# Summary

[**Summary 1**](#_yrdqfrv5fe3k)

[**Annotation 2**](#_4s7xqx2l807t)

[**Menu scene 3**](#_3ix708vzxald)

[**Settings scene 4**](#_i93tgrejtkdd)

[**Main scene 5**](#_1a8511foa007)

[**Physics 6**](#_mohvf1xdem62)

[**Working process and what was not done 7**](#_c2fcigp2mrtn)

[**User Documentation 10**](#_rotvi8qozqi8)

## 

# 

# Annotation

The Solar System Simulator app offers a visual representation of how celestial bodies interact with each other in the Solar system based on Newton’s Gravitational laws.Users can also get some information about the Sun and planets orbiting around it. Users can also change the speed of the simulator. The app's design is kept simple, solely focusing on illustrating planetary motions without additional complexities.   
  
Program was written using Visual Studio and Unity.  
There are three scenes in the program. Each one has several c# files attached to it.

1. Menu scene
   1. Menu.cs
2. Settings scene
   1. SettingsScript.cs
3. Main scene
   1. SolarSystem.cs
   2. SpaceBody.cs
   3. CanvasChanges.cs
   4. CameraMovement.cs

There are also two texts files that were used in programming process

1. values.txt - text file with information about the celestial bodies (more accurately, planets)
2. speed.txt - text file for transferring simulator’s speed value from settings scene to menu scene.

Now scenes and c#files will be described more detailed.

# Menu scene

Menu scene has three buttons

1. Start - by clicking on this button user can start the simulator
2. Settings - by clicking on this button user can get to settings scene
3. Quit - by clicking on this button the user can close the simulator.

Code for all three buttons is written in c# file Menu.cs and is pretty simple. Each button has its own function (it’s attached to the button in the unity file).

In Menu.cs were used two Unity libraries:

1. UnityEngine - basic Unity library, that allows to write code for Unity in VS
2. UnityEngine.SceneManagement - Unity library, that allows load/unload scenes

# Settings scene

Settings scene has one button “Back to menu”, text, input field and hidden error message text. Their changes, when a user interacts with them, are described in the SettingScript.cs file.

The user can put any float number less than 100 in the input field. The restriction is made due to the fact that the higher the speed the less accurate simulator will be.

Button “Back to menu” brings the user back to the menu scene. Its behavior is written in the ClickOnBack() function.

If a user puts the wrong type of data in the input field or float number 100 or more, they won’t be able to get back to the menu scene. Instead error message text will appear and wait till the user changes the input to correct one.   
  
Float number from the input field is stored in the text file speed.txt, so it can be accessed from other scenes.   
  
There are several functions besides the ClickOnBack() function:

1. CheckingInput() checks the validity of the input
2. ChangingInput() makes the error message text disappear when the user is changing the data in the input field
3. Start() is basic Unity function that is called first after scene load

# Main scene

Main scene showcases how celestial bodies integrate with each other based on Newton’s gravitational laws.

On the screen the user can see 9 celestial bodies - the Sun and eight planets. All celestial bodies have their real life velocity in km/s and distance from the sun in 10\*\*5 of km. The radius of the planets and the sun was chosen, so it would be more pleasant for the user. Masses of the celestial bodies are written in Earth masses (one Earth mass = 5.9722×10\*\*24 kg)

The choice of the value of gravitational constant will be discussed in the “Physics” part of the documentation.  
  
The user can also see the text that instructs how to follow the celestial body. In the following state some additional text appears: instruction, how to leave the following state, instruction, how to change following the celestial body to the next one and brief information about the celestial body.

In the upper right corner the user can see a red button that closes the game.  
  
Main scene uses four c# files to describe behavior of the objects

1. SolarSystem.cs is responsible for setting up and simulating the behavior of space bodies within the Solar System Simulator. It reads configuration files for initial settings, creates space body objects, and updates their positions and velocities over time to create a dynamic and realistic representation of celestial motion.
2. SpaceBody.cs defines the behavior of individual space bodies within the Solar System Simulator. It calculates their velocities based on gravitational interactions, updates their positions, handles click events to focus the camera on a selected space body, and supports time scaling for the simulation's overall velocity.
3. CameraMovement.cs controls the camera's movement and interaction within the Solar System Simulator. It allows users to navigate and follow space bodies, adjust camera settings, and transition between different bodies for observation.
4. CanvasChanges.cs controls the behavior of the UI canvas in the Solar System Simulator app. It manages the visibility of various UI elements, displays information about space objects, and handles the application quit process through the UI.

# Physics

The simulator works based on Newton’s gravitational laws. They were used instead of Kepler’s laws, because Kepler’s laws do not use masses of celestial bodies, therefore Newton’s gravitational laws are more accurate.

Another possible solution was to use Einstein’s general theory of relativity however in such simple cases like solar system, both general relativity and Newton’s laws will show almost the same calculations, while general relativity is much more complex to implement.

All celestial bodies have their real life velocity in km/s and distance from the sun in 10\*\*5 km. The radius of the planets and the sun was chosen, so it would be more pleasant for the user. Masses of the celestial bodies are written in Earth masses (one Earth mass = 5.9722×10\*\*24 kg)

The gravitational constant in the Simulator is 3.98584628. It was chosen like this through calculations and simplifying the Newton’s formula.

Newton’s formula looks like this:

F = G\*(m1\*m2/r\*\*2)

where

m1 - mass of one celestial body

m2 - mass of another celestial body

r - distance between the centers of their masses

G - the gravitational constant.

This formula is used to calculate the gravitational force between two celestial bodies. In the Solar system there are 9 of them (Sun and 8 planets), so the first algorithm worked as following

1. Choose one celestial body A
2. Calculating gravitational force between celestial body A and another celestial body in the system
3. Add calculation to the force that is applied on body A
4. Repeat 2. and 3. with all celestial bodies in the system A

After that we have force that is applied to celestial body A. Forces for all other celestial bodies are calculated through the same algorithm.

After calculating the forces it is possible to update bodies’ velocity using acceleration (F = ma => a = F/m) and previous velocity of the body.

After that velocity is used to calculate the next position of all the bodies.

However i noticed that it is possible to simplify the formula, so for celestial body A it looks as following:

a = G\*m2/r\*\*2  
where

a - acceleration of the body A

m2 - mass of another body

r - distance between another body and body A

Using this simplified version program does not have to calculate force and then acceleration, it calculates acceleration directly.   
  
Real gravitational constant is 6.674×10\*\*(−11) N⋅m\*\*2/kg. However due to the fact that we used mass of celestial bodies in Earth masses instead of kgs and r not in meters but 10\*\*5 km, the gravitational constant was chosen in order to make it possible for celestial bodies to orbit with their real life velocities in km/s.

# Working process and what was not done

During working on this project I encountered a lot of different problems, mainly due to me not being experienced in working with Unity. I successfully resolved most of the issues, however, it required a lot of time and effort.

One of the problems I didn’t know how to solve was a bug with Universal RP that can render celestial objects. I tried to implement it several times but every time it crashes the Unity file. Due to this fact the design of the celestial bodies is very simple.

# User Documentation

To open the program from, they need to download an archived folder from GitHub, decompress it and open file “SolarSystemSimulatorr.exe”.

When the user opens the program, they see the menu scene with three buttons. Button “start” starts the simulator, “settings” gets the user to the settings scene and button “quit” closes the program.  
  
In the settings scene the user can change the speed of the simulator, making celestial bodies move slower or faster. The speed value needs to be float and no more than 100, otherwise the user will not be able to go back to the menu.

In the main scene the user can use ASDW keys to move the camera in different directions. The user can also use arrows down and up to change the size of the camera.   
  
The user can enter the “follow” state if the celestial body is clicked on or using the F key. In this state the user can press the spacebar to change the followed celestial body.

In “follow” state ASDW keys do not work due to the fact that the camera is always focused on the celestial body. However, arrows can be used to change the camera size.

If Esc is clicked, the user will get back to the menu scene.

If the red button in the right upper corner is clicked, the program will be closed.