation, is free, compatible with the Unity engine [2; 3].

There are two main actions of ARCore: finding the current position of your device while moving and shaping your 3D world. For this purpose, technologies are used to track the position of the phone in the real world using its sensors; understanding of the shapes of the environment, finding the vertical, horizontal and angular surfaces of the plane; lighting assessment.

The position of the user along with the device is determined by various sensors (accelerometer, gyro) built into the gadget itself. At the same time, there are so-called key points and their movement is being explored. With these points and sensor data, ARCore determines the tilt, orientation, and position of the device in space.

To find the surfaces, move the camera. This is to help ARCore build it's real-world based on moving objects. Even if you leave the room and then return, all of the objects you have placed will remain in place, remembering their location relative to the key points and the locations relative to the world built by ARCore.

Lighting is also evaluated. This is done using signal recognition: the main light is the light emitted by an external source and illuminates all around; shadows that help you identify where the light source is; shading – is responsible for the intensity of illumination in different areas of the object, that is, helps to understand how remote from the source are parts of it; glares that seem to glow, that is, directly reflect the light stream from the source and vary depending on the position of the device relative to the object; a display that depends on the material of the illuminated object. By the way, the formed idea of illumination will influence the illumination of objects placed by the user. For example, if you place one 3D model by the window and the other under the table, the first will be bright and the other as if in shadow. This creates the effect of reality.

2.2 Modeling the movement of astronomical objects

Long-standing scientists have been trying to understand how objects move in the solar system. Most accurately the motion of planets, stars, asteroids – all objects – was characterized by Kepler, who discovered the three laws on which Newton derived the formula of gravity.

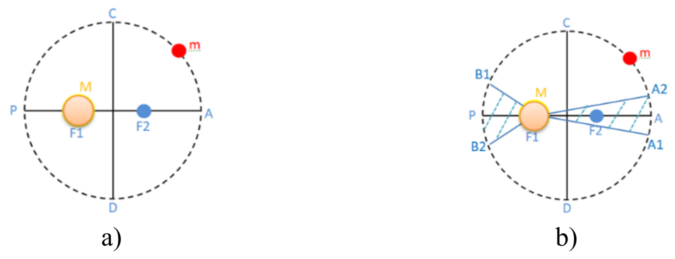


Fig. 1. Graphic representation of the first (a) and second (b) law of Kepler.

The first law states that all planets move in elliptical orbits (trajectories) in one of the focuses of which is the Sun. That is, all planets move around the center of mass, which is in the Sun because its mass is much larger than the mass of any planet. Figure 1 (a) show the elliptical orbit of the planet, the Sun with mass M, which is several times the mass of the planet m. The sun is in focus F1. The point closest to the Sun in the orbit of P is called perihelion, and the farthest A is aphelion or apogeal.

Another important concept is the eccentricity, which characterizes the degree of compression (elongation) of the orbit (table 1).

Table 1. The shape of the orbit depending on the eccentricity.

The value of the eccentricity	0	(0; 1)	1	(1; ∞)	∞
The orbit shape	circle	ellipse	parabola	hyperbola	straight

By Kepler's second law we learn that each planet moves so that its radius vector (the segment connecting the planet and the sun) passes the same area over the same period. Figure 1(b) shows that for the equivalent time *t*, during which you can go from point A1 to A2 and from B1 to B2, the identical areas S are throws.

This law explains why the movement of the planet accelerates as it approaches perihelion and decelerates when it reaches aphelion. That is, for these areas to be the same, the planet must move faster if it is close to the Sun and slower if far away. This law determines the speed of movement of the celestial body.

He also determined that the squares of the planets' rotation periods (*T*) refer to each other as cubes of the large hemispheres (*a*) of these orbits. This is Kepler's third law.

$$\frac{T_1^2}{a_1^3} = \frac{T_2^2}{a_2^3} \tag{1}$$

However, these laws are not enough to accurately describe the movement of all bodies. After all, they work only when the weight of one body in the group under consideration is significantly higher than the other. That is, if you consider the planet and its satellite, they will not work. Newton was able to correct it. He changed the third law by adding mass to it.

$$\frac{a_1^3}{a_2^3} = \frac{T_1^2}{T_2^2} \cdot \frac{M-m_1}{M-m_2} \tag{2}$$

The main force that controls the motion of the planets is the force of gravity. However, if bodies were attracted only to the Sun, they would move only according to Kepler's laws. In fact, the bodies are attracted not only by the sun but also to each other. Therefore, nobody in the solar system moves in a perfect circle, ellipse, parabola, etc.

Considering the basic algorithms of the program and the principles of motion of the planets, we can distinguish the basic parameters (table 2 and table 3) that you need to know about objects: eccentricity (ε), perihelion, a period of rotation around the Sun (orbital period), equatorial radius (the radius of the planet), the angle of inclination of the axis, the period of rotation about its axis (the period of rotation), the length of the ascending node (Ω).

2.3 Animation implementation

LeanTween was used to implement the animations, an efficient twin plugin used by Unity [15]. Its benefits are simple implementation, Canvas UI animation support, spline or Bezier curves, and event handler support available on the Asset Store [16].

Table 2. Object orbit parameter values.

	Eccentricity, ε	Perihelion	Length of ascending node, Ω
Sun	0	0	0
Mercury	0,2056	0,3075	48,3
Venus	0,0067	0,718	76,7
Earth	0,0167	0,9833	349
Mars	0,0934	1,3814	49,6
Jupiter	0,0488	4,950	101
Saturn	0,0542	9,021	114
Uranium	0,0472	18,286	74,2
Neptune	0,0086	29,76607095	131
Moon	0,055	0,00242	0,0

Table 3. The values of the parameters of planets and satellites.

	The orbital period, years	Radius of the planet	Axis angle, °	Rotation period
Sun	0	0,004649197	7,25	-0,7049998611
Mercury	0,241	0,00001630347	0,01	-0,15834474
Venus	0,615	0,00004045454	177,4	0,67
Earth	0,98329134	0,00004263435	23,45	-0,00273032773
Mars	1,882	0,0000233155	25,19	-0,0029
Jupiter	11,86	0,00047660427	3,13	-0,0011
Saturn	29,457	0,00040173796	26,73	-0,0012
Uranium	84,016	0,00017713903	97,77	0,0019
Neptune	164,791	0,00016544117	28,32	-0,00183775013
Moon	0,074	0,00001161764	6,69	-0,07945205479

The *ToggleBodies* method animates the appearance of a list of planets (on the left) when the corresponding key is pressed.

The *isOn* variable specifies whether to show the panel or hide it. *Value* method provides the necessary parameters for animation, such as color, size, etc. The example shows an animation of the panel placement variable. By default, it is located on the left side outside the visible window; it should be moved slightly to the right. The first parameter is the object, the second is the starting position, the third is the position to be reached, and the fourth is the time during which the transition will take place. The smoothness of the transition is governed by *setEase*(), which carries the name of the Bezier curve – *easeOutCubic*. In other words, the animation will be faster at first and slower at the end, and the panel will approach the destination smoothly. The *setOnStart*(), *setOnUpdate*(), *setOnComplete*() methods are required to manage changes. In this example, we use *setOnUpdate*(). That is, the position value will change with each frame. It calculates according to the distance, time, and curve that the subject should shift.

The *UIAnimationController.cs* script manages all the animations.

2.4 Design and implementation of individual modules of the system

The most important algorithm for organizing the motion of objects is to create an orbit for a planet, star, or satellite. Three scripts are responsible for this: *OrbitData*, *Orbiter*, *OrbitRenderer* (fig. 2).

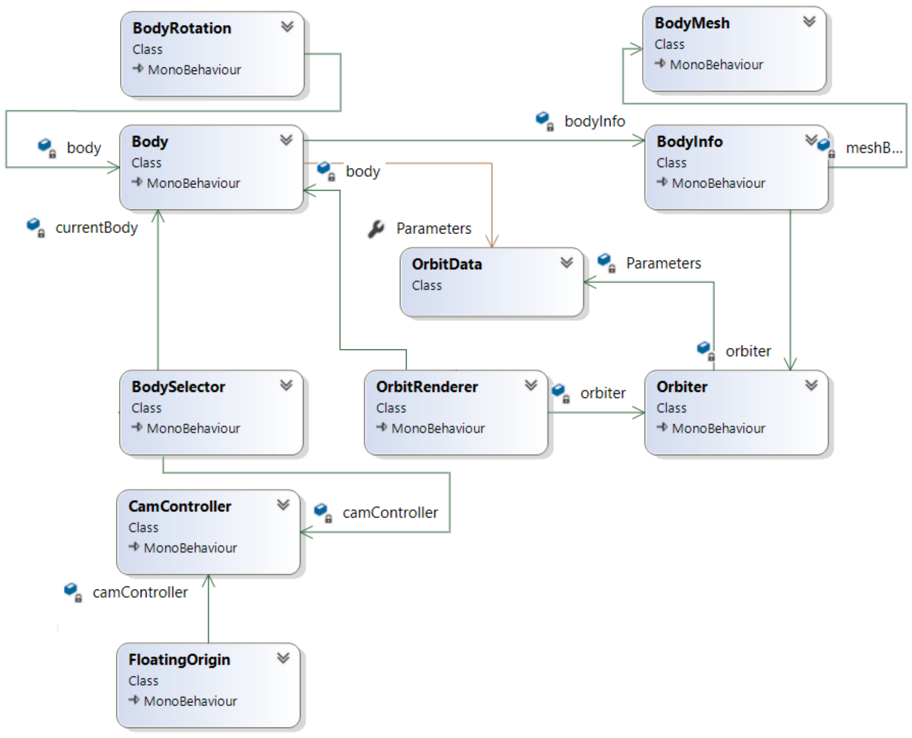


Fig. 2. Diagram of classes related to orbit creation.

OrbitData contains all the necessary data (parameters) about the orbit: eccentricity (degree of deviation from the circle), perihelion (closest to the sun orbit point of the planet), longitude, orbital period (period of the orbit), a period of rotation (around the axis), slope axis, radius, and mass (see table 2 and table 3).

Orbiter finds the shape of the orbit depending on the input parameters and moves the object along the found trajectory. Its main task is to form an array with a set of angles, each of which at some point in time must return an object to move in the trajectory of its orbit.

Consider in detail the process of performing the basic algorithms. The *time* variable contains the time elapsed since the application started. It will determine the position of the object in orbit. *Theta* is the angle at which time the object must return. *N* is the number of points that will merge into segments, thus forming an orbit. The *angleArray* array contains a list of angles from which one is required. *CosSinOmega* simply contains the values of the cosine and sine of the angle, created to avoid repetition in the code. *OrbitDeltaTime* (abbreviated as *OrbitDt*) defines the time it takes an object to move one step. The initial value of perihelion (*initialPerihelion*) is only used to work with the moon. Parameters include parameters from *OrbitData* for the current entity.

In *Start*, the necessary data is initialized, which will then be used in different methods. Variable *k*:

$$k = \frac{2\pi}{(1-e^2)^{\frac{3}{2}} \cdot orbitalPeriod} \tag{3}$$

where *e* is the eccentricity of the object, *orbitalPeriod* is the period of rotation of the planet around the Sun.

The *orbitDt* variable (the time in which one step is performed) is equal to the rotation period divided by the double number of steps that the object must pass:

$$orbitDt = \frac{orbitalPerod}{2 \cdot (N-1)} \tag{4}$$

The random variable from zero to the maximum value of the orbital period is written to the time variable. That is, when you launch an application, the object will appear in a different place, not every time in the same place.

Before selecting the time (time), you need to fill the *angleArray* array with angles. This is called the *ThetaRunge* method, which implements the fourth-order Runge-Kutta method, which gives more accurate results of differential equation calculations. Five steps are required to implement it: calculate the values of the four variables k_1 (5), k_2 (6), k_3 (7) and k_4 (8) and find the final result (9).

$$k_1 = f(x_i, y_i) \tag{5}$$

$$k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{hk_1}{2}\right) \tag{6}$$

$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{hk_2}{2}\right) \tag{7}$$

$$k_4 = f(x_i + h, y_i + hk_3) \tag{8}$$

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9}$$

The result we need – the value of the angle – is recorded at each step after finding the exact value of *w*. The last value is the number π because this is the end of the half orbit if the countdown starts from zero. That is, in the array are the values of the angles for half of the ellipse, and when the object begins to move in the second half – the angle will be counted in the opposite direction.

To prove the effectiveness of the Runge-Kutta method, we compare the values found with and without its algorithm (table 4). As a result of the calculations, we can conclude that using this method results in more accurate values and, at the same time, smoother displacement of the object.

Table 4. Comparison of angles.

	Runge-Kutta method			Simple calculation
	$i = 0$	$i = 1$	$i = 2$	
k_1	0,0127275451	0,0127273692	0,0127264807	
k_2	0,0127275012	0,0127271489	0,0127264447	
k_3	0,0127275012	0,0127271489	0,0127264447	
k_4	0,0127273692	0,0127268407	0,0127259605	
w	0,0127274865	0,0254546208	0,0381810508	
a[1]	0,0127274865	-	-	0,0127275451
a[2]	-	0,0254546208	-	0,0254550902
a[3]	-	-	0,0381810508	0,0381826353

After all the data in *FixedUpdate* have been calculated, the object moves in orbit. *FixedUpdate* was used because it is called a fixed number of times (fifty per second). As a result, the application will work the same on devices with different processors.

The current state of motion of the objects is checked and, unless paused, the time increases and the position of the object (*localPosition*) changes. That is, the object is constantly assigned a new position value, depending on how long it has been since the application was launched.

ThetaInt takes the time value and returns the angle you want to rotate. *Theta0* variable is the resultant angle, the value that returns the method.

```
public float ThetaInt (float t){
  float theta0 = 0;
```

We find the remainder of the division of time into the orbital period to obtain the value of elapsed time from the reference point – the starting point of the orbit. That is, if, for example, Mars has an orbital period of 1,882, and the value of elapsed time is 4,576, then it will be known that from the point of reference the planet has passed 0,812 years and to the endpoint, it is 1,070 years.

Next, we need to check which part of the orbit (in the first half of the second) is the planet. If in the first (0; π], find the number of steps completed:

```
  if(t <= Parameters.orbitalPeriod / 2){
    float i = t / orbitDt;
```

Since the variable *i* is of the float type and we mean an integer step, the exact value must be determined. To do this, use *Floor* and *Ceil* to round down the number found to the smaller and larger sides, respectively, and using *Clamp* to get the values:

```
    float i0 = Mathf.Clamp (Mathf.Floor (i), 0, N - 1);
    float i1 = Mathf.Clamp (Mathf.Ceil (i), 0, N - 1);
```

If the sequence number of the step is the same, select the value in the pre-formed array:

```
    if (i0 == i1) theta0 = angleArray [(int)i0];
```

If as a result of comparisonб we get two different numbers, then by formula (10) we find the value of the angle and return it:

$$theta0 = \frac{a|i_0|-a|i_1|}{i_0-i_1} \cdot i + \frac{i_0 \cdot a|i_0|-a|i_1| \cdot i_1}{i_0-i_1} \tag{10}$$

If the object is in the second part of the orbit (π; 2π), we already write to the variable *t* how much time it takes to get to the end of the orbit, and use the same algorithm to determine the current step:

```
    else { t = -t + Parameters.orbitalPeriod;
      float i = t / orbitDt;
      float i0 = Mathf.Clamp (Mathf.Floor (i), 0, N - 1);
      float i1 = Mathf.Clamp (Mathf.Ceil (i), 0, N - 1);
```

When calculating the *theta0* angle, we have the same formulas, but make the angle from the *angleArray* array negative and add 2π to it. This is necessary for the angle value to be selected in the opposite direction.

```
      if (i0 == i1)
        theta0 = -angleArray [(int)i0] + 2 * Mathf.PI;
      else
        theta0 = -((angleArray[(int)i0] –
          angleArray[(int)i1]) / (i0 - i1) * i + (i0 *
          angleArray[(int)i1] - angleArray [(int)i0] *
          i1) / (i0 - i1)) + 2 * Mathf.PI;
      return theta0;
```

ParametricOrbit takes the value returned by *ThetaInt*. Its purpose is to convert the received angle into coordinates (vector) because in Unity the object can be moved either by applying its force or by changing the current coordinates. At the beginning we find the values of cosine and sine of the angle:

```
public Vector3 ParametricOrbit (float th){
    float Cost = Mathf.Cos (th);
    float Sint = Mathf.Sin (th);
```

Next, we use the formulas (11) and (12) to calculate the values of the variables x and z that are needed to find the desired coordinates:

$$z = \frac{p \cdot (1+e)}{1+e \cdot cos\, t} \cdot \cos t \tag{11}$$

$$z = \frac{p \cdot (1+e)}{1+e \cdot \cos t} \cdot \sin t \quad (12)$$

where p is the value of perihelion, e is the eccentricity.

The formulas for finding the velocity hodograph in the plane (13) and (14) find the resulting values of the coordinates x and z (y is always zero):

$$zp = \cos\Omega \cdot x - \sin\Omega \cdot z \quad (13)$$

$$zp = \sin\Omega \cdot x + \cos\Omega \cdot z \quad (14)$$

GetVelMagnitude translates the length of the Unity velocity vector in kilometers per second, called in other scripts.

Scales_ScaleModeChanged is an event handler that controls the distance between the Earth and the Moon. The Moon mustn't intersect with the Earth, that is, increases the distance between them.

OrbitRenderer uses the found data from *Orbiter* and outlines the orbit.

Starting is initialized. And *LateUpdate* draws a line by connecting the dots. You can edit the number of points by changing the value of the *lineRendererLength* variable. The greater its value, the smoother the line display. The dotted orbit is achieved by assigning it to the appropriate texture.

2.5 Steps of implementation of support for ARCore technology

First, you need to check your support for ARCore technology with the gadget. This is managed by the *ARSupportChecker* script. The *CheckSupport*() method records the current state of support.

If you want to install the ARCore application, then go to the *Install*() method.

If the previous method found the gadget to be supported, a button to go to the AR scene is unlocked and the user will see a message at the bottom of the screen.

If the user learns that his device does not support ARCore, then he can use the application in 3D mode.

One of the main scripts is *BodyManager*. Looking at its name, we understand that it controls all objects, i.e. planets, satellites, asteroids, stars, and more. Its main task is to create the object itself. This is done in the *CreateStar*(), *CreatePlanet*(), *CreateMoon*(), *CreateAsteroidsBelt*() methods. They are similar to each other, so let's look at creating planets and asteroids, for example.

```
private void CreatePlanet(string name,
  OrbitData orbitData, bool hasBelt) {
  GameObject planet = Instantiate(planetPrefab)
    as GameObject;
  planet.name = name;
  planet.tag = "Planet";
  planet.layer = 9;
  planet.transform.Find("Planet").name = "Mesh" + name;
  planet.transform.Find("BB").name = "BB" + name;
  planet.transform.Find("Mesh" + name).localScale =
    Vector3.one * orbitData.radius;
  planet.transform.parent = transform;
  planet.GetComponent<Body>().Parameters = orbitData;
  if (hasBelt){
    GameObject rings =
      Instantiate(Resources.Load("Prefabs/Rings") as
       GameObject) as GameObject;
    rings.transform.parent =
      planet.transform.Find("Mesh" + name);
    rings.transform.localScale = new Vector3(5, 5, 5);}
  }
```

Essentially, prefab information is being filled. The *Instantiate*() method creates a prefab instance and initializes values such as *name*, *tag*, *layer*, assigns a personal mesh, orbit, and more. If the planet has rings, then they are created similarly.

When creating asteroids, the scheme is similar, but you need to arrange them not in a proportional way, but chaotic.

```
for (int i = 0; i < asteroidCount; i++){
  OrbitData orbitData = new OrbitData{
    eccentricity = Random.Range(0.01f, 0.04f),
    perihelion = Random.Range(2.2f, 3.6f),
    orbitalPeriod = Random.Range(3.5f, 6f),
    longtitudeOfAscendingNode = 80.7f,
    rotationPeriod = Random.Range(-0.1f, -0.01f),
    radius = Random.Range(0.005f, 0.01f),
    axialTilt = Random.Range(0, 30)
  };
  orbitData.perihelion *= Scales.au2mu;
  orbitData.radius *= Scales.au2mu;
  orbitData.orbitalPeriod *= Scales.y2tmu;
  orbitData.rotationPeriod *= Scales.y2tmu;
  GameObject asteroid =
    Instantiate(asteroidPrefabs[Random.Range(0,
    asteroidPrefabs.Length)]);
  asteroid.transform.parent = asteroidHolder.transform;
  asteroid.transform.Find("Mesh").localScale =
    Vector3.one * orbitData.radius;
  asteroid.GetComponent<Body>().Parameters = orbitData;
}
```

That is, for each asteroid an orbit is created, for which the parameters are selected at random. We create a copy of the prefab and give it a mesh.

Let's look at the algorithm for increasing and decreasing the size of objects when going from real to enlarge. There are three methods to do this: *GrowUp*(), *GrowDown*(), *DoResize*(). It is clear that the first two cause the court into which two parameters are passed: *realScale* and *largeScale*. These methods differ only in the order of these parameters.

The *DoResize*() method with *Lerp* looks for the middle between the starting point and the one you want to reach. The size of each frame changes over a predetermined period.

```
private IEnumerator DoResize(float from, float to){
  float percent = 0f;
  float speed = 1f;
  Vector3 initial = Vector3.one * from;
  Vector3 desidred = Vector3.one * to;
  while (percent < 1f){
    percent += speed * Time.unscaledDeltaTime;
    transform.localScale = Vector3.Lerp(initial,
      desidred, percent);
    yield return null;
  }
  transform.localScale = desidred;
  if (ResizeComplete != null) ResizeComplete();
}
```

Equally interesting is the *CamController* method, which controls the main camera and directs the program's actions when you click on the screen or with a computer mouse. Some variables play the role of boundaries.

The first two limit the speed of rotation of the system so that there are no too sharp movements on the *Y*-axis. The other two limit the angle that we can deviate along the *X*-axis, that is, the user will not be able to rotate the system in a spiral, but only within the required limits.

RotateCamera() is responsible for rotating the camera.

```
private void RotateCamera(Vector3 move){
  if(UnityEngine.EventSystems.EventSystem.current.IsPoint
erOverGameObject(fingerId))
    return;
  xDeg += move.x * xSpeed;
  yDeg -= move.y * ySpeed;
  yDeg = ClampAngle(yDeg, yMinLimit, yMaxLimit, 5);
  transform.rotation =
    Quaternion.Lerp(transform.rotation,
    Quaternion.Euler(yDeg, xDeg, 0), Time.deltaTime *
    rotationDampening / Time.timeScale);
  targetRotation.rotation = transform.rotation;
}
```

ZoomCamera() is responsible for scaling the camera.

To smoothly shift the camera, we use the *Lerp* method. We calculate from what point you want to get to and assign the current position to the desired value and equate the desired one to the current one. As we zoom in, we need to check the distance to the planet, that is, we cannot approach endlessly, only a certain distance.

When moving to some planetб the camera is fixed on it. This state is captured in the *isLocked* variable.

Touch controls are performed by the *HandleTouch*() method.

```
if (Input.touchCount == 1) {
  Touch touch = Input.GetTouch(0);
  ray = Camera.main.ScreenPointToRay(touch.position);
  mode = Mode.Rotating;
}
else if (Input.touchCount == 2){
  mode = Mode.Zooming;
}
if (DoubleClick(Time.time) && Physics.Raycast(ray, out
  RaycastHit hit, float.MaxValue) == true) {
    if (LockedTransform !=
      hit.collider.gameObject.transform.parent.transform)
      LockObject
    (hit.collider.gameObject.transform.parent.transform);
}
```

If one-touch enters the input, then you need to rotate the system or object, if two – enlarge or reduce the size. According to this variable mode is assigned the appropriate value: rotating or zooming. If a double click is made, then the planet is clicked and we can see it close, it is fixed. And then with the switch, depending on the value obtained in mode, the camera is shifted to the appropriate position.

The *HandleMouse*() method has a similar algorithm but has an excellent implementation. We also check which button is pressed or the scroll wheel is engaged and set the appropriate value in mode.

As described above, you can double-click to go to a planet or satellite. Already in the switch is further processing.

If no action is taken, the camera does not move. If you try to rotate the object, the coordinates of the cursor are read and the *RotateCamera*() method rotates the camera. *ZoomCamera*() method is called for scaling.

Figures 3 and 4 shows the relationships between classes responsible for implementing ARCore technology, localization, correct operation of different menus, and adjusting object data.

3 Results

We launch the application on a smartphone (fig. 5). Select the menu item AR. The first time you launch the application, you will be asked for permission to use the camera, you must confirm it. It is necessary to move the phone slowly so that the application finds a solid surface. The white spots along the entire found surface will be a sign of finding (fig. 6).

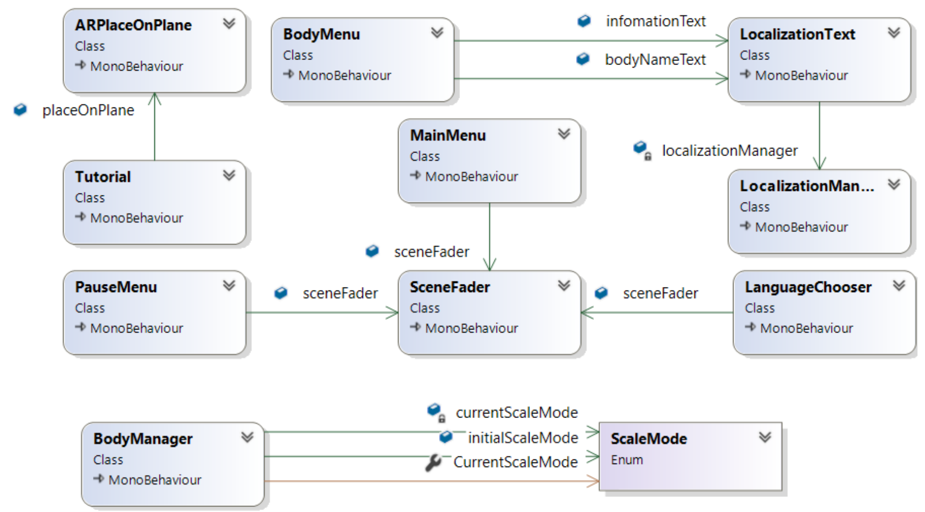


Fig. 3. Class diagram of the software application (part 1).

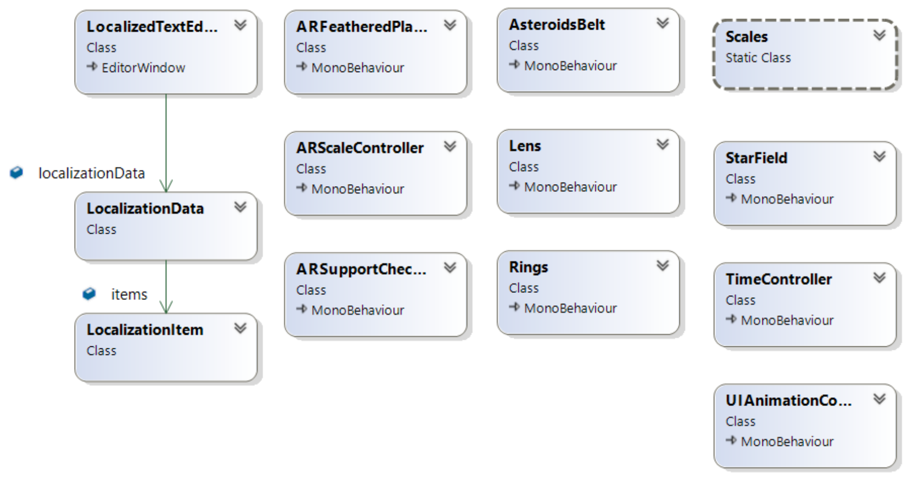


Fig. 4. Class diagram of the software application (part 2).

Then you need to click in the place where the center of the solar system – the Sun – will be. Depending on the selected point, the whole system will be located (fig. 7).

Let us try to walk, go around some planets, and get closer, for example, to Venus (fig. 8). As you get closer, the size of the object you are going to will increase, otherwise it will decrease accordingly. You can look at objects from above, below and from any side. To do this, just walk with the phone in hand.

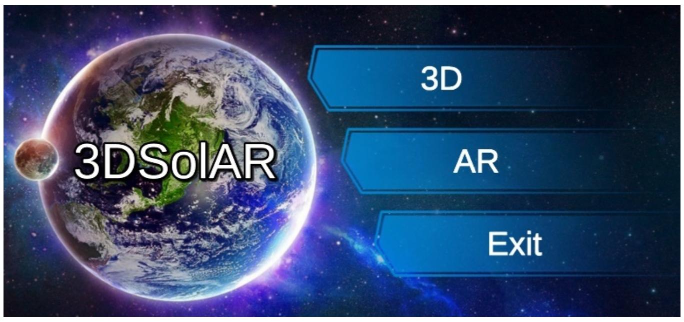


Fig. 5. The main menu of the application.

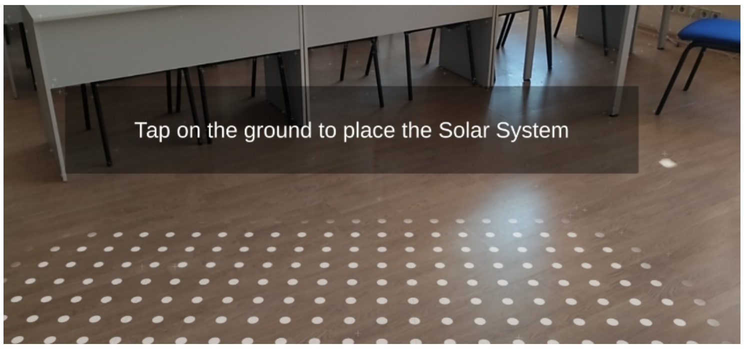


Fig. 6. The result of finding a solid surface.

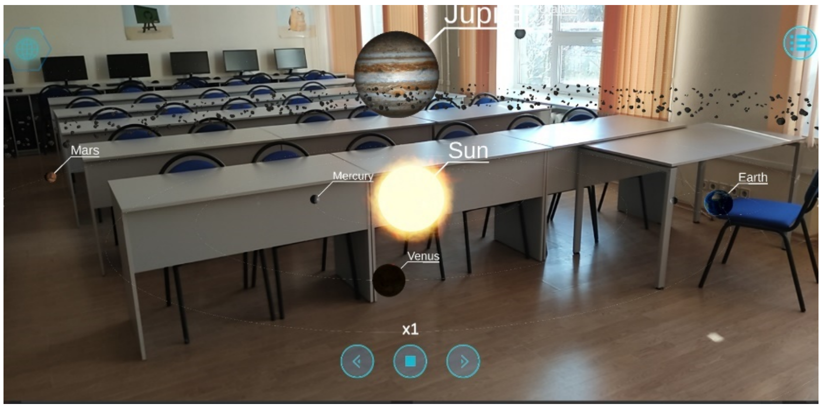


Fig. 7. System location.

In AR mode, you can read information about planets and satellites. You should click on the button from the top left and select the desired object from the list. We choose

Mercury (fig. 9). Again, the item Description is active by default.

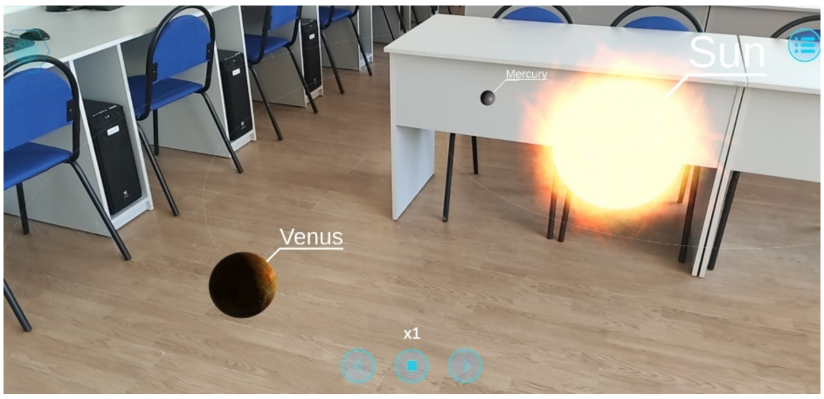
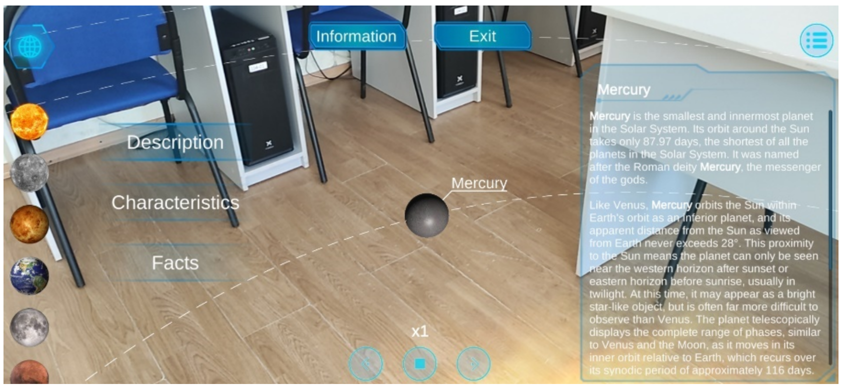


Fig. 8. The result of the approach of the camera to Venus.



Mercury

Mercury is the smallest and innermost planet in the Solar System. Its orbit around the Sun takes only 87.97 days, the shortest of all the planets in the Solar System. It was named after the Roman deity **Mercury**, the messenger of the gods.

Like Venus, **Mercury** orbits the Sun within Earth's orbit as an inferior planet, and its apparent distance from the Sun as viewed from Earth never exceeds 28°. This proximity to the Sun means the planet can only be seen near the western horizon after sunset or eastern horizon before sunrise, usually in twilight. At this time, it may appear as a bright star-like object, but is often far more difficult to observe than Venus. The planet telescopically displays the complete range of phases, similar to Venus and the Moon, as it moves in its inner orbit relative to Earth, which recurs over its synodic period of approximately 116 days.

Fig. 9. Description of Mercury.

Let's try to select other menu items “Characteristics” (fig. 10) and “Facts” (fig. 11). By the way, if the amount of information is too large and does not fit into the panel, there is a scrollbar on the side by which the text can be scrolled up and down.

If we want to close the information, all you have to do is press the Exit button (fig. 12). The peculiarity of the software application may be the fact that all the planets are represented in scale to the real size (fig. 13).

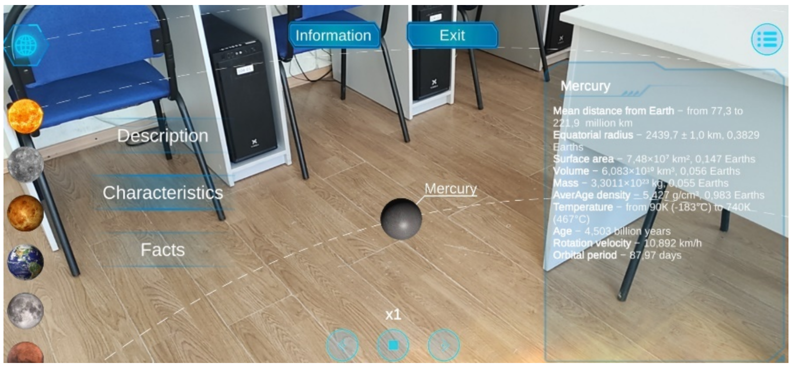
Selecting the 3D menu item displays the solar system in 3D (fig. 14).

Spreading two fingers up and down do zooming. Let us approach, for example, Venus. We see a display of brief information about the speed of its movement and the distance from the Sun (fig. 15).

Click the button at the top left. A list of planets appears. By selecting one of them, you can track its movement. Two additional Info and Exit buttons appear at the top of the screen.

By clicking on the “Information” button, you can select the menu item

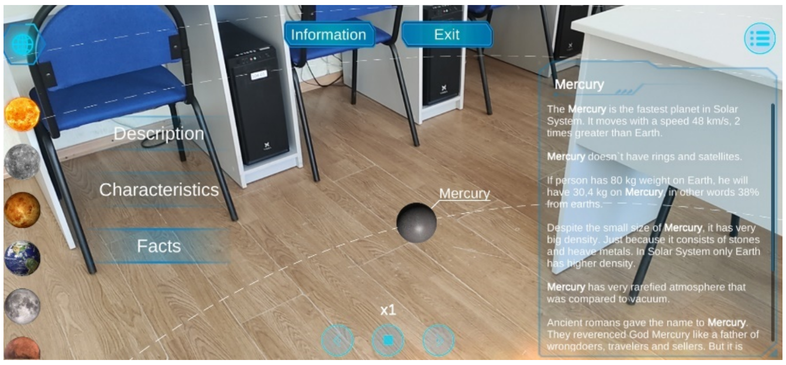
("Description", "Facts", "Characteristics") and read the relevant information (fig. 16). The description will open by default.



Mercury

Mean distance from Earth – from 77,3 to 221,9 million km
Equatorial radius – 2439,7 ± 1,0 km, 0,3829 Earths
Surface area – 7,48×10^7 km², 0,147 Earths
Volume – 6,083×10^{10} km³, 0,056 Earths
Mass – 3,3011×10^{23} kg, 0,055 Earths
AverAge density – 5,427 g/cm³, 0,983 Earths
Temperature – from 90K (-183°C) to 740K (467°C)
Age – 4,503 billion years
Rotation velocity – 10,892 km/h
Orbital period – 87,97 days

Fig. 10. Features of Mercury.



Mercury

The **Mercury** is the fastest planet in Solar System. It moves with a speed 48 km/s, 2 times greater than Earth.

Mercury doesn`t have rings and satellites.

If person has 80 kg weight on Earth, he will have 30,4 kg on **Mercury**, in other words 38% from earths.

Despite the small size of **Mercury**, it has very big density. Just because it consists of stones and heave metals. In Solar System only Earth has higher density.

Mercury has very rarefied atmosphere that was compared to vacuum.

Ancient romans gave the name to **Mercury**. They reverenced God Mercury like a father of wrongdoers, travelers and sellers. But it is

Fig. 11. The facts about Mercury.

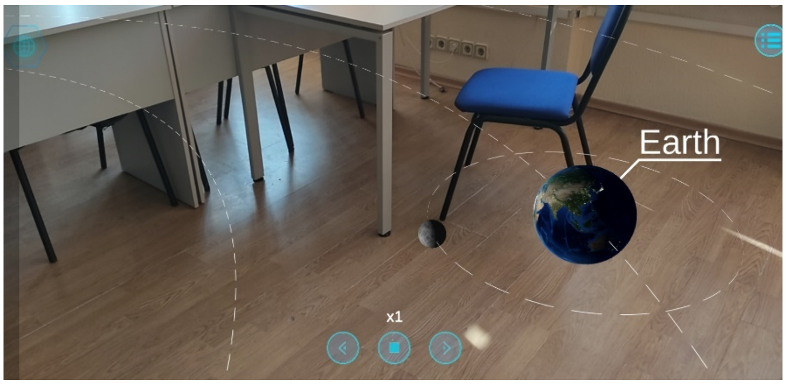


Fig. 12. Result of exit from information viewing mode.

To exit the viewer mode, you must press the "Exit" button, and then you can view all objects of the solar system from a distance.

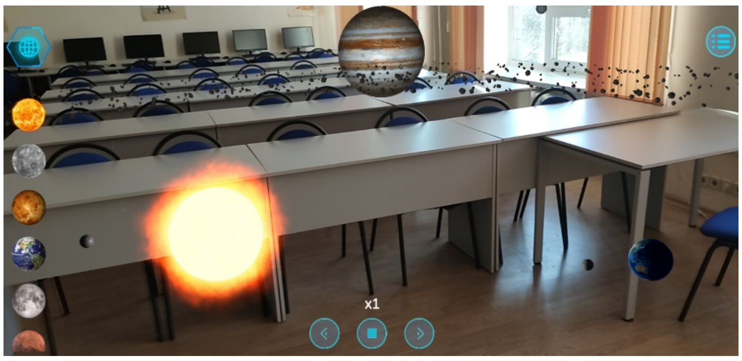


Fig. 13. Size of the Sun and planets.

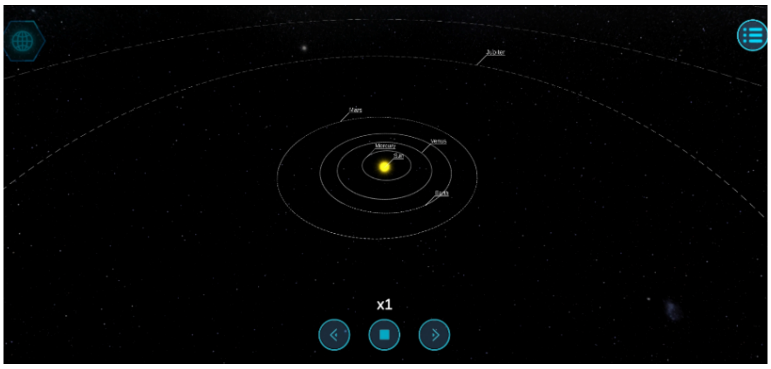


Fig. 14. The appearance of the solar system at startup.

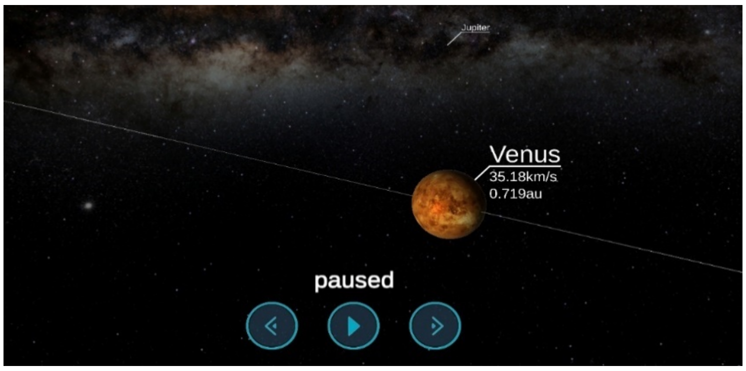
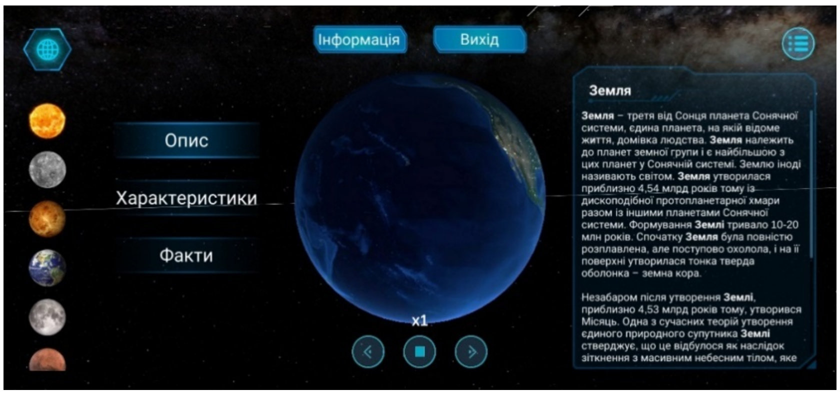


Fig. 15. Zoom in on Venus.

Settings button on the top and a settings bar will appear after clicking (fig. 17). Settings allow you to turn off music, orbits, and names, turn on Enlarged mode, and change the language from Ukrainian to English and vice versa.

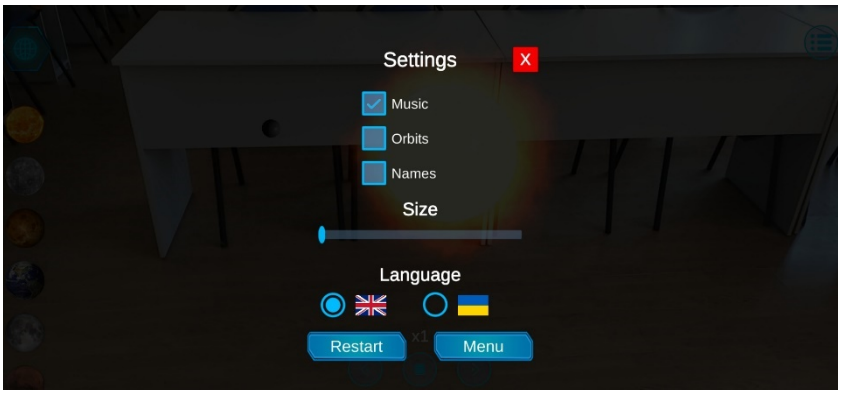


Земля

Земля – третя від Сонця планета Сонячної системи, єдина планета, на якій відоме життя, домівка людства. **Земля** належить до планет земної групи і є найбільшою з цих планет у Сонячній системі. Землю іноді називають світом. **Земля** утворилася приблизно 4,54 млрд років тому із дископодібної протопланетарної хмари разом із іншими планетами Сонячної системи. Формування **Землі** тривало 10-20 млн років. Спочатку **Земля** була повністю розплавлена, але поступово охолола, і на її поверхні утворилася тонка тверда оболонка – земна кора.

Незабаром після утворення **Землі**, приблизно 4,53 млрд років тому, утворився Місяць. Одна з сучасних теорій утворення єдиного природного супутника **Землі** стверджує, що це відбулося як наслідок зіткнення з масивним небесним тілом, яке

Fig. 16. Information about Earth.



Settings

- [✓] Music
- [ ] Orbits
- [ ] Names

Size

Language

● 🇬🇧 ○ 🇺🇦

Fig. 17. Settings.

If you go into the settings, then go to the main menu, and press the button "Exit" – the application will shut down.

4 Conclusions

The developed software application demonstrates the behavior of solar system objects in detail with augmented reality technology. In addition to the interactive 3D model, you can explore each planet visually as well as informatively – by reading the description of each object, its main characteristics, and interesting facts.

The model has two main views: Augmented Reality and 3D.

After analyzing the capabilities, disadvantages and advantages of existing platforms for the implementation of augmented reality in the software application, ARCore technology was chosen. After all, it combines rich functionality, the ability to implement the necessary ideas in this case and its optimized, well-established work.

Studying the principles of objects in the solar system was worked out different theories physicists and taken as a basis Kepler's laws.

To create 3D models, real parameters of objects were used, with the help of which the main thing was realized – the correct proportions in the sizes and velocities of objects and shapes and distances between the orbits of celestial bodies.

References

1. Cryogeneye: AR Solar System. https://play.google.com/store/apps/details?id=com.Cryogeneye.Solar (2018). Accessed 25 Oct 2019
2. Google Developers: ARCore API reference. https://developers.google.com/ar/reference (2020). Accessed 21 Mar 2020
3. Haranin, O.M., Moiseienko, N.V.: Adaptive artificial intelligence in RPG-game on the Unity game engine. CEUR Workshop Proceedings **2292**, 143–150 (2018)
4. Iatsyshyn, Anna V., Kovach, V.O., Romanenko, Ye.O., Deinega, I.I., Iatsyshyn, Andrii V., Popov, O.O., Kutsan, Yu.G., Artemchuk, V.O., Burov, O.Yu., Lytvynova, S.H.: Application of augmented reality technologies for preparation of specialists of new technological era. CEUR Workshop Proceedings **2547**, 181–200 (2020)
5. INOVE: Solar System Scope. https://play.google.com/store/apps/details?id=air.com.eu.inove.sss2 (2020). Accessed 21 Mar 2020
6. Kasinathan, V., Mustapha, A., Hasibuan, M.A., Abidin, A.Z.: First Discovery: Augmented Reality for Learning Solar Systems. International Journal of Integrated Engineering **10**(6), 149–154 (2018). doi:10.30880/ijie.2018.10.06.021
7. Kramarenko, T.H., Pylypenko, O.S., Zaselskiy, V.I.: Prospects of using the augmented reality application in STEM-based Mathematics teaching. CEUR Workshop Proceedings **2547**, 130–144 (2020)
8. Midak, L.Ya., Kravets, I.V., Kuzyshyn, O.V., Pahomov, J.D., Lutsyshyn, V.M., Uchitel, A.D.: Augmented reality technology within studying natural subjects in primary school. CEUR Workshop Proceedings **2547**, 251–261 (2020)
9. Morkun, V.S., Morkun, N.V., Pikilnyak, A.V.: Augmented reality as a tool for visualization of ultrasound propagation in heterogeneous media based on the k-space method. CEUR Workshop Proceedings **2547**, 81–91 (2020)
10. Nechypurenko, P.P., Stoliarenko, V.G., Starova, T.V., Selivanova, T.V., Markova, O.M., Modlo, Ye.O., Shmeltser, E.O.: Development and implementation of educational resources in chemistry with elements of augmented reality. CEUR Workshop Proceedings **2547**, 156–167 (2020)
11. Onepixelsoft: Solar System AR. https://play.google.com/store/apps/details?id=com.onepixelsoft.solarsystemar (2017). Accessed 21 Mar 2017
12. Pochtoviuk, S.I., Vakaliuk, T.A., Pikilnyak, A.V.: Possibilities of application of augmented reality in different branches of education. CEUR Workshop Proceedings **2547**, 92–106 (2020)
13. Shyshkina, M.P., Marienko, M.V.: Augmented reality as a tool for open science platform by research collaboration in virtual teams. CEUR Workshop Proceedings **2547**, 107–116 (2020)
14. Sin, K., Zaman, H.B.: Live Solar System (LSS): Evaluation of an Augmented Reality book-

based educational tool. In: International Symposium on Information Technology, Kuala Lumpur, 15-17 June 2010, pp. 1-6. IEEE (2010). doi:10.1109/ITSIM.2010.5561320

15. Unity – Manual: Unity User Manual (2019.4 LTS). https://docs.unity3d.com/Manual/UnityManual.html (2020). Accessed 21 Mar 2020
16. Unity Asset Store - The Best Assets for Game Making. https://www.assetstore.unity.com (2020). Accessed 21 Mar 2020
17. Wesoft: Solar System (AR), https://apkpure.com/solar-system-ar/com.wesoft.solarsystemdetailed (2018). Accessed 21 Mar 2020
18. Zhang, X., Fronz, S., Navab, N.: Visual marker detection and decoding in AR systems: a comparative study. In: Proceedings of the International Symposium on Mixed and Augmented Reality, 1 Oct. 2002, Darmstadt, Germany. IEEE (2003). doi:10.1109/ISMAR.2002.1115078