2021-2-IT02-KA210-SCH-000048086
Erasmus+

PLAY

PLAY

PROTECT, LEARN, ENGAGE AND ENJOY

Toolkit

Paidea
A.M.E.F.E
asociación malagueña de
educación y formación europea

Module 1

Unit 1 - Project overviews and objectives..... 2
Unit 2 - Digital Competence Framework for Educators (DigCompEdu)..... 6
Unit 3 - Gamification..... 10
Unit 4 - The educational value of using an approach centred on technology..... 13
Unit 5 - Project results..... 15

Module 2

Unit 1 - Unity 3D Platform: uses and potentialities..... 16
Unit 2 - Interface..... 19
Unit 3 - How to create codes using the C# programming language..... 52
Unit 4 - Input, physics, user interface, scenes and rendering.....68
Unit 5 - Practical guide using codes of the PLAY video game.....79

Module 1

Unit 1 - Project overviews and objectives

Play - protect, learn, engage and enjoy is a project funded by the European Union under the Erasmus+ program, the EU programme in the fields of Education, Training, Youth and Sport.The partnership is made of an Italian and a Polish school, a Spanish Association of European Education and Training and Paidea, an Italian edtech company. The idea stemmed from the will to make school an environment where students can freely experiment and build their knowledge using a practical approach through the support of technology. Things that are taught in school should be directly linked to the world we live in and should enhance the development of digital skills that are necessary in a technological and constantly evolving society, following the directives of the Digcomp.

The PLAY project aims to promote the digital skills of students aged from 11 to 14 years old and to raise awareness of the importance of protecting our planet by directly engaging them in the development of a video game through the Unity 3D platform and by making them learn the C# programming language.
One of the Sustainable Development Goals envisaged by the 2030 Agenda highlights the need to take urgent action to combat climate change and its impact. PLAY uses a ludic approach to increase students' awareness of the importance of modifying our lifestyle and to enhance the development of digital competences that represent an important skill set in the world of work. After specific training, the students developed a 3D video game where it is possible to interact with different environments to understand the main causes of the waste of resources and pollution as well as to get to know sustainable alternatives that will contribute to the reduction of our environmental impact.

Unity is a cross-platform game engine primarily used to develop video games and simulations for PC, consoles, mobile devices and websites. It is possible to programme in C#, Java, UnityScript or Boo, a language similar to Python. In the project the selected programming language has been C# as it is very easy to use, so it's suitable for beginner developers.

They have developed the video game with the supervision of expert programmers and trainers. More specifically, the activity has foreseen the following phases:

- Training and practice. In the first stage the pupils have been trained on how to use Unity and on how to programme in C# (together with the teachers); after being provided with a theoretical background, they have practised how to write efficient code.
- Development plan. Several brainstorming sessions have been organised, where students have been able to discuss their ideas and cooperate. After that, they have had to carry out research on the main causes of climate change and on greener solutions. This led to the selection of six settings, which represent the interactive background and levels of the game.
- Production. This is the stage of the actual development of the game.
- Testing and validation. When the output has been completed, the final version has been tested by the partner schools' students that have not taken part in the project.

The teachers have actively participated in all the stages of production of this activity, which enabled them to learn innovative teaching methods and also to develop new skills.

The main objectives of the project can be summarised as follows:

1. promoting the development of digital and programming skills in young students that will make them competitive in the labour market;
2. raising awareness, among students and the community in general, of the causes of global warming and of the importance to fight climate change by modifying our habits;
3. promoting the development of innovative and digital teaching methods by providing teachers with the necessary knowledge and competence, in order to increase the level of motivation of students and their academic performance.

By developing the videogame, the students have had the opportunity to practically experiment with programming and have acquired digital competences that will be useful in their future professional life. Computer programming has an integral role in our world. Learning the fundamentals of programming can give students a competitive edge in this technology-driven world. Programming skills are also important for learning to innovate, create eco-friendly solutions for global problems. The students have at the same time broadened their knowledge on such a crucial topic and have become responsible individuals that will take on an active role in the fight against climate change. Becoming aware of such a threat and helping others understand the consequences of our actions has had an important impact on the participants' lives, making them more responsible and conscious about the environment and their role in society. The activities envisaged were also intended to promote inclusion and teamwork. Of course some students were more digitally proficient and others were more interested in the environment and sensitive to this topic. By working together they had the possibility to learn from each other.

In the new learning programmes it is necessary to use suitable methodologies, for example gamification or game-based learning, to engage students, to stimulate their creativity, to tailor learning programmes and to make the contents more appealing. Each one of these aspects represents a key variable that teachers should take into account to reach pedagogical and learning objectives. This demands that the teachers possess a significant Teachers Digital Competence: this empowers the teacher for the use of technology not only as support for their existing practices, but also to use it to rebuild educational and training settings.

Unit 2 - Digital Competence Framework for Educators (DigCompEdu)

The European Framework for the Digital Competence of Educators (DigCompEdu) is a scientifically sound framework describing what it means for educators to be digitally competent. It provides a general reference frame to support the development of educator-specific digital competences in Europe. DigCompEdu is directed towards educators at all levels of education, from early childhood to higher and adult education, including general and vocational education and training, special needs education, and non-formal learning contexts.

Digital competences are necessary in learning and at work and make us active members of society. In relation to education, as important as understanding the competences themselves is knowing how to develop them.

What are digital competences?

Digital competence is one of the elements of the eight key competences and refers to the conscious and critical use of the entire spectrum of digital technologies enabling the acquisition of information, communication and basic problem solving in all aspects of life.

What is the DigCompEdu framework?

The frame distinguishes 6 coverage areas:

1. Professional commitment - the use of effective communication technology, cooperation and communication development.
2. Digital resources - use and use of resources, creation, evaluation and sharing of resources.
3. Teaching and learning – using different methods and tools in teaching and learning.
4. Evaluation - control of various methods and tools for consideration and feedback.
5. Supporting learning – using manufacturing technologies to empower and make learning environments more accessible, exclusive, inclusive and learning-focused.

6. Supporting the competences of learners - supporting and facilitating the development of their competences.

The European Teachers' Digital Competence Framework is a standardised framework that provides a reference and guide for working with teachers' digital competences. It identifies 6 useful categories of digital learning activities that are broken down into specific competencies.

The framework aims to show how digital technologies can be used to support professional development in education and training. The emphasis is not only on technical skills, but primarily on the context of teaching and learning. The framework itself aims to "provide a common language and logic as a starting point for developing, comparing and discussing different instruments for teacher digital competence development at national, regional or local level".

In order to better understand the nature of digital competences, the European Commission has prepared the European Digital Competence Framework for Citizens (DigComp), which has been divided into five areas: information and data analysis; communication and cooperation; digital content creation; security; and problem solving. In total, they cover 21 competencies.

The continuous development of digital competences concerns both children and teachers. Everyone, regardless of age, if only acquires the skills of new technologies, will facilitate their functioning in modern society.

It is necessary to train children about their responsibility for implementation and preparation for life in the modern world, where the main message is that the media and technologies are developing rapidly. It is also necessary to improve teachers' training in the field of teaching with the use of modern technology in such a way that they purposefully support one conducted didactic process, supporting the increase in the effectiveness of teaching and learning. Recommendations of the European Parliament and the Council and many legal acts on education evidence for educational activities focused on the development of scientists studying them for life in modern society. One of these competences is digital competence.

For additional information please visit the following webpage: https://joint-research-centre.ec.europa.eu/digcompedu_en

Unit 3 - Gamification

There are many definitions of **gamification**. According to Brian Burke (research vice president at the technology company Gartner and expert in enterprise architecture) “Gamification is the use of game mechanics and experienced design to digitally engage and motivate players to achieve their goals”. Oxford dictionary defines Gamification as “the application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity,[...] to encourage engagement”. According to Cambridge dictionary, gamification is “the practice of making activities more like games in order to make them more interesting or enjoyable”. According to Yu-kai Chou (an Author and International Keynote Speaker on Gamification and Behavioral Design) there are 8 core drives that make games engaging for people:

1. **Epic meaning & Calling** → You are part of something bigger than yourself.

2. **Development & Accomplishment** → You are motivated because you feel you are improving and achieving mastery.

3. **Empowerment of creativity and Feedback** → Giving people even some simple elements like LEGO® building blocks to people and letting them free to find their own way to combine them is a very engaging process. As our students did when they were asked to create some “puppets” using plastic and paper waste material (*photos*).



4. **Ownership & possession** → When you feel you own something, you want to improve it, you want to protect it and you want to get more.

5. **Social Influence & Relatedness** → We are influenced by what people around us do.

6. **Scarcity & Impatience** → It relates to the fact that sometimes you want something only because you can't have it.

7. **Unpredictability & Curiosity** → If you don't know what's going to happen next you're always thinking about it.

8. **Loss & Avoidance** → If you do something you want to avoid loss, because none wants bad things to happen.

In Yu-kai Chou's opinion everything we do is based on one or more of these core drives. According to other studies games make kids smarter, because video games fundamentally present a continuous process of learning to users. They are constantly evolving and moving forward. Games are wired to produce a particular reaction in people. A game represents a challenge and when you overcome it dopamine is released in your brain, so we have learning improvement, multitasking increases connection and a strong dopamin loop in the brain and this produces an intrinsic reinforcement.

Since 2010 Gamification has become a methodology in education all around the world. Studies show that transforming educational goals through exciting challenges, using video game models, for example using progress badges or giving the possibility of seeing performance graphs, satisfies the need of people for competence in a specific activity and increases the perceived value of the task. Gamification doesn't mean to change the lesson into a game but to introduce videogames element into education. Videogames can stimulate our motivation, interest, creativity, sense of belonging and happiness in people. All these feelings are resources that we can spend in our daily activities.

So if we introduce videogame rules and schemes in all tasks we have to do in our everyday life we could complete them with more motivation and satisfaction, having better results.

Unit 4 - The educational value of using an approach centred on technology.

Integration of information technology resources into learning has numerous advantages: in particular, it provides alternative ways of learning, allows the acquisition of cognitive skills and knowledge that are important for lifelong learning and above all promotes motivation. Many of today's high-demand jobs were created in the last decade, according to the International Society for Technology in Education (ISTE). As advances in technology drive globalisation and digital transformation, teachers can help students acquire the necessary skills to succeed in the careers of the future. The COVID-19 pandemic is quickly demonstrating why online education should be a vital part of teaching and learning. By integrating technology into existing curricula, as opposed to using it solely as a crisis-management tool, teachers can create immersive and engaging learning experiences. The effective use of digital learning tools in classrooms can increase student engagement, help teachers improve their lesson plans, and facilitate personalised learning. Technology provides students with easy-to-access information, accelerated learning, and fun opportunities to practise what they learn. It enables students to explore new subjects and deepen their understanding of difficult concepts, particularly in STEM disciplines.

The main benefits of using technology in education can be summarised as follows:

- Increased Collaboration and Communication. Educational technology can foster collaboration. Not only can teachers engage with students during lessons, but students can also communicate with each other. Through online lessons and learning games, students get to work together to solve problems. In collaborative activities, students can share their thoughts and ideas and support each other.
- Personalized Learning Opportunities. Technology allows 24/7 access to educational resources. It can be used to tailor learning plans for each student. Teachers can create lessons based on student interests and strengths. An added benefit is that students can learn at their own pace and review material to get a better understanding of essential concepts.
- Curiosity Driven by Engaging Content. Through engaging and educational content, teachers can spark inquisitiveness in pupils and boost their curiosity, which research says has ties to academic success. Creating engaging content can involve the use of AR, videos, podcasts and of course games.

The introduction of technological elements and video games in the learning processes in educational environments is increasingly present in the debates on how to improve the experience of our young people in a context where the digital is being incorporated into more domains of our lives. Educators have been using games to teach in this way at least since 1971, when three student teachers created the classic educational game The Oregon Trail for use in a U.S. history course. Early arguments for this approach often centred on the idea that "video games are fun, and as such, provide an intriguing prospect for coercing some children to learn" (Silvern 10). The fact that games are effective at motivating players has remained a fundamental rationale for teaching with content-aligned games. Prensky describes students who came of age in the internet era as "digital natives" who naturally learn better while using technology (Prensky, "Listen to the Natives" 9-13). Studies support the idea that 21st-century youth have a strong affinity for technology, particularly video games and communications networks (Jones et al 6, Lenhart et al 2-3). Other writers attribute the appeal of games to their engaging mix of factors including challenge, fantasy and interactivity (Malone 162; Owston 978). Video

games can be a very stimulating learning tool for young people in schools. In addition to addressing certain curricular content, they can support the development of certain skills necessary to promote good habits. This was the exact objective of the project: finding an effective way to raise awareness among young students of the urge to preserve our planet by directly engaging them and making them develop technical and soft skills.

Unit 5 - Project results

Students involved in the project approached programming language by using Scratch and then Unity to create the videogame. This helped them to enhance their digital skills; they increased their interest and improved their knowledge in Global Warming, pollution and eco-sustainability. They were stimulated to find greener and sustainable solutions to combat global warming and pollution. Creating a video game that included study topics such as global warming, eco-sustainability and pollution fits into that strategy known as gamification. This is not only a strategy used in school, but also in working contexts to stimulate people to find creative solutions to solve problems and to enhance students' or workers' motivation for reaching a specific goal.

During mobilities students improved their knowledge of foreign languages (English), they knew the habits and culture of another country increasing their mutual understanding, empathy and the acceptance of diversity. Students and teachers visited different landscapes, exchanged their knowledge and competence and improved their ability to work in a team and to cooperate. This project helped students and teachers to develop a network of relationships with local educational institutions and with schools of other countries that will last beyond the end of the project.

Module 2

Unit 1 - Unity 3D Platform: uses and potentialities

INTRODUCTION.

The advance of technology is amazing. Who could have imagined 100 years ago what we are capable of doing today with the support of technological tools.

One of these tools is Unity 3D, which gives us the possibility to create real world spaces. We would like to emphasise that, after reading this toolkit, the end users will be able to build their own virtual tour of the scenario of their choice, as long as they let their imagination run wild and thus give their own artistic touch to the desired project.

The project presented in this toolkit was designed for people who do not have any knowledge in Unity 3D so, if you want to learn it from scratch you are just in the right place, we will start from the basic configuration of the development environment in Unity 3D, what each button and each application menu is, ensuring prior knowledge of these utilities, before starting to develop the virtual tour. It will be made clear in the handling of the dimensions in Unity 3D, because as its own name indicates, it handles the 3 dimensions (x, y, z) and when one does not have practice in the handling of the program it can be something tedious, because it is a question of habit to learn to be unwrapped with these dimensions. Having prior knowledge of the tool, we will begin with the creation of our project and if we are still not clear about some buttons, the doubts will be clarified as we progress, and as we practise and gain experience in the handling of Unity 3D. When the creation of the first space or virtual tour is completed, the students will be able to create their own environment. One cannot imagine how powerful this tool is and how easy it is to use, let alone how easy it is to create a virtual tour with Unity 3D, so you are invited to develop this tour step by step so that you can achieve your desired goals.

UNITY 3D.

Unity 3D is a tool to develop mainly video games, few know how to use it and you don't need to be an expert in programming to be able to use it, because with a few small

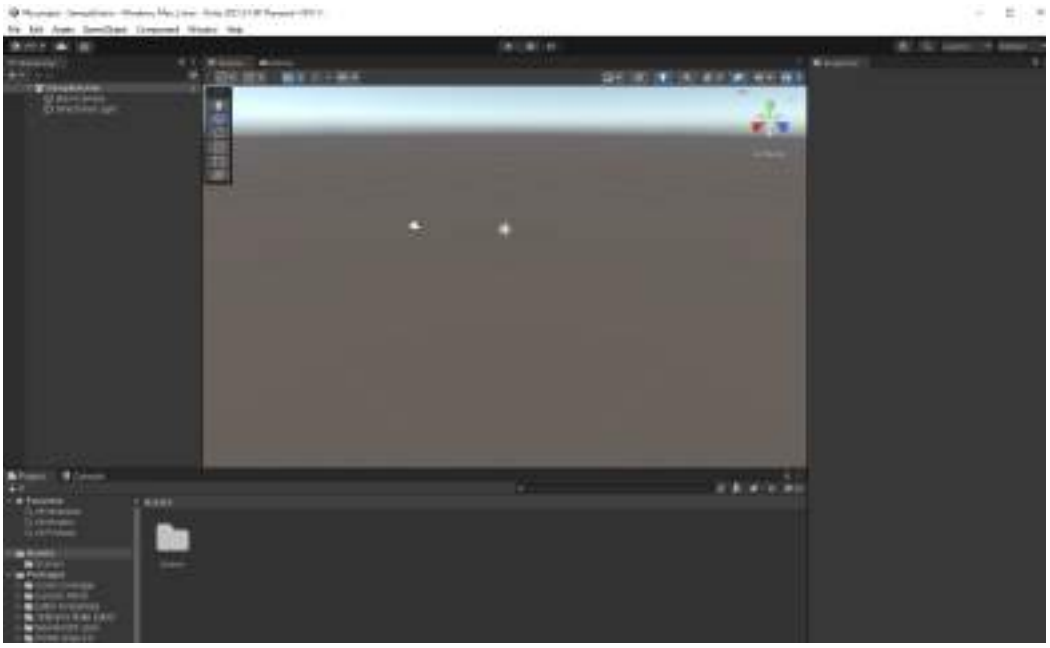
notions of the required language, you can structure any interactive game. "Unity 3D, a 3D graphics engine for PC and Mac, is used to develop games, interactive applications, visualisations and 3D animations. Unity has support for platforms such as PC, Mac, Nintendo, Wii, Iphone, Android and the web using its "Unity web player" plugin. Unity is an application created by Unity Technologies which "was founded in 2004 by David Helgason (CEO), Nicholas Francis (CCO), and Joachim Ante (CTO) in Copenhagen, Denmark after their first game, GooBall, was unsuccessful. The three recognised the value of the engine and development tools and set out to create an engine that everyone could use at an affordable price. Unity Technologies has received funding from the likes of Sequoia Capital, Capital WestSummit and iGlobe Partners. Unity's success has come in part due to its focus on the needs of independent developers who can neither create their own game engine nor the necessary tools or licensing to fully utilise the options that become available. The company's focus is on "democratising game development", and making the development of interactive 2D and 3D content as accessible as possible to as many people around the world as possible". With the rise of the iPhone in 2008, Unity was one of the first engines to start supporting this platform and today, Unity is being used by 53.1% of developers according to a survey conducted by Game Developer mobile and social technology, to create hundreds of games for Android and iOS devices. Unity has a visual editor to be able to create the games in it, because all the content of the game is built from this editor and the way the objects behave are programmed using a script language (JavaScrip); the above gives us to understand that you don't need to be an expert in languages like C++ to be able to develop a game or an animation with Unity 3D. Unity is structured by managing and creating scenes for the development of the desired application, a scene can be any part of the game or animation, either a level of the game or a certain area. You start with a blank space in which you can shape whatever you want to create using the unity tools. This unity engine also includes a terrain editor, where you can sculpt the shape of the terrain using the visual tools offered by unity, you can paint, texture, add grass, place trees or similar, or even import other materials from other development engines. Unity is accessible to any type of public, as it is developed in several versions, free and professional, both have great advantages when developing what is required, however, the most complete version is the

professional one, but it should be clarified that this one the free version has a cost that not everyone can afford and if you are someone who is just getting familiar with the tool, the free version is more than enough for the time being. Besides Unity 3D, there are other tools on the market, which may even be more famous, such as UDK, Epic Games or CryEngine. However, Unity 3D has a great advantage over these and that is that we are not obliged to develop on Windows operating systems, as Unity 3D also has a version for Mac operating systems. Unity also provides the facility not only to import terrains, but also 3D models, textures, sounds, etc. with just a few clicks, which can be used at any time during development.

Unit 2 - Interface

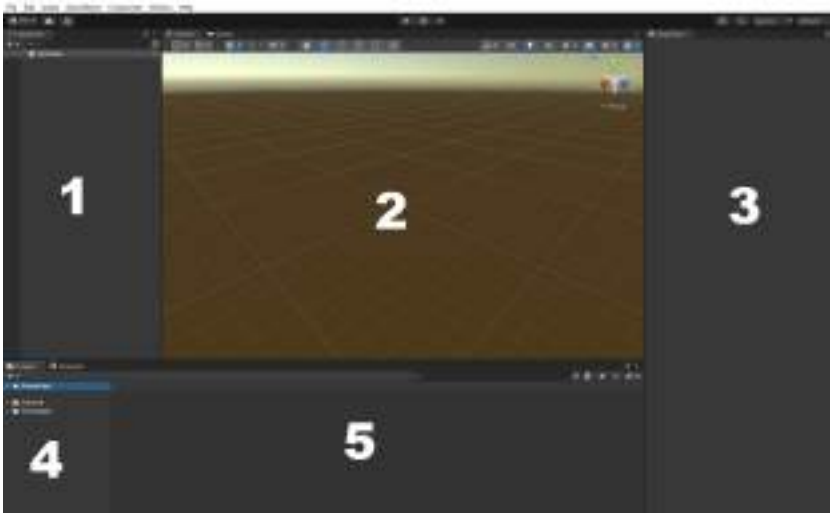
1. FIRST ACCESS

There are two versions of Unity, a free version and a professional version. This manual will be based on the functionalities accessible to beginners with the free version. The free version can be found on the Unity website. Once the application is installed, we can access a demo where we can see the capabilities of Unity and allow new users to discover the functionalities it offers by studying the creations of its developers.



2. USER INTERFACE

In this part of the manual we will look at what the Unity user interface is made up of. There are mainly 5 areas of the Unity interface, listed in the image below:



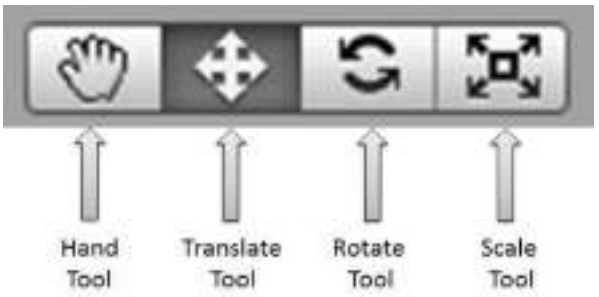
- Hierarchy view. Where they will have all the objects in the current scene.
- Scene View This is the area where we visually construct each scene. The easiest way to use the scene view would be to drag objects from the object management view to the scene view which will place the object in the scene; you can then position, scale and rotate it without leaving the scene view. The scene view is also the place where you edit terrains (sculpting them, painting textures and placing elements), place lights and cameras and other objects.
- Inspector View. This view has several functions depending on what the user selects, whether they want the characteristics of an object or terrain configuration. If you select “camera” then it will show the properties where you can customise features such as rotation, position or scale.
- Project view or object management. This is the library for our project, similar to a library of tools and/or objects. You can import 3D objects from different applications into the library, you can import textures and create other objects that you can store for use in your project. A normal project will contain several scenes and a large amount of assets, so it is necessary to structure the library in different folders that will make the assets organised and easier to use.
- Search view. It is used to locate objects that we need to use in the realisation of the project.

3. APPLICATIONS MENU

The Unity options menu at the top left of the screen is shown below. Throughout the manual, the utilities of each of the sections of this menu will be shown.

3.1 CONTROL BUTTONS

Below the display options you will see a row with 4 buttons that you can use Q, W, E, R to toggle between each of the controls, which are detailed below:



- **Hand Tool (Q):** This control allows us to move around in the scene view.
o ALT will allow us to rotate.
o CTRL will allow us to zoom in and out
o SHIFT increases the speed of movement while using the tool.

- **Translate Tool (W):** Allows us to move any selected object in the scene in the X, Y and Z axes.

- **Rotate Tool (E):** Allows us to rotate any selected object in the scene.

- **Scale Tool (R):** Allows us to scale any selected object in the scene.

When using the translation, rotation or scaling tools we will see a gizmo around the selected object with lines for each of the axes. We can use this gizmo to perform translation, rotation and scaling operations.

3.2 PLAYBACK BUTTONS

In Unity you can run your game without leaving the editor, which is a boon for designers who are building levels and developers who are adding new game mechanics

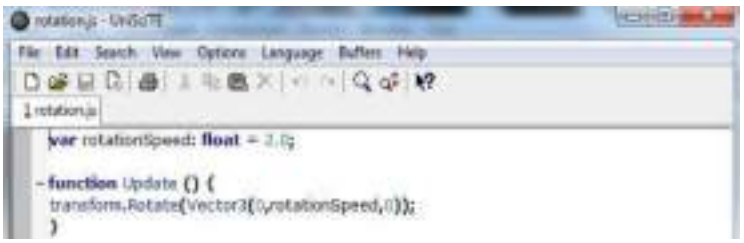
The image shows the playback controls, which are located at the top of the editor. You can enter a game view at any time by pressing the play button to see the status of the project (first from the left), pause using the pause button (centre) or skip forward using the right button. You can play from the game view or extend it to full.

3.3 SCRIPTS

Scripts allow the definition of the game logic. Throughout the tutorial you will see several simple scripts such as defining a movement, turning lights on or off. The operation is very simple, first you define the desired behaviour in a script and this is added as a component to the object for which you want to assign that behaviour, you can even add a script to an empty object. There are 3 basic functions for scripts:

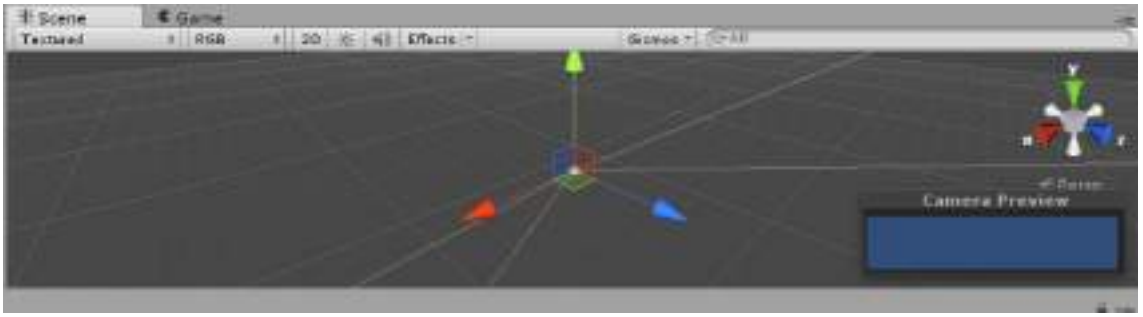
- Update( ) function: function that defines the main loop of the game so it is executed once for each frame that is rendered.
- Awake( ) function: The variables defined within this function shall be initialised just before the game starts. This function shall be executed only once for the duration of the script instance.
- Start( ) function: This function is very similar to "Awake( )" but has a vital difference; it will only be executed if the script instance is enabled. This will allow us to delay the initialisation of some game state variables. We must also keep in mind that the "Start()" function will always be executed after the "Awake()" function has finished.

Unity 3D also has a default script editor called uniscite.



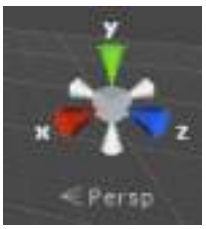
4. STARTING UNITY 3D

The first thing to do is to start Unity 3D. When you do this a blank project will load. If you want another scene then we look in the menu: "File -> Open Scene" and look in the project directory ("Assets/scenes"). Display modes by default the scene view has a 3D perspective of the scene.



Perspective

You can change the scene view to a number of views: top down, side and front. On the right side of the scene view you will see a "Gizmo" which looks like a box with cones coming out of it.



- With this Gizmo we can change the perspective of the view.
- Clicking on the box will take you to perspective mode.
- Clicking on the green cone (y) will take you to Top-Down mode.
- Clicking on the red cone (x) will take you to Side mode (right).
- Clicking on the blue icon (z) will take you to Front mode.
- We also have 3 grey cones that will take us to the following modes:
- "Back, Left and Bottom".

4.1 HIERARCHY VIEW

The hierarchy view contains all the elements that make up the game scene. You can manipulate all the objects in the scene from this view, every time you enter an element in the scene, an entry for this element is added in this view. Selecting an object in this view also selects it in the scene and in the inspector. This allows and facilitates moving, scaling, rotating and deleting the object or editing its parameters. All this without touching a single line of code.

To insert a new object to the scene you can click on "CREATE" and all the objects you want to add will be displayed.

4.2 SCENE VIEW

The scene view is the part of the 3D graphical environment for creating each scene. To work in the scene view the easiest way is to drag an object from the project view to the scene view, which would place the object in the scene, and thus place it where you want it to be.

The scene view is also the place where you edit terrains (sculpting them, painting textures and placing elements), place lights and cameras and other objects. The following will explain the scene view options:

On the left side of the scene view you will find a set of buttons to change the general display settings. Let's look at each of them, from left to right:

Render mode

The default will be "Textured". If clicked, a drop-down list will appear with a number of different rendering options:

- Textured: Textures are rendered in the view.
- Wireframe: Surfaces are not rendered, we only see the mesh.
- Textured Wire: Textures are rendered, but we also see the mesh.
- Render Paths: change the representations and characteristics of objects according to the projects being created.
- lightmap Resolution: to give a more optimal resolution to the objects that make up the scene.

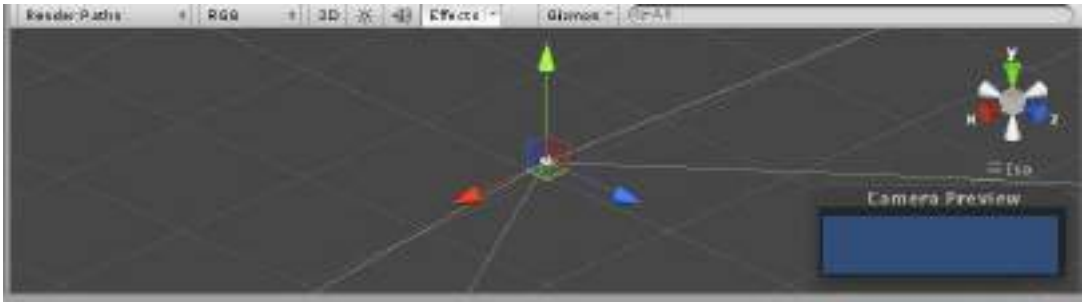
Colour mode, displayed as "RGB".

If you click on this button, a drop-down list will appear showing the available colour modes and you will be able to see the different ways of displaying the scene:

- RGB: All colours are rendered.
- Alpha: The mode is changed to "Alpha".
- Overdraw: The mode is changed to "Overdraw".
- Mipmaps: The mode is changed to "Mipmaps".

3D-2D option

Where we change the scene display from an X, Y, Z plane to an X, Y viewing plane.



Light switch

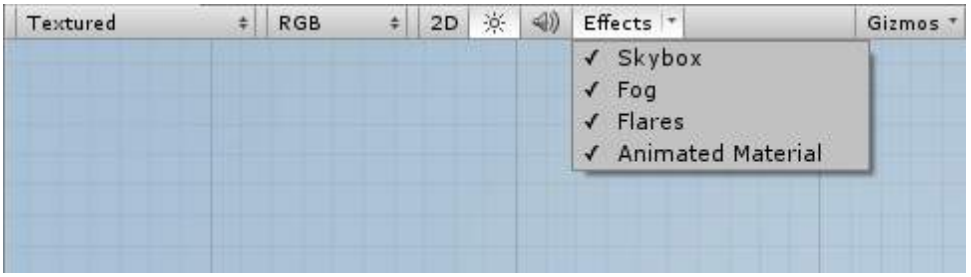
The next button turns the stage lighting on or off.

Audio

The next button turns the scene audio on or off.

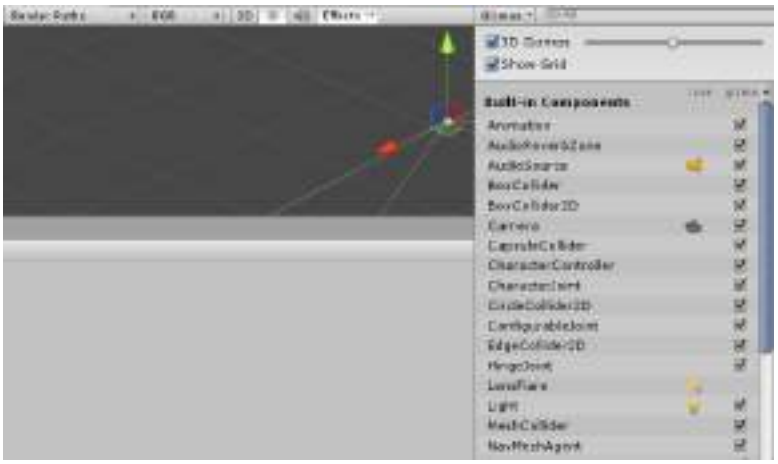
Effects

The effects button is used to disable video effects to improve performance while designing the project.



Gizmos

They help to refine the graphics of the scene even when using programming languages.



4.3 GAME VIEW

The game view allows you to preview the game in the as-is view or by maximising it to full screen. In this view you can see the game in motion to check that the game is working properly.

The Game view is very useful but it has a drawback, when testing the project on the computer, it moves with the keys or the mouse which does not help much if the project is being developed for another platform, for example a mobile device. In this case the tests have to be done using an emulator or compiling a mobile version.

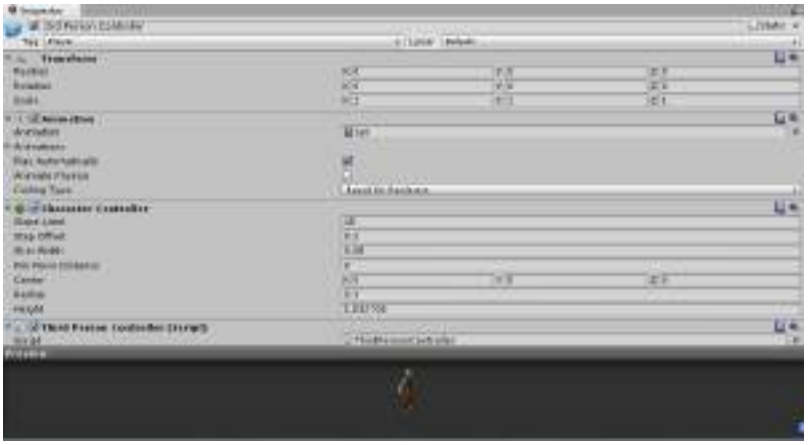
- The "Maximize on Play" button: allows you to maximize the full screen view of the game.
- The "Stats" button:
- The "Gizmos" button:
- Free Aspect" drop-down:

4.4 INSPECTOR'S VIEW

In the inspector view, the parameters of the objects selected in the scene view or in the hierarchy view are displayed. This way when you select an object in the scene, you can modify its parameters in the inspector (position, rotation, translation, etc). The inspector also serves as a tool panel for some elements such as terrains allowing you to sculpt them, add textures and more.

In the inspector view you can see the elements that define the behaviour of the objects, these elements are called components. In addition, objects and their associated components can be activated or deactivated here. The components can be scripts, animations, colliders...etc.

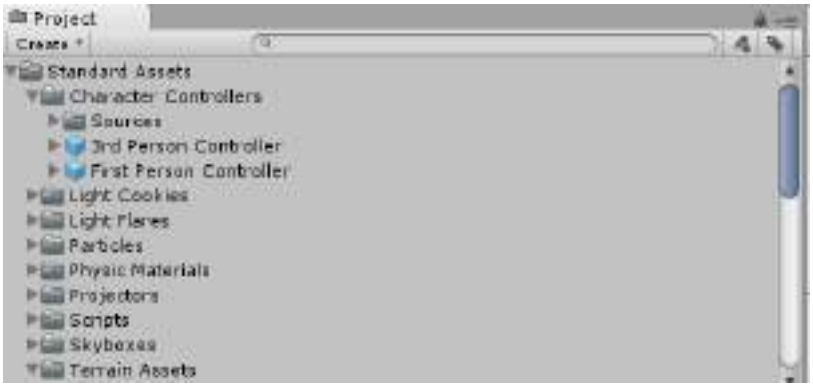
Finally, the inspector changes depending on the selected element.



4.5 PROJECT VIEW

The project view is a library of assets for the creation of the game. In this view are stored all the elements that are created and imported to be used in the game. These elements can be 3D objects, textures, sounds or scripts.

In this window it is possible to have a multitude of resources but this does not imply the use of all these resources for the creation of the video game.



Since this view contains both the elements used in the scene and the elements not used in the scene, it may contain a large amount of data. This implies that a good structure has to be maintained to facilitate the user's work. The organisation of this view can be done by creating a hierarchy of folders. The "create" dropdown contains the options that allow a good organisation of the view.

5. CREATE A NEW PROJECT

Creating a new project involves going to the top bar of Unity and clicking on File -> New Project to open the "Create new Project" dialogue window.

First select the folder in which you want to save the project by clicking on "Browse...". In order to create a new project, you have to import the necessary packages for the project.

Packages are sets of assets and can be imported when creating the project or at any time afterwards.

The imported packages can be the ones predefined in Unity, others created by the user or obtained from the Web, some of them are free to download and others are paid.

From the list that appears on the website, select the assets and click on "download".

For the assets found in the programme, we select the ones we want and click on "Create". The basic package to import is the standard set of assets.

When the new project is created, Unity 3D is restarted by creating the project structure in the specified folder. The Unity 3D start screen opens, showing the imported assets in the project view and a camera in the scene and hierarchy view.

Some of the most important packages are:

- Character Controller: Allows you to create scripts to move the character.
- Particles: Allows the creation of particles such as fire, smoke, waterfalls and explosions.
- Skyboxes: Enables the creation of skies.
- Terrain: Allows terrain to be created and sculpted.
- Water: Create water (sea, rivers, etc).
- Tree: Creator to create trees.

5.1 CREATE A NEW SCENE

Unity 3D works by scenes and when we create a new project it automatically creates a new scene, if we want to create another scene, go to File _ New Scene and we save it automatically in File _ Save Scene. The scene is saved in the asset folder of the project and will appear in the project view (folders can be found in Documents/New Unity Project or whatever name we call the scene).

Once the scene is saved we can create all the elements that will compose the scene, and we have different ways to manipulate the scene.

Some tricks for scene manipulation:

- If we want to see the scene from different angles we can rotate around using ALT and moving the mouse.
- If you want to zoom in or out you can use the mouse wheel.
- The arrows on the keyboard allow you to move the scene left, right, up and down. The same can be done by holding down the mouse and moving it in one direction or the other.
- To increase the speed of movement of the scene hold SHIFT.

When creating a scene, a camera appears by default in our world. If we click on "Main Camera", the characteristics of the camera can be displayed in the inspector and can be modified (positions, rotation, scale).

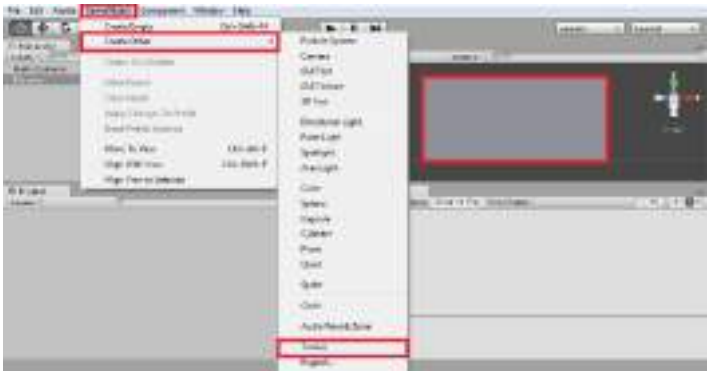
The camera defines the player's point of view and several cameras can be combined in the same scene.

5.2 CREATING THE GROUND

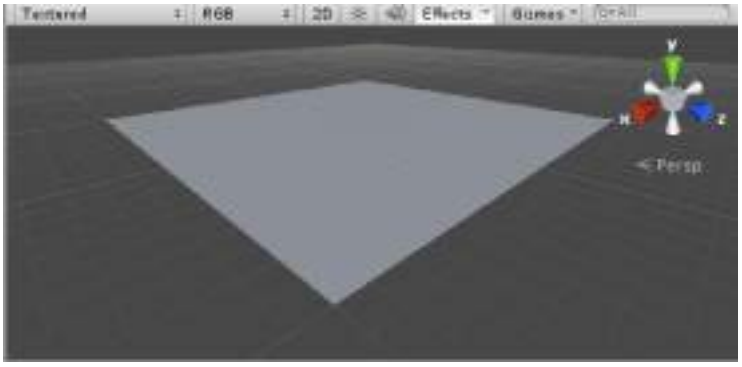
Unity 3D has a tool with which you can define any terrain, it normally handles terrains as a flat mesh, but you can texture and sculpt without leaving the editor. From the menu you can create a terrain by clicking on:

- “Terrain -> Create Terrain”.

You get a flat terrain, if the terrain is not visible you have to deactivate the lights in the scene view.



Once the terrain has been created, the following result is obtained:

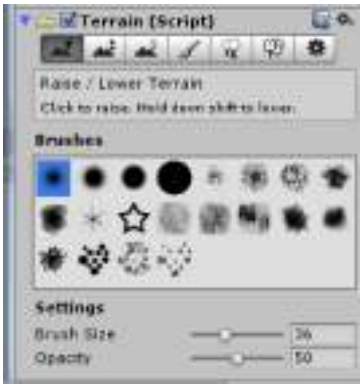


If we want to customise the terrain and make it more realistic we have to make use of the terrain tool. The terrain is selected in the scene view or in the hierarchy view so that the terrain editing tool appears in the inspector view.

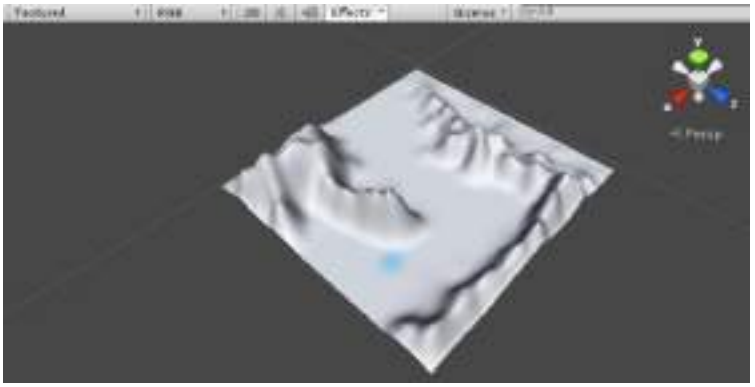
This tool is divided into 3 parts:

- Transform: Allows to move, rotate and scale the terrain on the x,y,z axes.
- Terrain Script: Contains several tools and properties for terrain that will be explained later.
- Terrain Collider: Contains the collision properties for the terrain.
- We are going to proceed to detail the "Terrain Script" panel and we will see a row of buttons. These terrain editor buttons allow you to perform different tasks. The description of what the buttons allow you to do from left to right:
- Raise / Lower: allows you to raise and lower the terrain geometry using a brush.
- Set Height: we paint the terrain with a height limit.

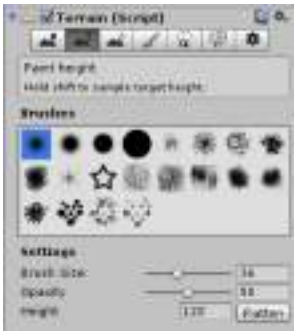
- Smooth: allows you to smooth a terrain to eliminate corners.
- Paint Texture: allows you to paint textures on the terrain surface.
- Place Trees: allows you to place trees.
- Paint Detail: allows you to draw terrain details such as grass.
- Terrain Settings: Access to the terrain properties where we can change several properties.



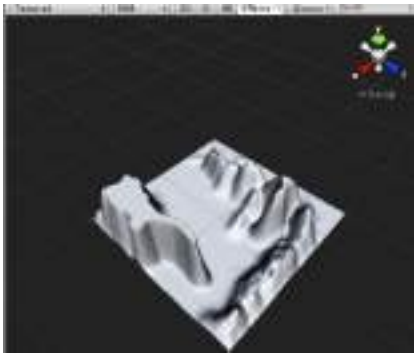
First you define the heights of the terrain by clicking on the places where you want to place a mountain, the longer you hold down the mouse the bigger the mountain will become. The terrain can be transformed into the following image:



The second option "Set Height" allows you to set a maximum height in the "Height" field.



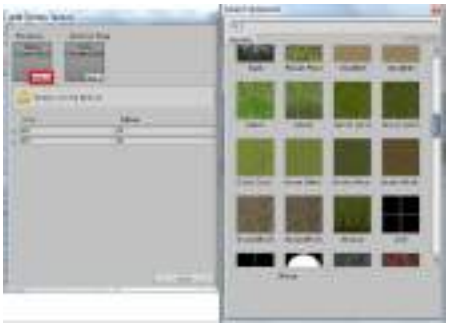
By setting a maximum height, the terrain can be further shaped to give it the right geometry. Once the maximum height is reached, the terrain is flattened. The third option "Smooth" allows you to smooth the peaks of the terrain. It is advisable to sculpt in steps, first the large parts and then refine the smaller details. At the end you get a terrain as shown in the picture:



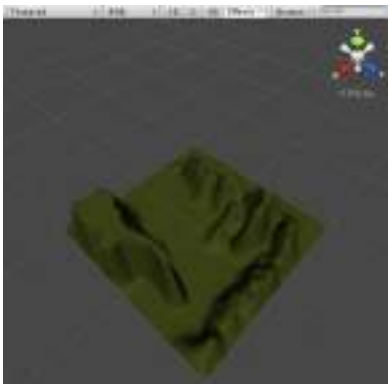
Once a terrain with the desired appearance is obtained, several textures can be applied. The first texture is considered the base texture and is applied to the entire terrain. The other textures will be applied in higher layers.

The fourth option "Paint the terrain texture" allows you to add texture to the terrain. First choose the type of brush you want to use in "Brushes", then add the chosen texture in "Textures" by clicking on "Edit texture> Add Texture".

An "Add Terrain Texture" window appears, click on the red highlighted circle to get the "Select Texture 2D" window, you can choose a texture and it will appear in the "Splat" field and in the "Texture" field in the inspector. You can also choose the width and length of the texture.



Once the textures have been applied, they will appear on the terrain, which will look as shown in the following image:



Unity 3D offers special support for introducing trees. You can add as many trees as you want using the fifth option "Places tree". First you use the "Edit Trees" option to

choose the tree you want to add to your world, then you customise the tree by defining several parameters, the size of the brush with which the tree is to be placed, the density of the trees (determines the number of trees to be placed), the colour variation (colour difference between one tree and another), the height and width of the tree.

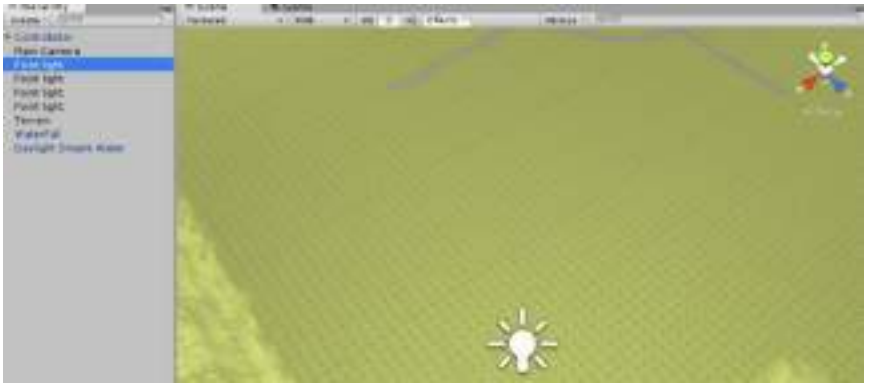
Finally, click anywhere on the terrain to add the tree at the chosen location. To delete the trees, hold down the Shift key and click on the terrain.

You can also add several trees to your world in bulk to create a forest in one go. Click on "Terrain->MassPlace Trees" and a dialogue box will appear in which you can determine the number of trees you want to introduce into the game.

A tree can be selected by going to the folder "Standard assets > Tree Creator > Textures", click and drag it to the terrain.

5.3 LIGHTING

If you want to illuminate the stage with the same intensity just add point light and put the desired lights on the stage: créate>point light



If high quality lighting is desired, Unity integrates the Beast technology, which is an Autodesk software that generates high quality lighting mappings.

"Create Lightmap" is used for different light creations that allow varied lighting for different terrain textures.

Access Windows>Lightmapping and see the following:

To control the resolution of the lightmaps, the resolution value is set in the Bake panel and applied automatically.

Lightmapping improves the definition on the ground as an area of trees and their shadows.

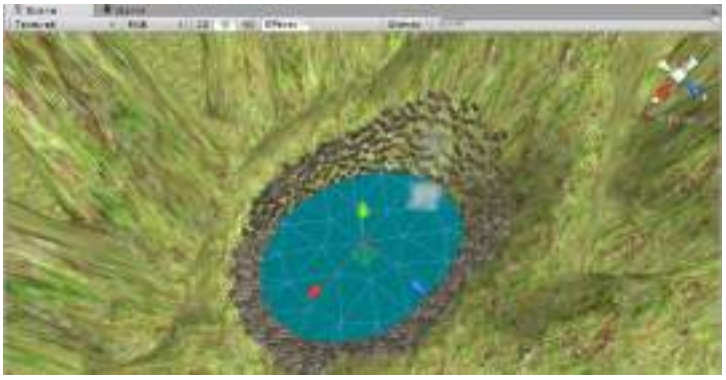
5.4 LAND ENVIRONMENT

The terrain environment will help the user to customise the environment in which the scene he is building is located, for example by adding a sky or water.

5.4.1. ADDING WATER TO THE SCENE

Unity includes several ways to add water to the scene, one of which is that the user can use directly. Two of these directly are the "Bake de lightmapping" and the "Nighttime Simple Water".

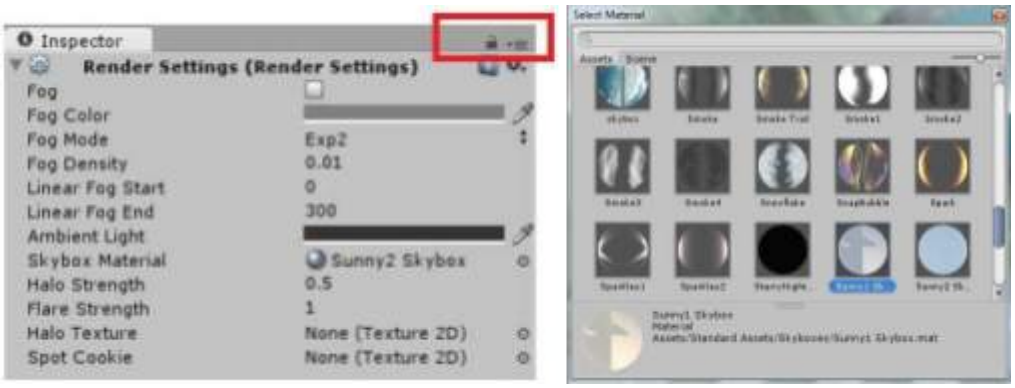
As an example we will add "Daylight Simple Water" inside the terrain. Go to "Standard asset > Water", select the "Daylight Simple Water" and drag it into the scene.



The characteristics of the water can be seen in the inspector:

5.4.2 ADD SKY

You can add the sky in two ways, by attaching a skybox to the camera or directly to the scene. Go to "Edit > Render Settings" to activate the fields in the inspector as follows:



This menu allows you to configure the scene's sky, fog effects and some global lighting effects. You can import your own textures or use the default ones from unity assets.

There are 2 options for applying the sky to the scene:

Option 1: go to the Render Settings menu, click on the small disc icon to the right of the Skybox Material field, a new window will open with all the material resources loaded in the project.

Option 2: in the project view under "Standard Asset > Skyboxes", select the appropriate sky for your scene and drag it to the "Sybox Material". For this result the sky "Sunny 1 Skybox" has been chosen.

Unity 3D offers several options to customise the environment of the scene, these are some of them:

- Fog: allows you to add fog to the scene, the scene becomes misty.
- Fog Color: the colour of the fog, default colour is grey.

- Fog Mode: linear, exponential or exponential squared. This controls how the fog fades.
- Fog Density: density of the fog, only used for exponential and exponential squared. a. A higher density will decrease visibility and a lower density will increase visibility.
- Linear Fog Start/End: start and end of fog distances, only used if the linear fog option is activated.
- Ambient Light: default is a grey. If we want to give the scene a different lighting scheme, changing the Ambient Light is a good start.
- Skybox Material: the sky texture of the scene.
- Halo Strength: Size of all light halos relative to their range.
- Flare Strength: Intensity of all flares in the scene.
- Halo Texture: reference to a texture that will appear as the glow of all halos in the lights.
- Spot Cookie: The reference to a Texture2D that will appear as the cookie mask for all spotlights.

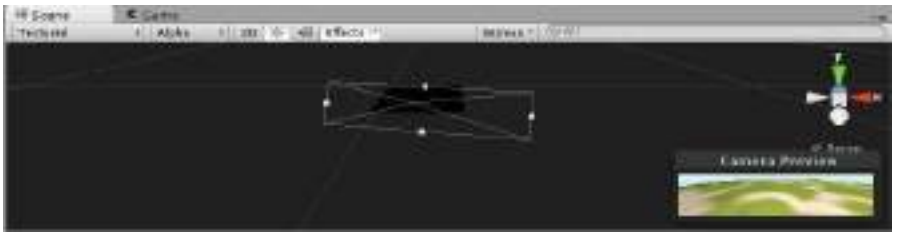
5.4.3 MOVEMENT

Unity includes a first person controller that makes it easy to create games with a first person view. By accessing the "Standard Assets > Character Controllers", select the "First Person Controller" and drag it to the position where you want to place it in the scene.

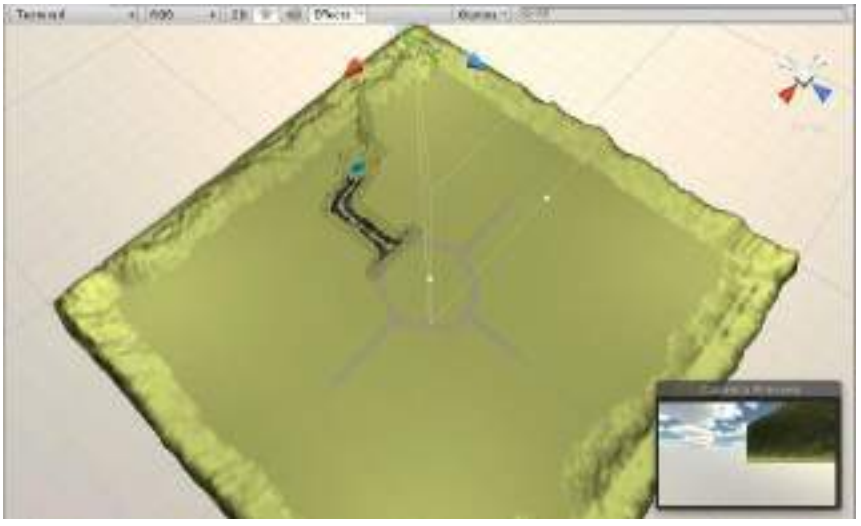
In previous versions of Unity you could only program the movement with a small script shown below:

```
var velocidad=8;
public var ptosalida:GameObject;

function Update () {

        var direccion=transform.TransformDirection( Vector3( 0, 0, 0.1 ));

        if( Input.GetKey("up")){
                //transform.Translate( 0 , 0 , 0.1 );
                rigidbody.AddForce( direccion*velocidad );
        }

        if( Input.GetKey("left"))
                transform.Rotate( 0 , -1 , 0 );

        if( Input.GetKey("down")){
                //transform.Translate( 0 , 0 , -0.1 );
                rigidbody.AddForce( direccion*velocidad*-1 );
        }

        if( Input.GetKey("right"))
                transform.Rotate( 0 , 1 , 0 );
}
```

You can see the camera that has been added to your scene, from the four camera points you can increase or decrease the focus that the camera will have.

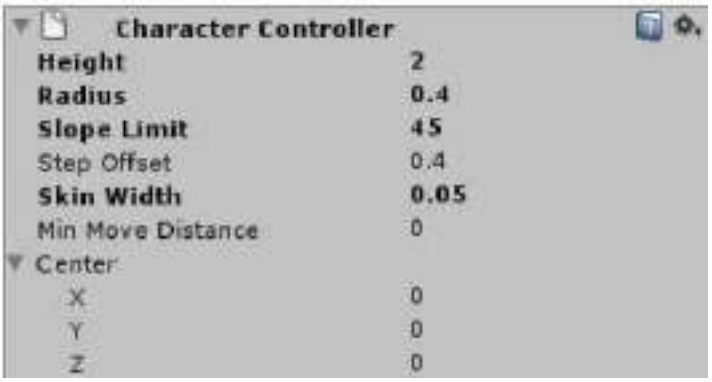


Once we have our camera ready, the movement can be done through several options: Movement: with arrows or keys (W: forward, A: left, D: right and S: back). The camera is controlled with the mouse.



The jump with the space key.

The "First Person Controller" can be customised in the same way as the other elements in the inspector. To do this, it is selected in the hierarchy view and its characteristics appear in the inspector.



Explanation of controllers:

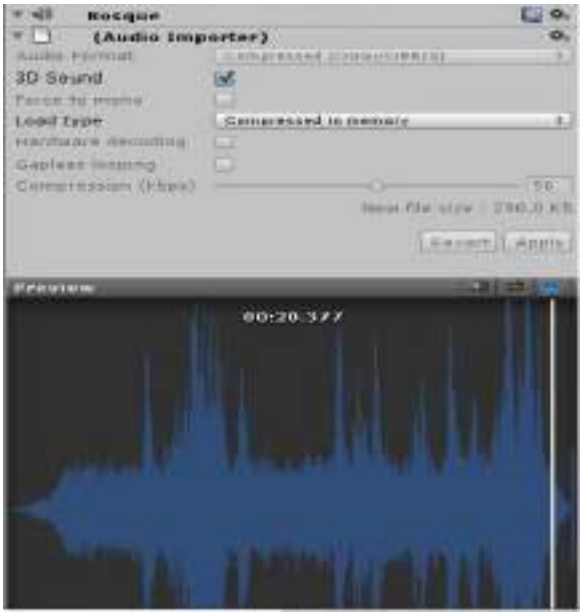
- Height: height of the controller
- Radius: controller radio
- Slope limit: maximum value of the slope that the controller can climb.
- Step offset: this value is between 0.1 and 0.4 for a human being of 2 metres in size.
- Skin width: allows you to control the character's colliders so that the character is not blocked.
- Min move distance: If the character tries to move below the indicated value, it does not move at all. This can be used to reduce vibrations. In most situations this value should be left at 0.
- Center: the position of the controller, its x, y, z coordinates.

5.4.4 AUDIO ON THE SCENE

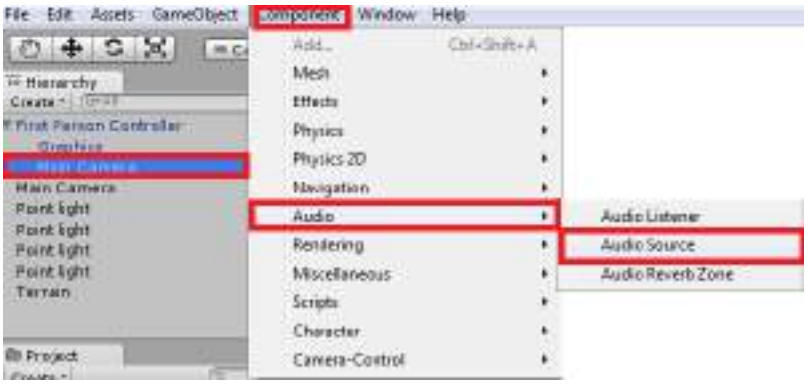
In we can insert sound into the game. There are two different types of audio, 3D audio or 2D audio.

3D audio is heard at one position and attenuates as you move away from that position. 2D audio is heard evenly throughout the scene.

First you create a folder named "sound" in which you save an imported sound, for example one that represents the environmental sounds of a jungle or forest.



Select the "Main Camera" and go to Component> Audio> Audio Source.



The imported sound is then dragged into the "Audio Clip" box in the inspector view and the audio is configured:

The "Play On Awake" checkbox is activated so that the sound starts at the beginning of the project.

The "Loop" checkbox is activated so that the music is looped and repeated as many times as necessary.

On the "Listner" graph in the component, select the green point at the highest point and drag it to the centre of coordinates (0,0).

A minimum distance of 0.1 is defined in "Min Distance".

5.4.5 TURN STAGE LIGHT ON AND OFF

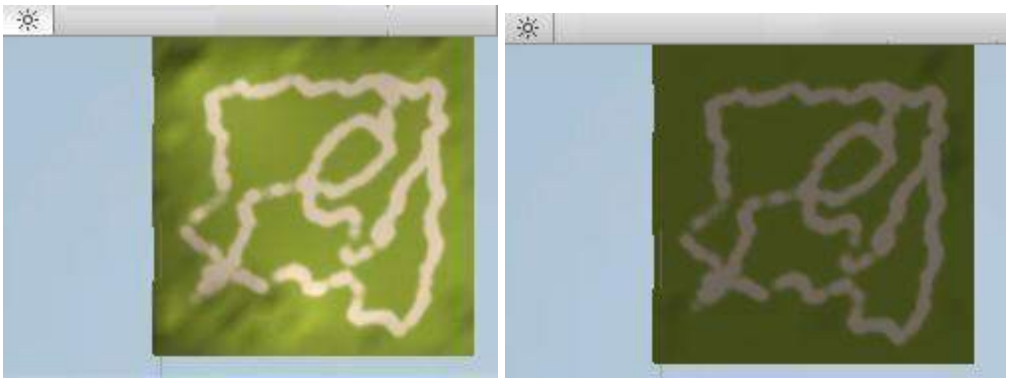
In this version of Unity turning the stage light on and off works by clicking the light button found in the stage view options, but if you want to turn the light off in the stage you create at some point, you can write a simple script as shown below:

```
var myLight : Light;
var myLight1 : Light;

function Start(){
    myLight = Light.FindObjectOfType(typeof(Light));
    }


function Update () {

    if(Input.GetButtonDown('Fire1')){
        myLight.enabled = !myLight.enabled;
        myLight1.enabled = !myLight.enabled;
    }
}
```

Explanation of the script:

- myLight = Light.FindObjectOfType(typeof(Light)); _ searches the scene for the object of type "light" (the light in the scene).
- if(Input.GetButtonDown("Fire1")) when the left mouse button is pressed.
- myLight.enabled = !myLight.enabled; _ changes the state of the light, if it is on, it is turned off and vice versa.
- myLight1.enabled = !myLight.enabled _ this line is duplicated because there are 2 lights in our scene.

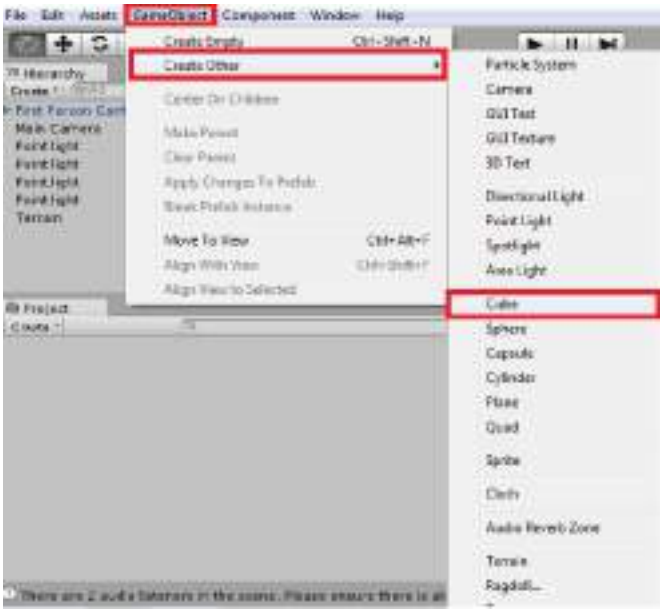
Once the script is created, it is added to the object and in the parameters that appear in the inspector view, drag each of the 2 lights so that it looks like this:



5.4.6. CREATION OF AN OBJECT

We will explain how to create an element, in this case a cube.

To create the predefined objects in Unity 3D, go to GameObjet > Create Other and a list of items is displayed.



Some important components of GameObject are:

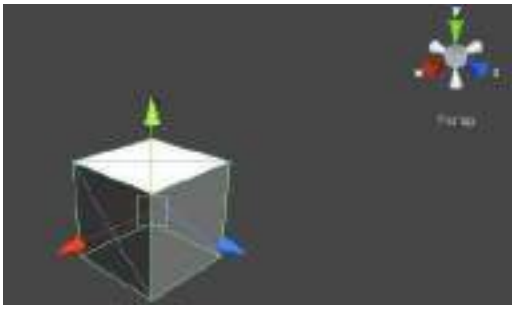
- Camera: allows you to insert a camera into the scene, although when creating a project Unity places a camera by default.
- Directional light: introduces a uniform directional light that allows you to see the scene.
- Cube: put a cube in the scene.
- Sphere: put a sphere in
- Capsule: insert a capsule
- Cylinder: to put a cylinder
- Plane: introduces a plane into the scene. When one of the "GameObjects" is selected, it appears in the scene view, game view and hierarchy view. In addition, in the inspector view you can see the attributes of the object for the parameterisation of the object.

5.4.7 MANIPULATION OF THE OBJECT

Once we create the object, in this case a cube, we can modify it to have the necessary dimensions, position and shape to achieve the desired characteristics.

The position of the cube can be changed using the second control button:

As we can see in the following image, we can move the cube on the X (red), Y (green) and Z (blue) axes. To move the cube just click on the axis where you want to move it and drag it to the new position.



The third control button allows you to rotate the cube.

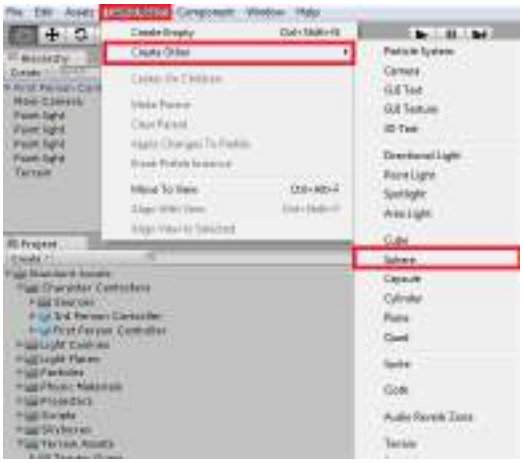
Comparing the previous image, in this one we see how the cube rotates.

The different circles that appear around the cube when the rotation option is selected allow the user to choose the axis on which he/she wants to perform this operation. To

rotate an element, click on the one identified by coloured circles and move the mouse in the selected direction.

5.4.7 CREATION OF A WORM

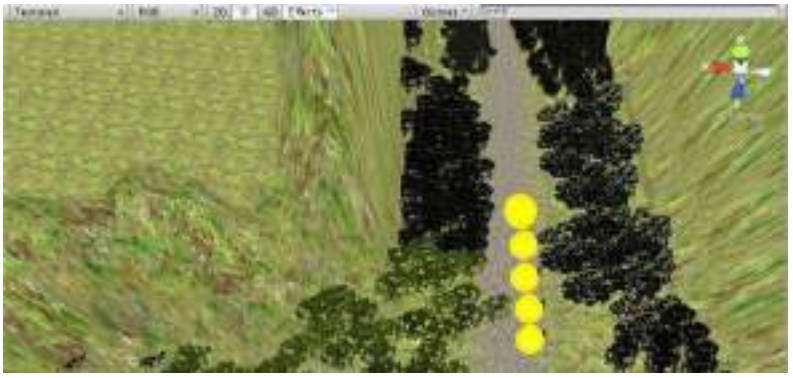
As we already know how to create an object, we are going to create a simple character, a worm based on spheres. The first sphere is created in "GameObject > Create Other > Sphere":



Place the sphere on the terrain in the position you want. To create other identical spheres, press "CTRL + D", this new sphere appears on top of the old one and can be moved. Repeat the same action until you have 4 spheres. You can see that the spheres appear in the hierarchy view where you can rename the first sphere to "head" and the others to "body". To rename a Game Object, select it in the hierarchy view, right click and "Rename". Modify the spheres representing the body by making them smaller using the inspector for more precision.

In addition, to optimise the organisation of our project, an empty object is created in "GameObject > Create Empty", renamed as a worm and the head and body are dragged onto the worm.

Then a texture is imported, applied to the character and the following result is obtained:

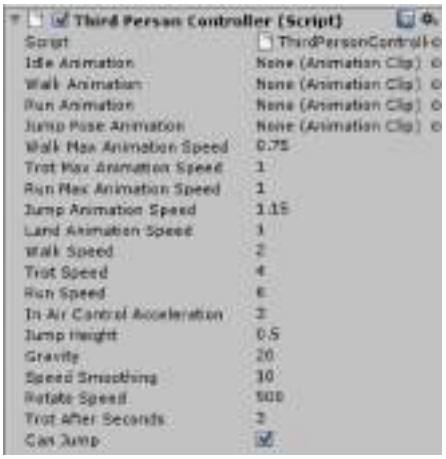


Finally, to make all the elements that make up the body of the worm follow the head, the "Smooth follow" component will be used, which will be explained in the next section.

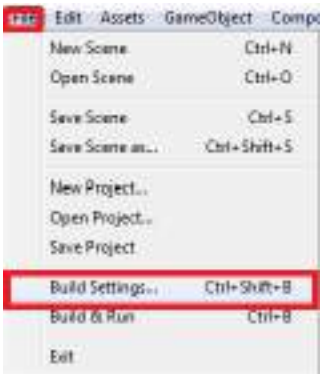
5.4.8 MOBILITY OF THE OBJECT

Select the head in the hierarchy view, go to Component> Scripts> Third Person Controller. In this way you can give mobility to the head and control it in third person. If you want to move the whole snake, you would have to apply the "Third Person Controller" to the whole worm.

And we can see in the inspector view how the new component appears with all the parameters that can be configured.



To finalise the project we can export run the project:



You select the platform and architecture on which you want to export the project, you can export on platforms WINDOWS, MAC, LINUX, ANDROID, IOS can with free

licensing if you want to export to game consoles such as XBOX, WII and PLAY STATION would have to pay for licence.

Finally, click on Build and Run and the project starts to export, give it a name and store it in the folder of your choice, ending with an executable of the platform you have chosen, taking into account the architecture of the computer on which the project is being carried out.

At the end of the project, it can be concluded that video game engines are very useful tools that facilitate the work of creators by allowing them to focus on the creative aspects and not so much on the structural aspects, in order to satisfy the demands of consumers rather than to adapt to demanding consumers.

Unity 3D is one of the most current engines for the design of video games, it is very complete and a very powerful engine. In addition, each version has been adapted to new trends such as multi platform design. This characteristic gives Unity3D an advantage over some video game engines.

To carry out this project we have used the free version of Unity 3D, but because it is free it is no less powerful. Unity offers users a simple environment that allows the creation of a project very quickly and since it is not necessary to have deep knowledge of programming anyone can use it with a few basic notions.

We must bear in mind that Unity not only allows the creation of video games or virtual tours, but it is also a very popular tool for architects and designers, since it allows the creation of 3D environments in a simple and fast way.

Unit 3 - How to create codes using the C# programming language

INTRODUCTION.

In 2002, Microsoft announced the release of the C# language, similar to C++ and Java, but improved (in music the symbol # stands for "one semitone higher"). This development was an important part of Microsoft's "dot net" initiative, which we will describe below. Because of the similarity between C# and its predecessors, learning it will make the C, C++ and Java languages more understandable if you ever need to use them.

- C# is an object-oriented language, derived from C++ and Java.
- Microsoft's .NET framework is an important product, which includes the languages C#, C++ and Visual Basic.
- Programs are lists of instructions that computers obey automatically.
- Nowadays, the main trend in programming practice is the Object Oriented Methodology (OOP), and C# fully supports it.

1. CREATING A PROJECT WITH C#

In this manual we will use the acronym "IDE" (Integrated Development Environment) to refer to the C# integrated development environment. Like word processors, programs that facilitate the creation of documents, the IDE has tools for creating (developing) programs. All the tools we need for that purpose are integrated, so we can do all our work within the IDE instead of having to use other programs. In other words, the IDE provides us with a complete programming environment.

1.2 INSTALLATION AND CONFIGURATION

To follow the exercises you need to download Microsoft Visual C# 2008 Express Edition. Follow the installation instructions and register the product if you are prompted to do so.

We will now explain how to group the programs you create with C# in a specific folder. When you create a program, C# generates several files and places them in a new folder. The resulting set of files is called a "project". It is a good idea to create a top-level folder to store all your projects. The default folder that C# uses is called Projects and is located inside the My Documents folder in Visual Studio 2008 on the C drive. Almost all users find this to be appropriate, but if for some reason you need to change that default location, follow these steps:

1. Click on the Tools menu at the top of the screen. Select Options... Make sure that Show all settings is selected.
2. Click on the Projects and Solutions option and then on the ... button to the right of the Project Location section.
3. The Project Location window will then open, where you can choose an existing folder or create a new one. Click the OK button when you are finished.

From now on C# will save all the work you do in the default project folder or in the one you have specifically selected. In any case, the first time you save a project, a dialog box with a Browse button will appear, in case you need to use a different folder than the one you have selected.

1.3 CREATING THE FIRST PROGRAMME

The program we will create next will display the *Hello World* message on your computer screen. Regardless of the above, you will need to follow these general steps to create any programme.

Figure 2.1 The home page

Figure 2.2 The New Project window

- Open the IDE. The Home Page will appear, as shown in Figure 2.1.
- **Click on the Create: Project** link... Figure 2.2 shows the **New Project** window that will appear in response.
- Make sure that the **Windows Forms Application** template is selected. Choose a name for your project, which will also become a folder ID. We recommend that you use only letters, digits and spaces. In our case we used the name **First Hello**. Click **OK**. A layout area similar (but not necessarily identical) to the one shown in Figure 2.4 will appear.
- To facilitate your foray into the C# language, the toolbox should always be visible. Click on the **View menu** and select **Toolbox**. Now click on the "pushpin" symbol in the toolbox, as shown in Figure 2.3; the toolbox will be permanently fixed and open. Click on **All Windows Forms** to see the list of available tools. Your screen will now look almost identical to the one in Figure 2.4.

As previously mentioned, the above steps are common in the realisation of any project. Let us now perform a specific task for this particular project.
Note the form and toolbox illustrated in Figure 2.4. We will now select a control from the toolbox, and place it on the form. Here are the steps to follow:

- Locate the toolbox and click on the **Label** control.
- Move the mouse pointer to the form. Click and, without releasing the button, drag to create a label as in Figure 2.5.

- Now we will set some properties of the label: right-click on the label and select **Properties**. Move the mouse to the Properties list at the bottom right of the screen, as shown in Figure 2.6.

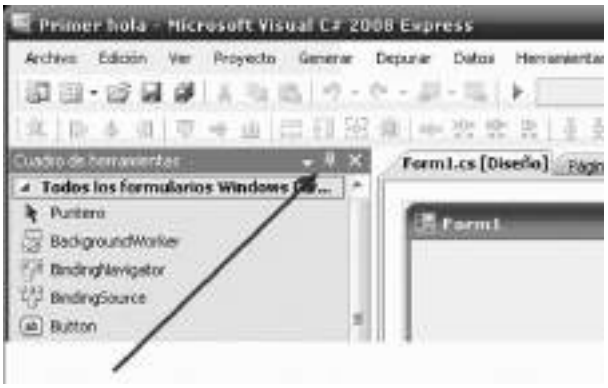


Figure 2.3 The toolbox is fixed and open.

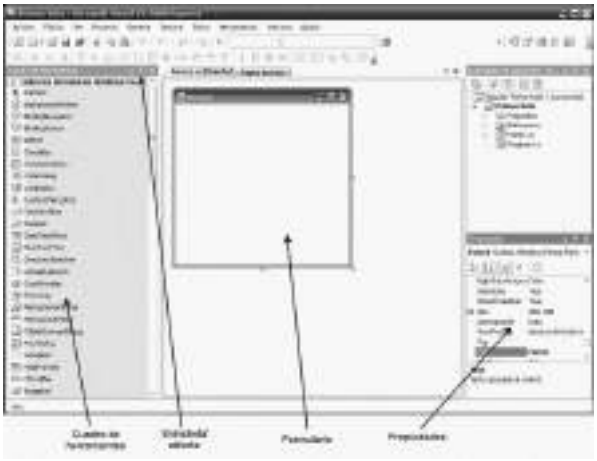


Figure 2.4 The IDE at design time.

Figure 2.5 A label was added to the form.

Figure 2.6 Label properties.

Figure 2.7 Click on the arrow to run the programme.

- Scroll down to the **Text** property and replace the text **label1** with **Hello World**.
- Now let's run the program by clicking on the arrow at the top of the IDE (Figure 2.7); this is the Start Debugging button.

A new window will appear, as shown in Figure 2.8. This is the programme you have created. It only shows some text, but it is a real window in that you can move it, resize it and close it by clicking the **X** icon in the top right corner. Experiment with this window, and then close it.

To save your programme for later use:

- Go to the **File** menu and select the option **Save All** (hereafter, to indicate this type of actions we will use a short form like this: **File | Save All**).
- **The Save Project** window will then appear, as shown in Figure 2.9. Make sure that the option **Create directory for solution** is unchecked. Leave the other options as they are and click **Save**. When you save the project again, the same options will automatically be used and the **Save Project** window will no longer appear.

You can now use the **File | Exit** command to close the IDE.

The next time you use C#, the name of your project will appear on the Start Page, so you can open it with a single click, without having to repeat the work we did to set up the project.

Figure 2.8 The **First Hello** program running.

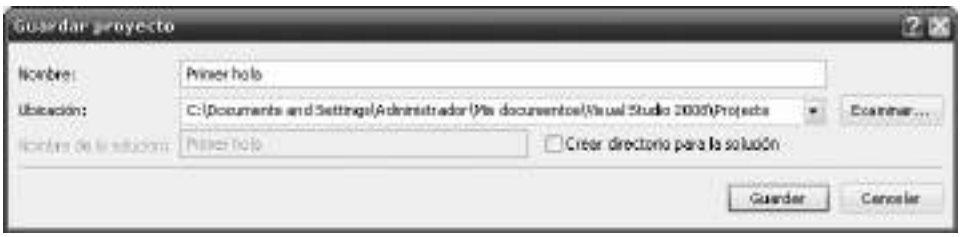


Figure 2.9 The **Save project** window, with the **Create directory** option deactivated.

1.4 DESIGN-TIME CONTROLS

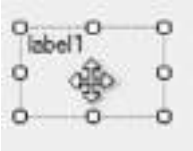
In our First Hello programme we placed a label on a form and modified the text to be displayed. The main purpose of the exercise was to review the steps involved in creating a project. Next, we will learn some of the basics of controls and properties.

What is a control? It is a "device" that appears on the screen, either to display information, to allow the user to interact with the application, or both. The C# IDE itself uses many controls. For example, in the previous exercise you used the drop-down menus to save projects, and the OK button to confirm actions. For Windows applications, the toolbox contains about 70 controls; part of the programming task involves selecting the appropriate controls, placing them on a form, and setting their properties. In contrast to "run-time", this phase is related to the use of the controls and is called "design time". When the program runs, the user interacts with the controls.

The programmer's job is to create a graphical user interface (GUI) to facilitate this interaction. Let us now look at how the controls can be manipulated at design time.

- It is possible to select a control from the toolbox and place it on a form. The initial position assigned to it is not important, as we can easily change it.
- It is also possible to move the control. If you place the mouse pointer over a control, a four-headed arrow will appear next to it, as shown in Figure 2.10. You can then drag the control with the mouse.
- You can change the size of the control. When you click on a control, several small boxes (size handles) appear at its edges. Place the mouse over one of these small boxes and a double-headed arrow will appear, as shown in Figure 2.11. Drag the edge or corner to resize the rectangle.

In fact, the method of resizing depends on the control itself. For example, the label control has a property called AutoSize (automatic sizing, according to the dimensions of the text entered) which we set to



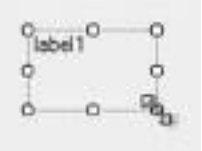


Figure 2.10 Controls can be moved ... **Figure 2.11** ... and resized as appropriate.

If we leave the AutoSize property set to True, the width of the label will be determined based on the width of the text to be displayed, and the height will be determined based on the font size of the text. Other controls allow the width to be varied by dragging the handles with the mouse, but not the height (which is set according to the fonts). Some controls - such as buttons - allow their size to be changed in any direction. In general, the size of controls depends on their intended use, so we recommend experimenting with them. However, since labels are so common, we should not forget their AutoSize property.

Next, we will look at properties. Here is an analogy: televisions have properties such as the colour of the casing, the size of the screen, the channel it is showing at any given time, its price and its brand.

Similarly, each control has a set of properties that can be adjusted at design time to suit our requirements. We will see later how to change a property at runtime.

After placing a control on the form, to view its properties, right-click on it and select Properties; this will display the properties window of the chosen control. The left column contains the names of the properties, and the right column contains their current value. To change a property we must modify the value in the right column. For some properties it may be necessary to select several additional options, such as when changing colour settings and text fonts. Sometimes, when the range of values to choose from is very wide, it is necessary to open an additional window.

Another vital aspect of a control is its name. This is not itself a property, but for convenience it is shown in the list of properties as (Name). The brackets indicate that it is not actually a property.

When we place multiple labels on a form, the IDE selects their names as follows:

label1label2label3 ...

These names are acceptable for now, but in the following chapters we will see that it is better to change the default names of some controls to more representative names. To change the name of a control you need to modify the text to the right of (Name) in the list of properties.

1.5 EVENTS AND *BUTTON* CONTROL

The program we created in the previous exercise is not very representative, since it always shows the same words and the user cannot interact with it. Next we will enrich this program so that a text message appears when the user clicks on a button. This is an example of how to use an event.

Almost all events are triggered by the user, and are generated when the user manipulates a control in some way at runtime. Each control has several events to which it can respond, such as the click of a mouse button, a double-click, or the placement of the mouse pointer over the control. Other types of events do not

originate from the user; for example, notification that a Web page has finished downloading.

In the programme we will create next, we will detect an event (the click of a button); then we will make a text message be displayed on a label. Here is how we will create the user interface:

- Create a new project called Hello button.
- Place a label and a button on the form. The position in which you do this is not important.
- Type Click here in the Text property of the button.
- Modify the Text property of the label, so that it appears without content.

Run the program, although it is not yet complete. Note that you can click on the button, and that it changes its appearance slightly to confirm that it is being pressed; nothing else happens. Close the form.

Now let's see how to detect the click event. Double-click on the button inside the layout form. This will open a new information panel, as shown in Figure 2.12. Note the tabs at the top:

Form1. csHome page Form1.cs [Design] [Design

You can click on these tabs to switch panels. The Form1.cs pane displays a C# program. This is known as "program text", or C# "code". Let's modify this code using the IDE editor.

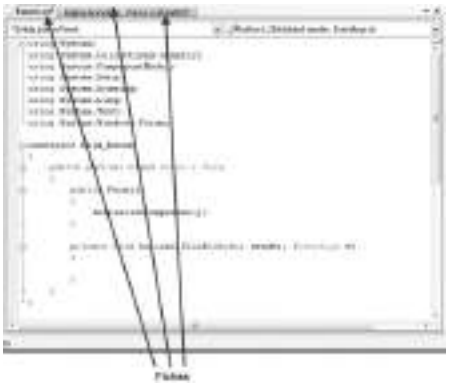


Figure 2.12 C# code in the editing pane.

Scroll through the code until you find the next section:

private void button1_Click(object sender, EventArgs e)

{

}

This section of code is known as a *method*. The name of the method is **button1_Click**. When the user clicks on the **button1 button** at runtime, it will carry out, or "execute", whatever instruction we place between the two lines above.

In this specific program we will use an instruction to place the message **Hello World** in the text of the **label1 label**. The instruction to perform this action is:

label1.Text = "Hello world";

Write the instruction exactly as shown here, between the lines **{** and **}**. In the following chapters we will explain the exact meaning of lines like the one above (which imply an "assignment instruction").

The next step is to run the programme. There are two possibilities for this:

- If the programme runs correctly, when you click on the button you will see the message **Hello World** appear on the label.

Figure 2.13 Wrong action. Tick the check box and click **No**.

- The other possibility is that the program will not run due to an error. In this case C# will display a message like the one in Figure 2.13. Check the checkbox for the option **Do not show this dialog box again**, and click **No** to continue. The message will not appear again; henceforth the only evidence of the error will be an underline in the text of the code.

In any case, the error must be corrected: review the code, correct any typos, and run the program again. We will discuss errors in more detail later.

Here are the new features of this programme:

- It can respond to the click of a button. The process of placing code so that the corresponding action is performed when an event occurs is known as "handling" the event.
- Modify the value of a control's property at runtime. Previously we only did this at design time. This is a very important part of programming, as we often have to show results by placing them on a certain property of a control.

1.6 OPENING OF AN EXISTING PROJECT

To reopen a previously created project, save and close the project in progress and go to the Start Page. This shows the projects you have worked on most recently; to open a project, simply click on its name. If the project you are looking for is not in the list, click on the Open: Project... link and navigate to the appropriate folder. Inside the folder is a file of type . sln (solution). Select this file and open the project. To view the form go to the Solution Explorer, located at the top right of the window, and double-click on the form name.

1.7 DOCUMENTATION OF PROPERTY VALUES

In this book we need to provide you with the values of the properties that we will use in the controls. When the number of properties is small we can explain them in a few statements, but if the number is larger we will use a table. In general we have several controls; each control has several properties and each property has a value. Note that there are a large number of properties for each control, but we will only list those that you have to modify. The others will keep their default value. Here is an example:

Control	Property	Value
label1	Text	Hello world
label1	BackColor	Web

button1 Text Click here

The values listed correspond to the following controls:

-label1 whose text is Hello world, and whose background colour is red;

-button1 whose text is Click here.

Be careful when writing property names. We will discuss this in more detail in later chapters; for now, suffice it to say that you should do so with an initial capital letter, and no spaces.

1.8 ERRORS IN THE PROGRAMMES

It is common for bugs to occur in programs; fortunately, the IDE can detect some of them for us. Here is an example of an error:

label1.Tixt = "Hello world".

If you type this line and run the program, the IDE will underline it. Placing the mouse pointer over the underlined part will bring up an explanation of the error. Note that the explanation may be difficult to understand (even for an expert), or that the error may lie elsewhere in the program. However, the first step should always be to inspect the underlined area for typing errors. As you learn more about the C# language, your ability to detect errors will improve.

Once the highlighted errors have been corrected, the program has to be executed. In fact, before this happens, another program called a compiler is activated, which checks the code for perhaps more errors. Only until all compilation errors have been corrected can your program be executed.

Compilation errors are displayed in the Error List window, which opens below the code. By double-clicking on the error in the results window, the IDE will take you to the point in the code where the error needs to be corrected.

Anyone starting to write programs will face many compilation errors. Don't be discouraged; this is part of the learning process.

After correcting the compilation errors, the program will be able to run. At this point we may notice that it does so incorrectly, which would mean that there is a "run-time error", or bug. Such errors are more difficult to correct, and debugging is therefore necessary.

1.9 FUNCTIONS OF THE EDITOR

The editor is not limited to displaying what the programmer writes; the following are some of its additional virtues:

- It offers standard cut, copy and paste operations to the clipboard, allowing you to copy code from other Windows programs.
- To write larger programs, additional controls and events are required. The drop-down lists at the top of the editor allow you to select controls, plus a specific event for that control. The editor is automatically placed in the appropriate method.
- The editor will display the C# code as it is typed according to certain rules. For example, spaces will automatically be inserted around the = symbol.
- Some lines will need indenting - shifting the text to the right - for which additional white space must be inserted. This is usually done in four-space intervals. As we will see in later chapters, C# code consists of sections within sections, and indentations help indicate where each section begins and ends.

The process of indenting a program is also known as formatting. The IDE can do this automatically via the menu:

Editing | Advanced | Formatting the document It is a good idea to format the code frequently.

- Each type of control has a particular set of properties. If you type the name of a control followed by a dot, as shown below: ***label1.***

and wait a little while, the editor will provide you with a list of the properties of the label1 tag. If this tool is not configured on your system, you can activate it by selecting the option:

Tools | Options | Text Editor | C#

and then ticking the checkbox Automatic membership list.

- Sections of code that we are not working on can be collapsed into a single line of code. To do this, click on the small "-" symbol to the left of the editor. To expand a section, click on the "+" symbol. You will find these symbols next to lines like the following:

private void button1_Click (object sender, EventArgs e)

You do not need to remember all these tools, as they are always at your fingertips to help you.

One of the tasks related to code layout that is not performed automatically is the splitting of long lines. To ensure that a whole line is visible on the screen, we can choose a suitable place to split it and press the Enter key. The IDE will indent the second part of the line by four spaces. Here is an example:

private void button1_Click (object sender, EventArgs e)

If we press the Enter key after the comma, the code will appear like this:

private void button1_Click (object sender, EventArgs e)

Often the IDE will create very long lines which, of course, should not contain errors, so it is best not to split them. However, as you become more experienced with C#, some of these long lines will be of your own creation, and it will be very convenient to split them so that you can see all the text at once. This also improves the readability of the printed code (known as listing).

1.10 THE MESSAGE BOX

Previously we used the label control to display text on the screen, but we can also use a message box to do this. This control does not appear in the toolbox, because it does not take up permanent space on a form; it only appears when it is required. The following is a code snippet that displays the Hello World message inside a message box when a button is clicked:

```
private void button1_Click(object sender, EventArgs e)
{
MessageBox.Show("Hello world");
}
```

Figure 2.14 shows what happens when you run the program and click the button: the message box is displayed, and you must click the OK button to make it disappear. This feature implies that message boxes are used to display important messages that the user cannot ignore.

To use a message box, type a line like the one above, but use your own message inside the double quotes. At this point we will not explain the purpose of the Show statement or why the brackets are required. We'll talk about all this when we study the methods in more detail.

Figure 2.14 A message box

1.11 HELP

The help system works on the basis of two possible sources: one local, located on your own computer, and the other via the Internet. The Internet source is updated frequently; however, the local version is sufficient when you start working with C#: In

case you do not have an Internet connection, the program will automatically use the local help.

If you wish to specify where the help should come from, you can reconfigure the system as follows:

1. In the main C# window, choose ***Help | Search....***
2. In the new help window, select ***Help | Help on Help***. You will then be presented with several options as to which source of help will have the highest priority.

The C# help system is extensive, but if you are new to programming you may find the information it contains complicated. It is most useful when you have some experience with C# and require precise technical detail.

The most useful options in the Help menu are Index and Search, as they allow you to type in some text for the system to locate pages that may be useful to you. The difference is that Index only looks at page titles, while Search also looks at page content.

1.11.1 Programming fundamentals

- Controls can be placed on a form at design time.
- It is possible to set the properties of controls at design time.
- Programmes are able to modify properties at runtime.
- When an event occurs (such as clicking a button), the C# system uses the appropriate method, but it is up to the programmer to place code within the method to handle that event.

1.11.2 Common programming errors

- Forget to terminate the execution of your program before trying to modify the form or the code.
- Confuse the design-time form with the run-time form.

1.11.3 Coding secrets

- In C# code, to refer to the property of a control we use its name followed by a dot and the name of the property, as in:

label1.Text

- To a code section between:

private void botonNumero_Click(Object sender, EventArgs e)
{
}
is known as the method.

- Message boxes are not placed directly on the forms. To make them appear on screen we must use the following instruction:

MessageBox.Show ("Here is the text you want");

1.11.4 New language elements

An introduction to properties, methods and events.

1.11.5 New IDE features

- Programmes are contained in projects.
- The IDE creates a folder containing the files needed for a project.
- It is possible to move and resize controls on forms.
- The toolbox contains a wide range of controls.
- Right-clicking on a control allows us to select its properties.
- Double-clicking a button at design time creates methods to handle events.

1.11.6 Summary

Part of the programming task involves placing controls on forms and setting their initial properties. The C# IDE performs this task directly, but you need to practice with and read about the IDE.

User interaction - Input

To allow the user to interact with our applications, Unity automatically provides an "Input" class.
For more information see:
https://docs.unity3d.com/Manual/class-InputManager.html
https://docs.unity3d.com/ScriptReference/Input.html

Input provides the possibility to verify whether and when a physical button (joypad, mouse, keyboard) is pressed, to obtain the position of one or more physical axes (es. analog joystick, shifting of the mouse) or to create virtual axes starting from two digital buttons (ex.speed - slows down button).
Unity is able to receive input from touch screens and from different smartphone sensors (ex. gyroscope, camera and accelerometer)

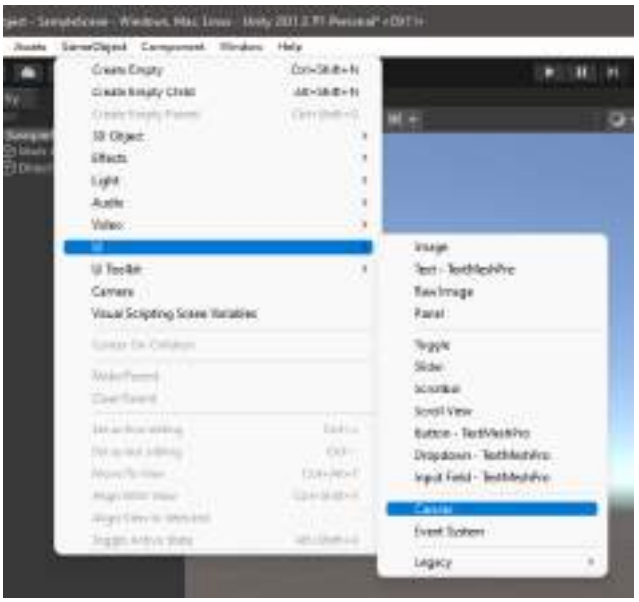
```
void Update(){
        if(Input.GetKeyUp(KeyCode.Space)){              //Input from keyboard or joypad
                Debug.Log("Space has been released");
        }
}

void Update(){
        float myVerticalValue = Input.GetAxis("Vertical");        //input from analog
joystick or virtual axes
         if(myVerticalValue  != 0)
         {
                //eg move the player
         }
}
```

```
void Update(){
        if(Input.GetMouseButton(0)){       //for mouse and touchscren input
                Debug.Log("Mouse 0 button has been clicked");
                Ray myRay = Camera.main.ScreenPointToRay(Input.mousePosition);
                 if(Physics.Raycast(myRay)){
                                //I clicked on something because myRay hit something
                 }
        }
}

void Update(){
        Vector3 myDeviceAcceleration = Input.acceleration;  //Input from accelerometer
                if(myDeviceAcceleration.magnitude > 10)
        {
         Debug.Log("Il device è stato scosso forte");
         }
}
```

It is good to check and manage the input in the Update function (because the function is invoked every time a frame is created) in order to have a responsiveness which is proportional to the framerate of the application.

User Interaction - Output

In case of log output (as text) on console it's possible to use the function:
`Debug.Log("customText");`
Most of the interaction between user and application takes place through the user interface or UI.
Unity allows you to easily create a user interface using the GameObject Canvas. For more information see:
https://docs.unity3d.com/2021.3/Documentation/Manual/UICanvas.html
https://docs.unity3d.com/ScriptReference/Canvas.html



Canvas allows you to create a programmable and customizable user interface using different elements.

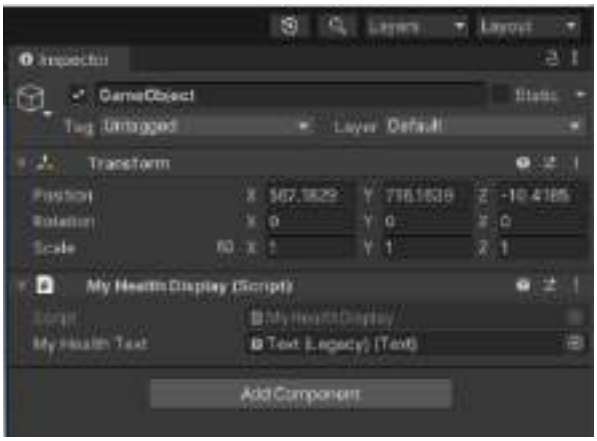
It's possible to add text and buttons.

To create a code that controls text through the button, add a script component to an Empty GameObject.

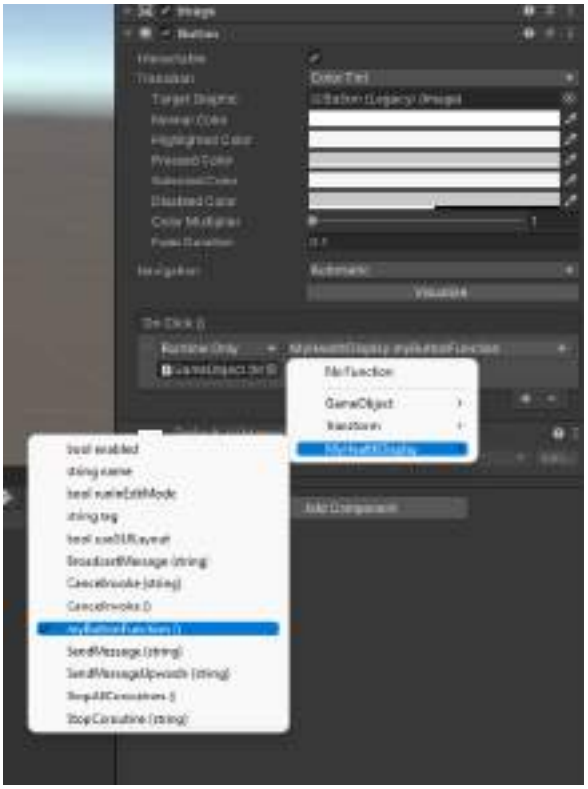
```
using UnityEngine.UI
public class HealthDisplay : MonoBehaviour {
        public Text myHealthText;
        int myHealth = 10;
```

```
        public void myButtonFunction(){                    //function to connect to
the button
                myHealth--;                                 //lower the healt
                myHealthText.text = "HEALT : " + myHealth;          //updates the label
        }
}
```

Link the text to the script using the attribute Public Text (My Health Text).



Link the button to the Empty GameObject script through the button's onclick() feature using the public function (void myButtonFunction()).



Physics - Collision

The collisions in Unity are processed thanks to the Collider components(BoxCollider, SphereCollider, MeshCollider) and managed using events (OnCollisionEnter, OnCollisionExit, OnCollisionStay). For more information see:
https://docs.unity3d.com/Manual/collision-section.html
https://docs.unity3d.com/ScriptReference/Collision.html

```
void OnCollisionEnter(Collision acollision){
        if(acollision.relativeVelocity.magnitude >= 10){     //if a ball collides with gameObject
                Destroy(gameObject);                    //delete gameObject
                Destroy(acollision.gameObject);         //Destroy the ball
        }
}
```

In Edit > Project Settings > Tags and Layers it's possible to add new Layers

In Edit > Project Settings > Physics > Collision Matrix it's possible to define among which layers collisions must occur.

The modification of the component transform or the use of features "transform.Translate or Transform.Rotate" don't produce collisions.

To avoid unnecessary computations enter the collision component only when necessary.

RigidBody

The main aspects of dynamics are simulated through the Rigidbody component.

https://docs.unity3d.com/Manual/rigidbody-physics-section.html

https://docs.unity3d.com/ScriptReference/Rigidbody.html

Through the use of the Rigidbody component you can apply forces and moments, you can set speed, friction forces and gravity and create movements that produce collisions.

```
private void Update(){
     int jumpForce = 100;
    int speed = 10;
    if(Input.GetKeyDown(KeyCode.Space))
        gameObject.rigidbody.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
    if(Input.GetKeyDown(KeyCode.R)
        gameObject.rigidbody.AddTorque(transform.right * speed, ForceMode.Force);
}
```

It is preferable to detach the use of physical simulations from the frame rate using the function.

FixedUpdate piuttosto che Update.

```
private bool goingForward;
private void Update(){
         goingForward = Input.GetKey(KeyCode.UpArrow);
}
```

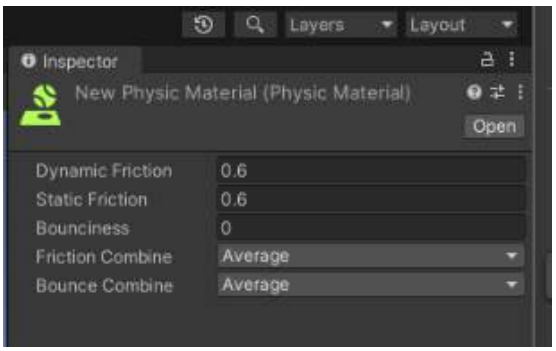
```
private void FixedUpdate(){
        if(goingForward){
                gameObject.rigidbody.AddForce(Vector3.Forward * speed,
ForceMode.Force);
        }
}
```

Very fast movement speeds may not be managed in time.

Physic Material

Physic Materials used on a GameObject with a collider enable to define some of its physical properties as the creation of static and dynamic friction and the elasticity of the object.

https://docs.unity3d.com/Manual/class-PhysicMaterial.html



Joints

It is possible to link two GameObject (with RigidBody) through the joint component.

https://docs.unity3d.com/Manual/joints-section.html

FixedJoint creates a fixed joint between the object this component has been attached to and another specified object. The result is a fixed joint where one of the objects physically follows the movements of the other (the same as a gameObject child follows a GameObject parent).

Hingejoint creates a hinge joint between the object this component has been attached to and another object. The result is a fixed constraint between the two objects which only have the ability to rotate on the axis of the hinge.

SpringJoint creates a spring joint between the object this component has been attached to and another object. The result is a joint between two objects that follows the physics of a spring.

Ray (obstacles and bullets)

The Ray class allows you to create rays to probe the presence of objects starting from a specific point and direction.
The RaycastHit class is used to memorise collisions though rays.
The function Physics.Raycast(.....) is used to check for collisions along the radius.
https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

```
Ray myRay = new Ray(myRayStartPoint,myRayDirection);
//Ray(transform.position, transform.forward) is a ray that starts from the object and is directed forward
RaycastHit myHit;
float rayDistance = 100.0;
if (Physics.Raycast(myRay, out myHit, rayDistance )){
    Debug.Log(myHit.collider.name); //we can get the collider of the hit object and its name or other components
}
```

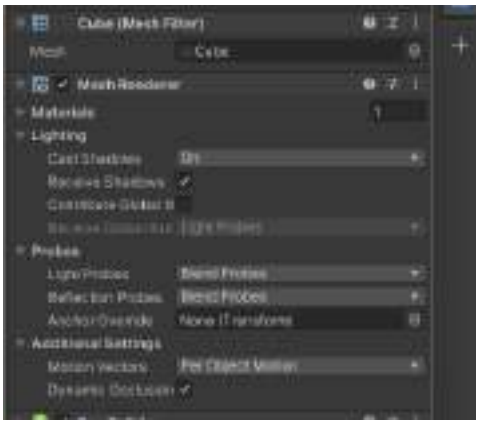
https://docs.unity3d.com/Manual/CameraRays.html

The Physics class provides several functionalities which are very useful for physics.
https://docs.unity3d.com/ScriptReference/Physics.html
https://docs.unity3d.com/Manual/PhysicsSection.html

Rendering basics

There are different components that characterise the graphics of the objects. The mesh filter component defines the shape of our GameObject.
https://docs.unity3d.com/Manual/mesh.html

With the Mesh Render component it is possible to define if the meshes of our GameObject have to project and receive shadows and how they must behave when they reflect light.
https://docs.unity3d.com/Manual/class-MeshRenderer.html



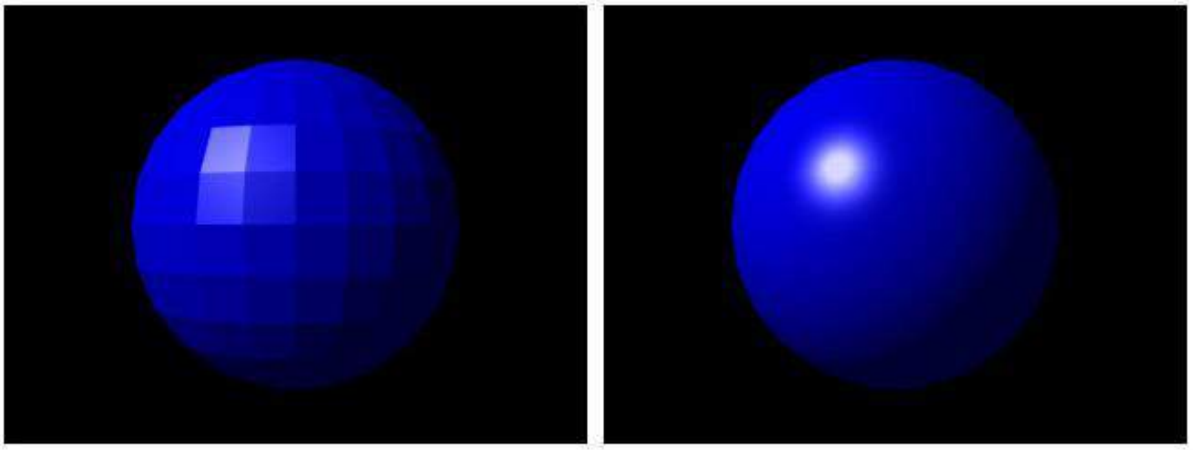
Shaders

Shaders are mathematical models used by Unity to tell the graphics card how to represent the objects on screen according to lights, shadows, effects…

Each shader performs different calculations and gets different visual effects (for example realistic shadows, cartoon effect, pixel effect…)
https://docs.unity3d.com/Manual/Shaders.html

Vertex shaders calculate reflections and shadows on a face of the mesh by averaging what happens on the vertices of the face (faster but lower quality. Reflections and shadows follow the shape of the mesh).

Pixel shaders calculate lights and shadows taking into account all the aspects related to an object set in a single screen pixel (slower but with higher quality, shadows and reflections are independent from the shape of the mesh they are attached).



Materials

Materials define the values that the shaders must use during the graphic calculations. Using the same shader it is possible to define materials with different colors, transparencies, brightness, opacity.

Not all shaders can handle all properties (eg transparency).

https://docs.unity3d.com/Manual/Materials.html

Texture

Textures can define color and other material properties pixel by pixel.

Importing a texture into Unity is as simple as dragging it into your Assets. To use it in a material, just drag it into the property of the material for which we want to use it. (e.g. albedo).

High-resolution textures complicate and lengthen graphic calculations exponentially.

https://docs.unity3d.com/Manual/Textures.html

Sources of Light

In Unity there are different light sources.

Point Light is a Light that's located at a point in the Scene and emits light in all directions equally

Spot Light is a Light that's located at a point in the Scene and emits light in a cone shape

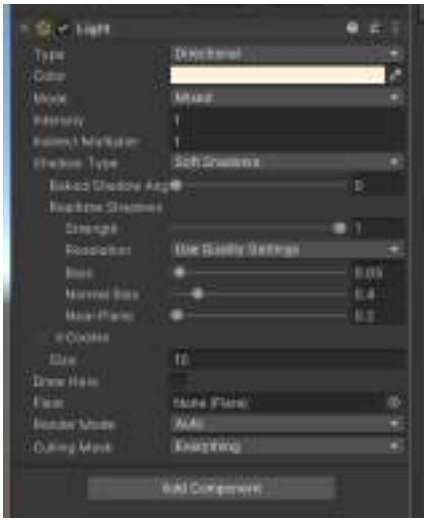
Directional Light is a Light that's located infinitely far away and emits light in one direction only

Area Light is a Light that's defined by polygon in the Scene, and emits light in all directions uniformly across its surface area but only from one side.

For each light source it is possible to define its intensity, its colour, in some cases the direction, the effect distance and whether or not this produces shadows.

https://docs.unity3d.com/Manual/Lighting.html

https://docs.unity3d.com/Manual/LightingOverview.html



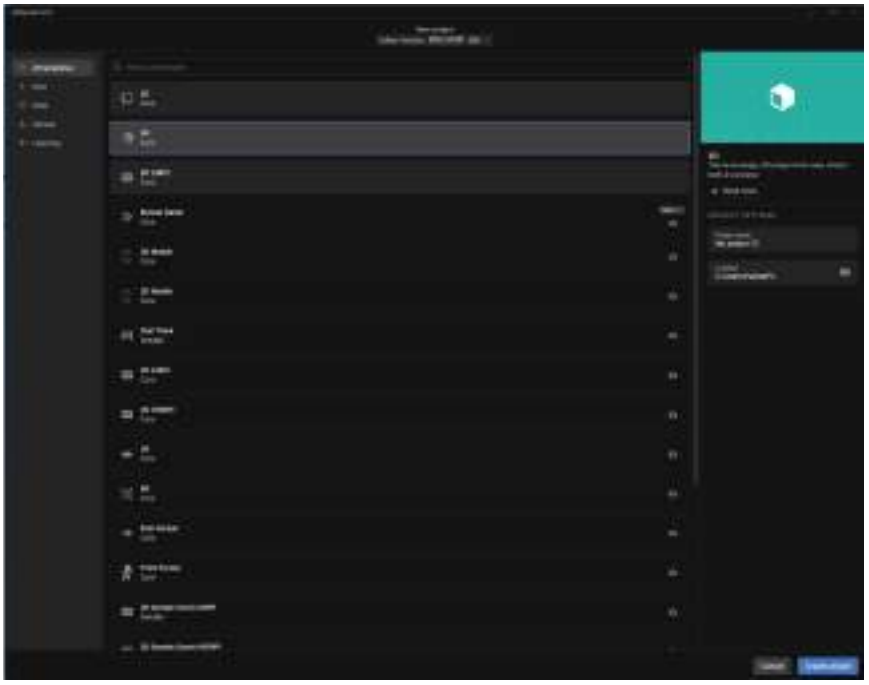
On mobile devices, the use of shadows is not recommended.

Unit 5 - Practical guide using codes of the PLAY video game

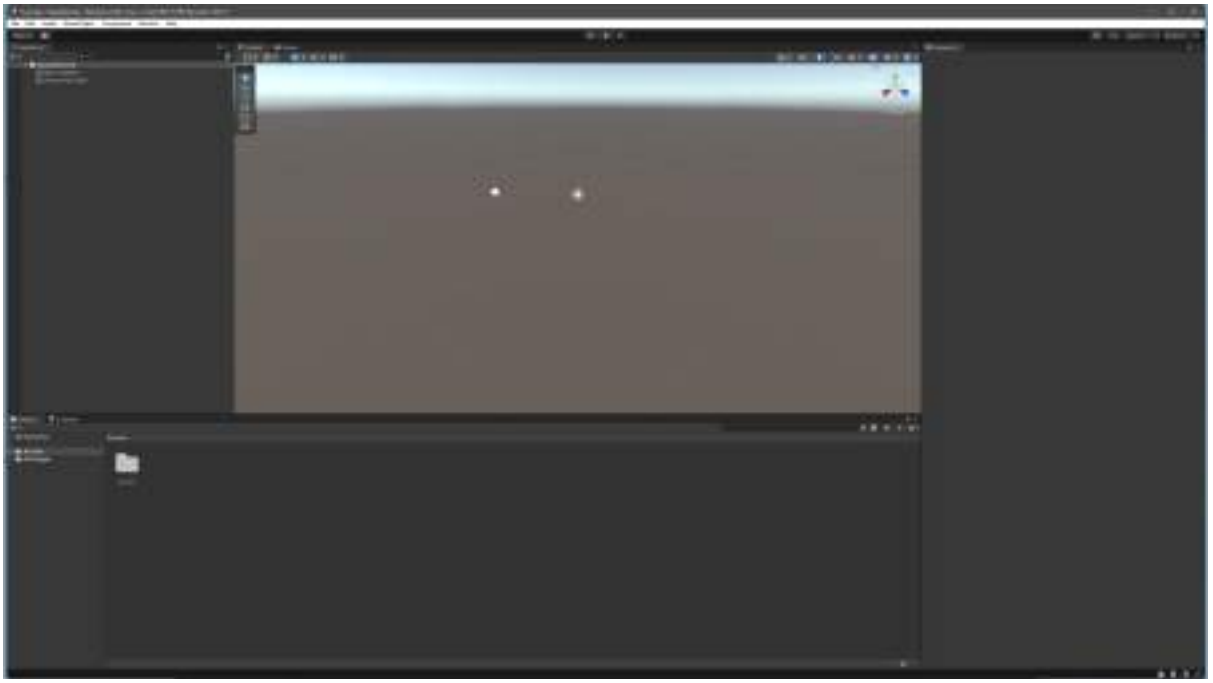
In this chapter we will see how to realise some of the features in the video game.

Let’s create a new project by pressing the *New project* button.

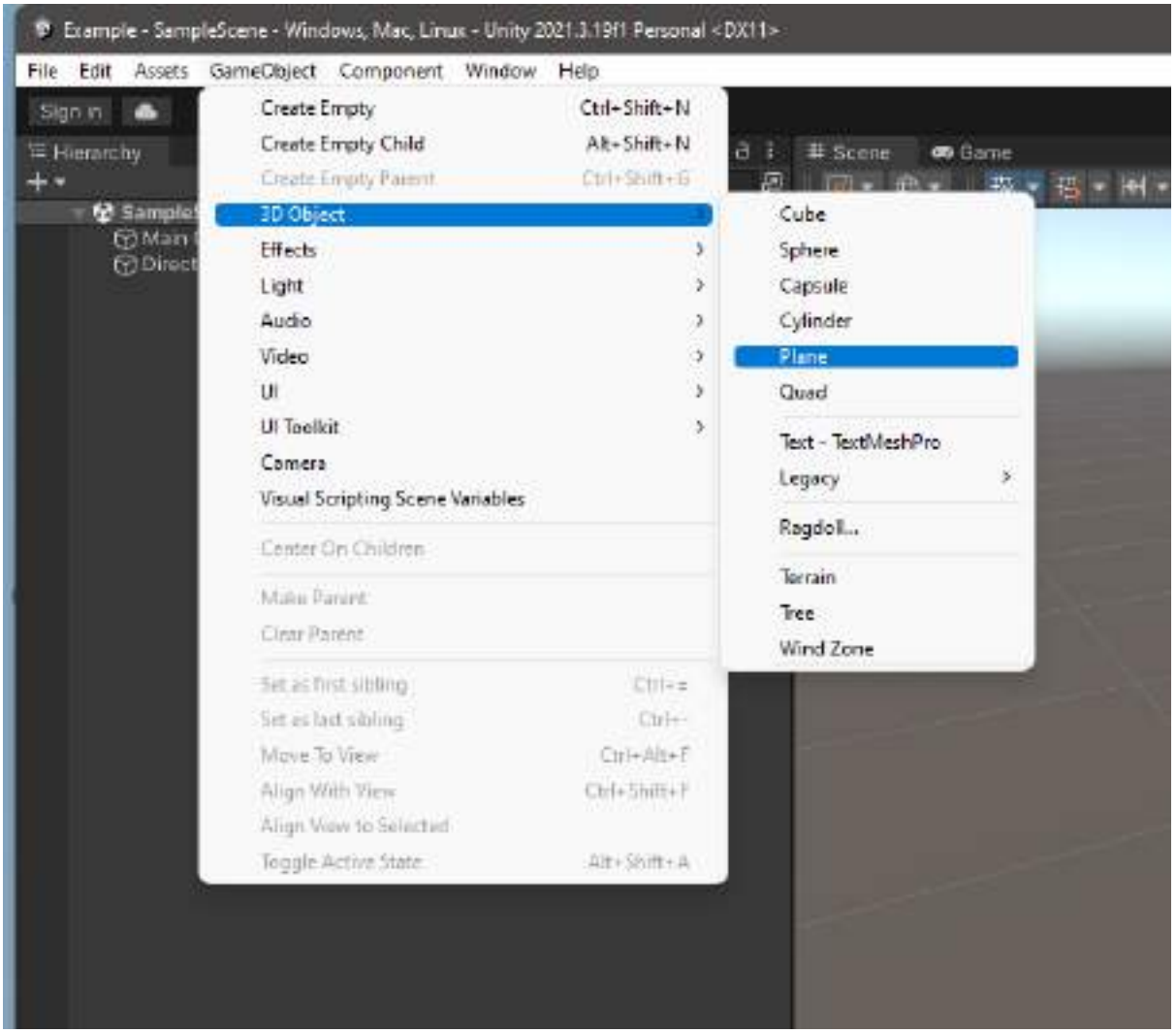
Now, select “3D-core”, set the project name and select “Create Project”.



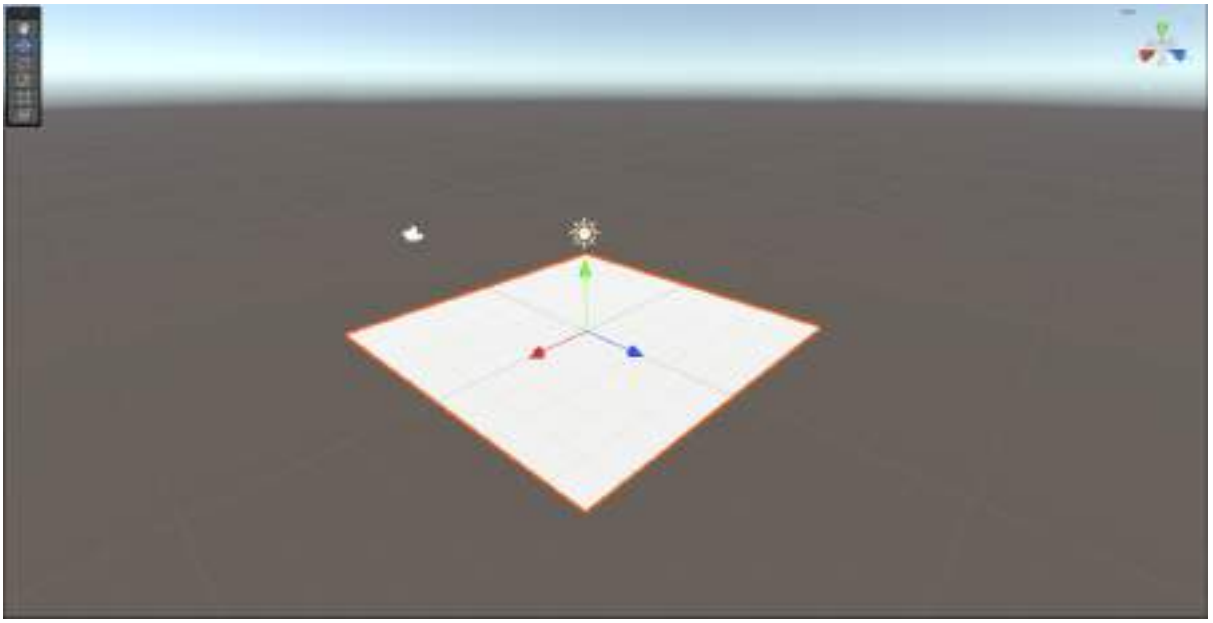
Once that the project is created, the Unity user interface will appear.



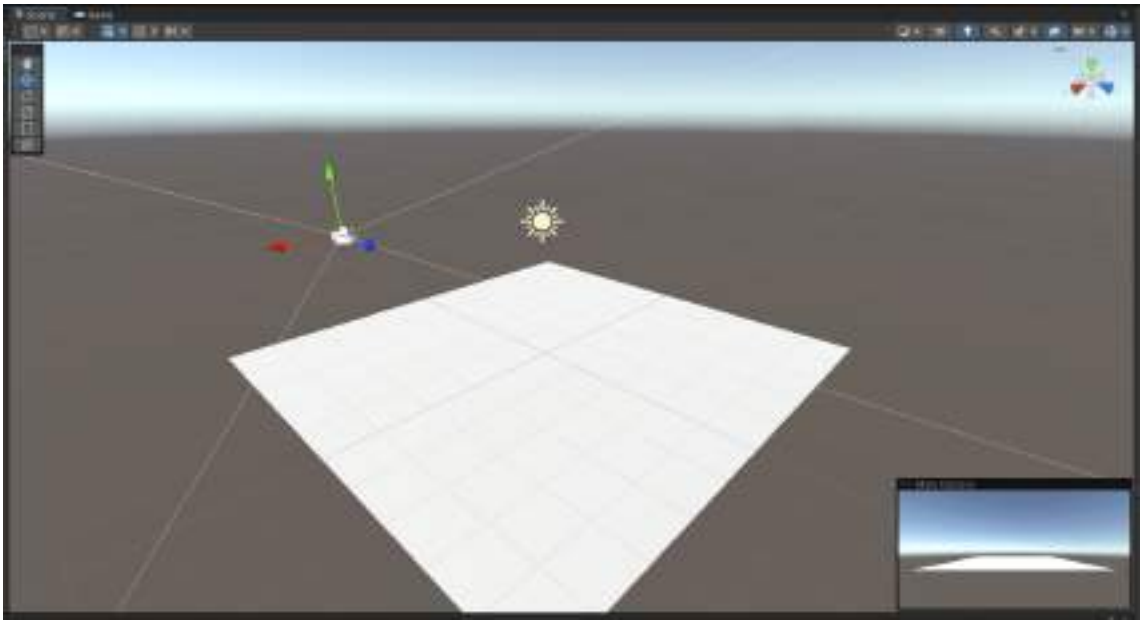
Let’s add a plane from the menù, following the written passages below: GameObject -> 3D Object -> Plane

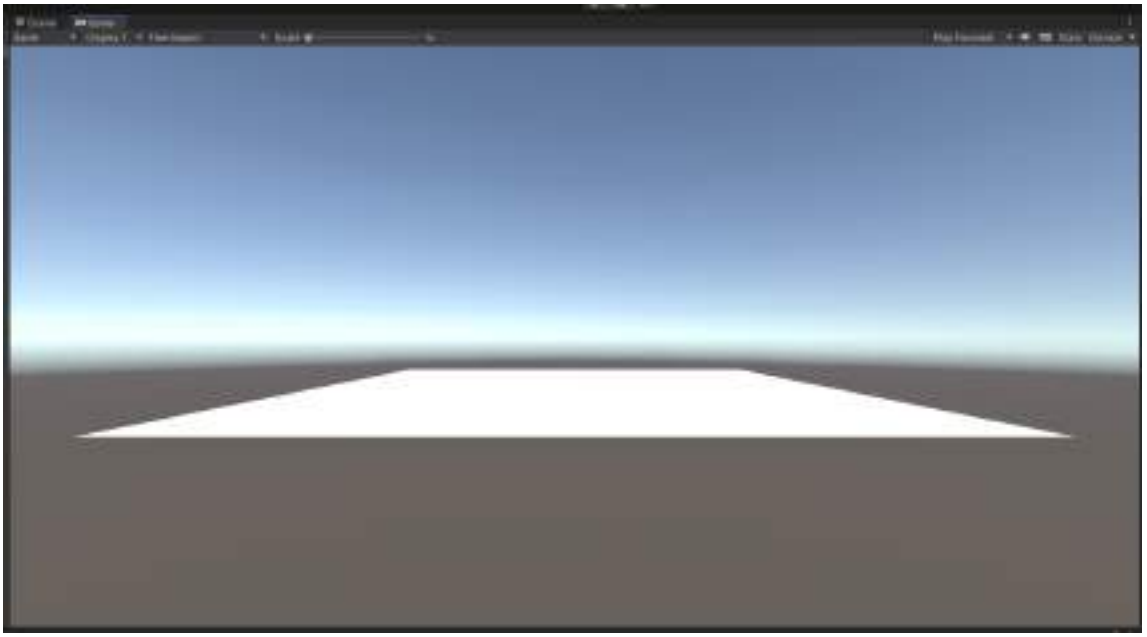


The plane will look like this:

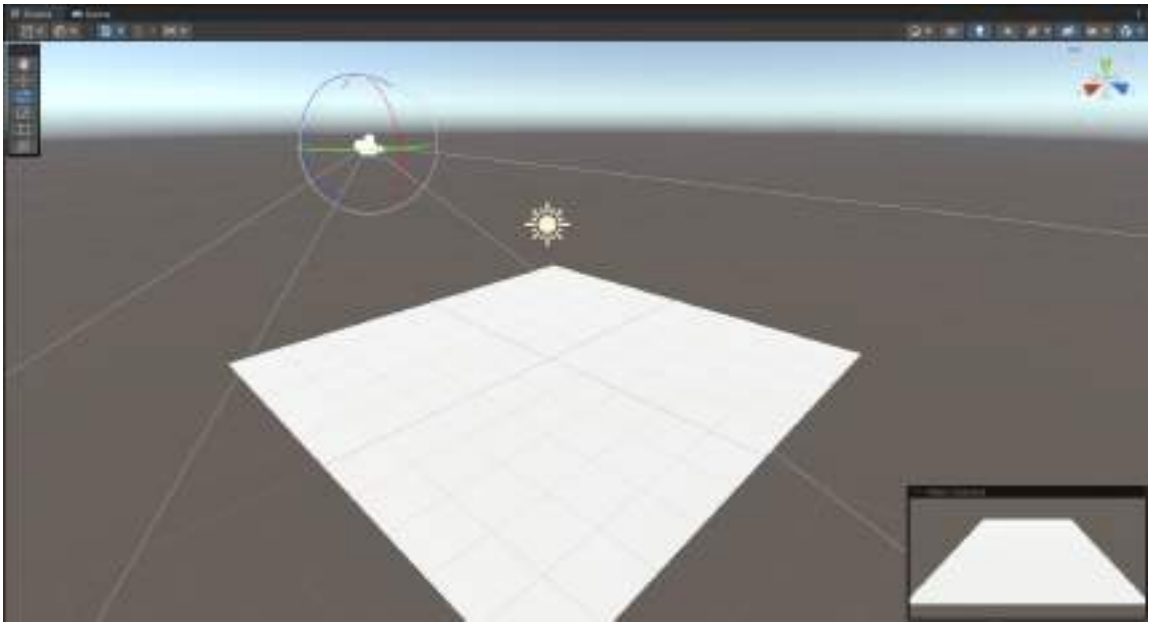


Select the camera and make sure the plane is framed correctly



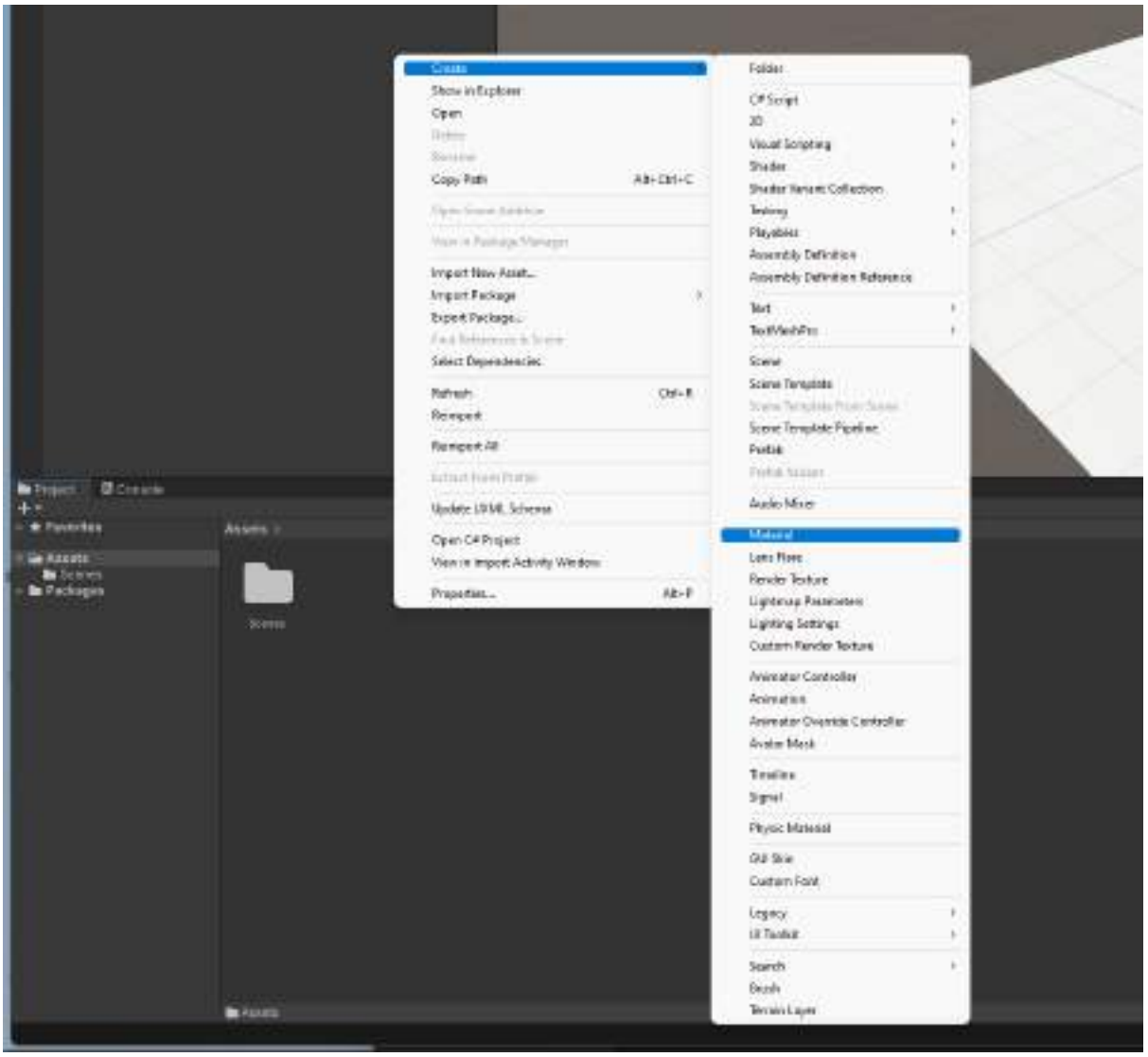


Move and rotate the camera with the appropriate tools on the left side of the scene view.



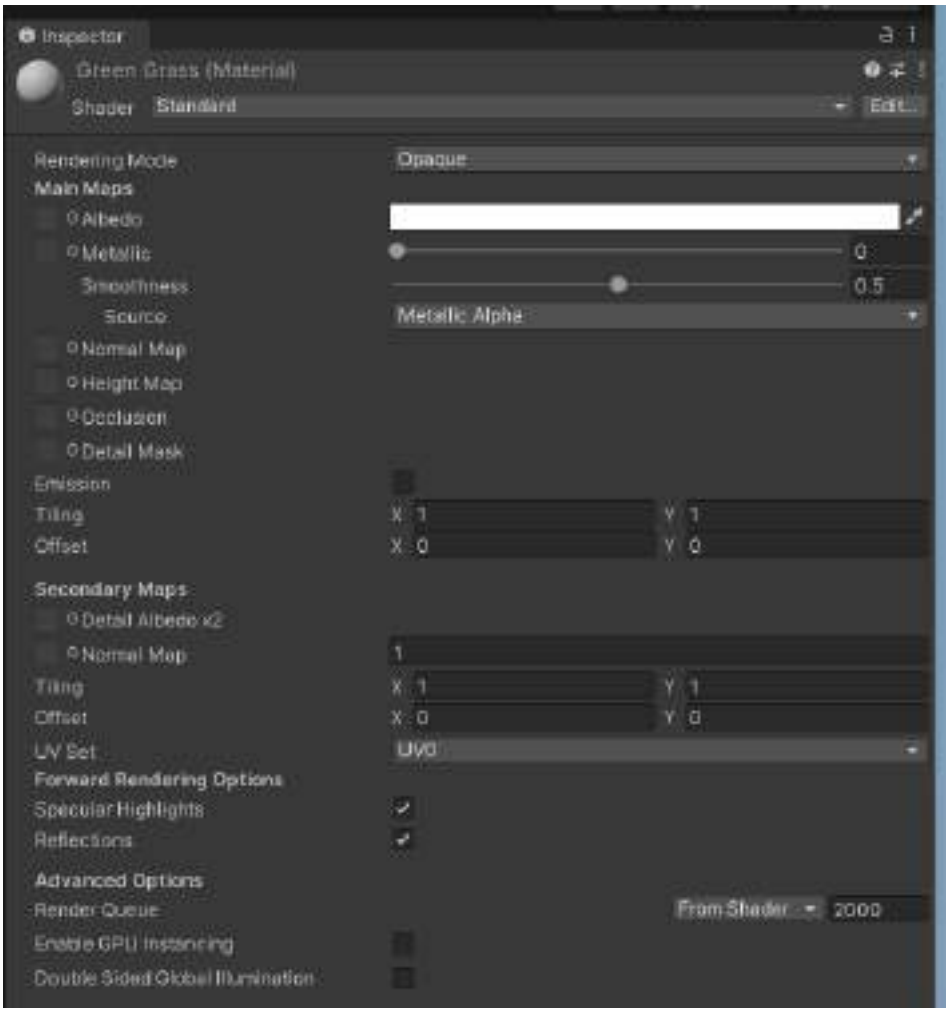
Let’s create a new material to colour the floor green.

Click with the left mouse button on the panel Assets, Create -> Material.

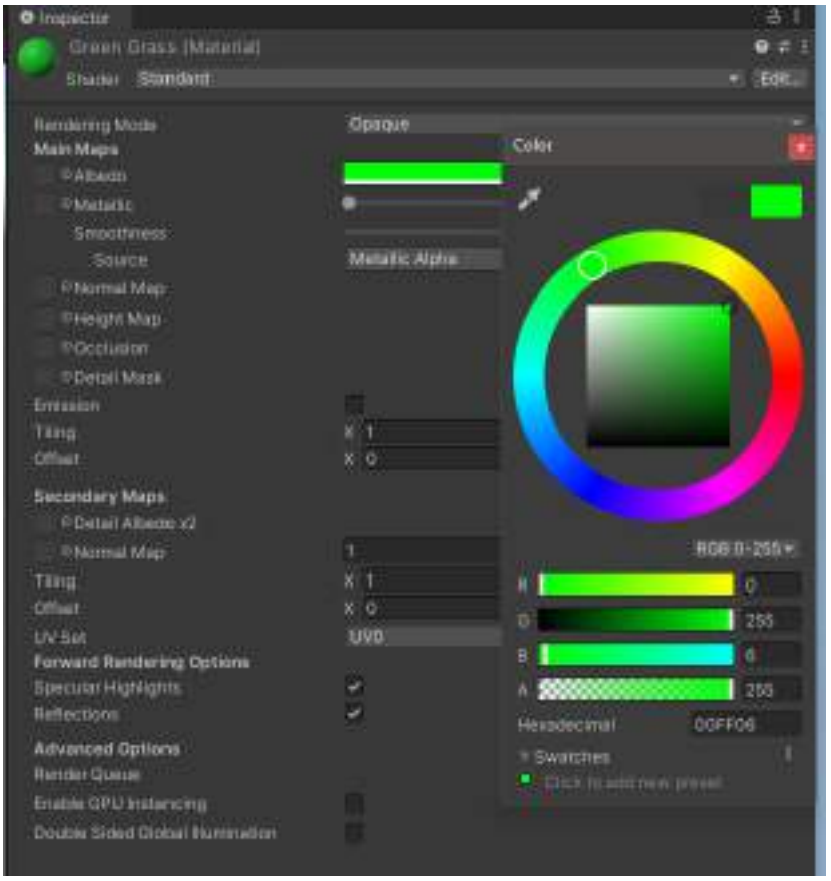


Let’s name the material

Set the colour in the "inspector" panel by pressing on the rectangle coloured in white.



Choose the colour.



Drag the material 

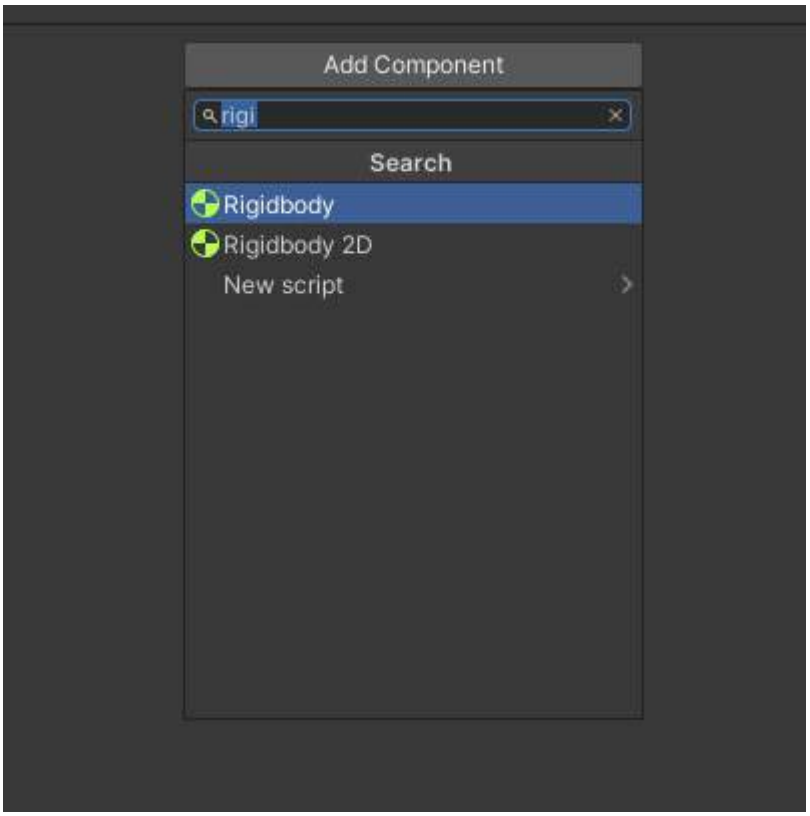
from the Assets and bring the plane into the scene view to change its material.

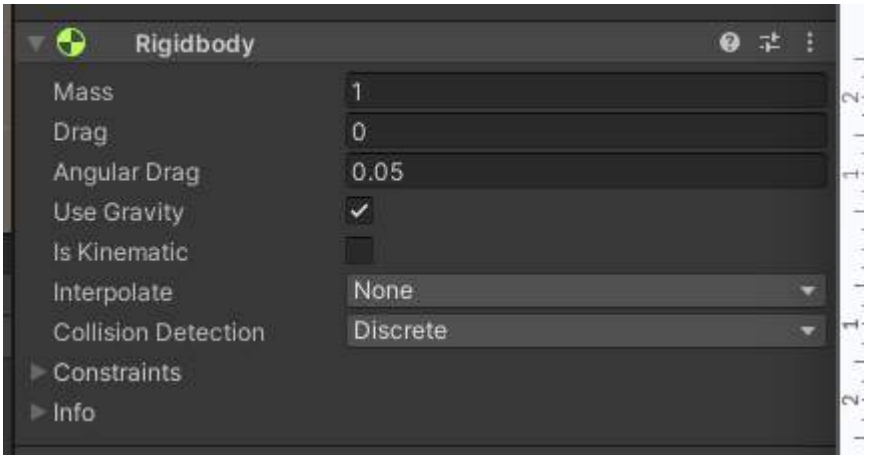
With the same method create a cube and colour it light blue.

Let’s add the Rigidbody component:

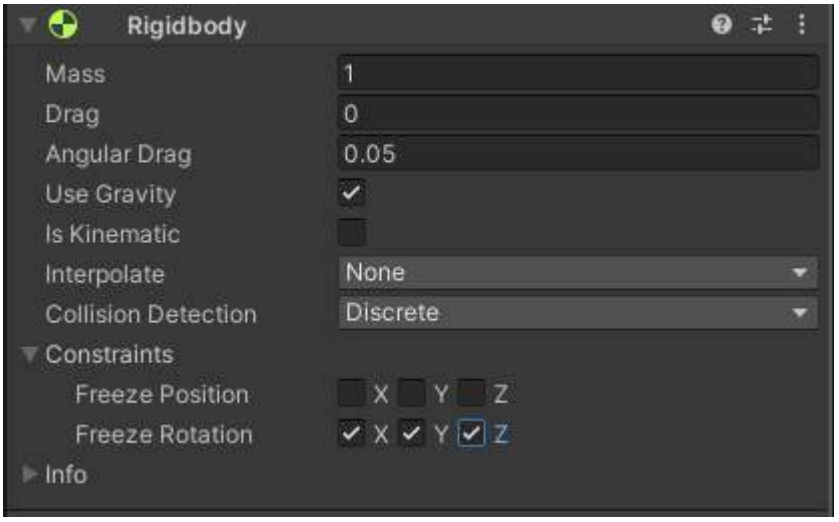
Select the cube, press Add Component on the Inspector panel, search and select "Rigidbody".

This will allow the cube to be controlled using physics and to be subjected to gravity.





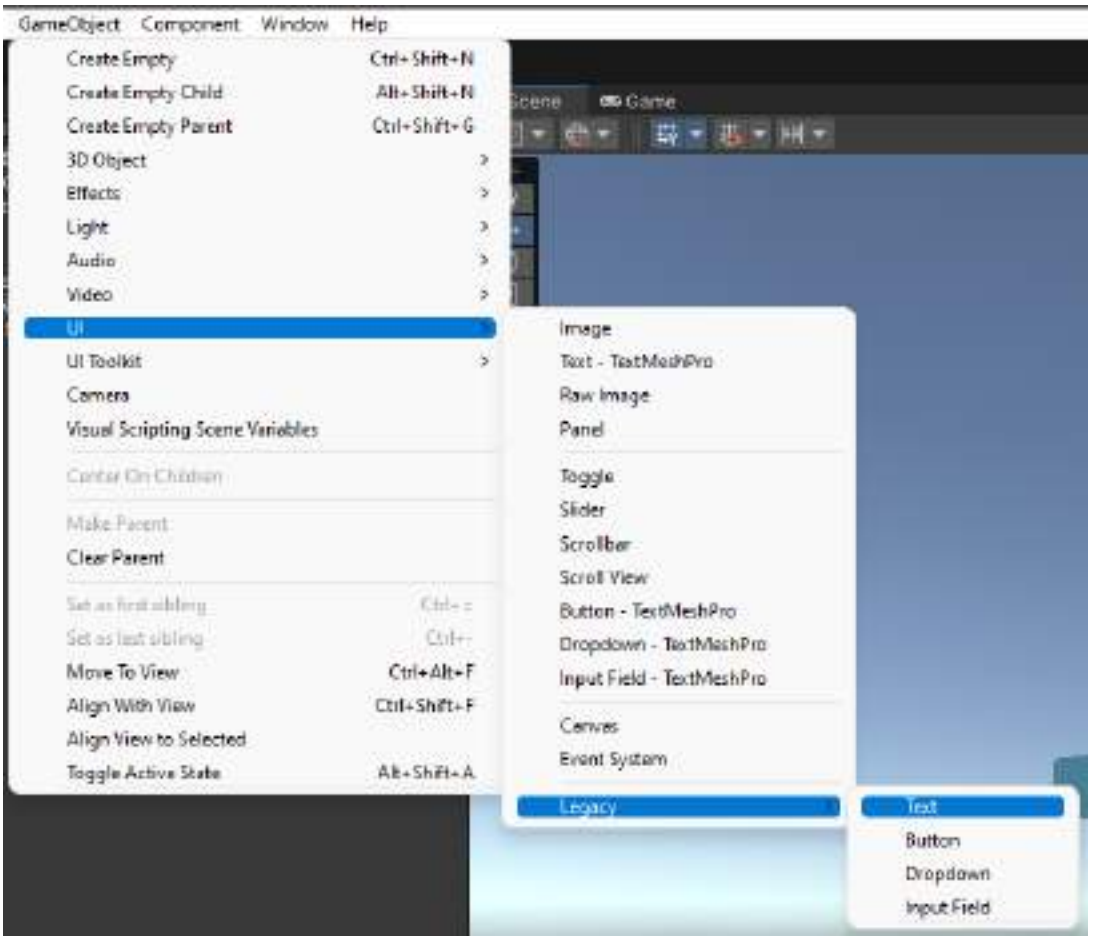
Select Freeze Rotation (X, Y, Z) if you want to prevent the cube from rotating while moving.



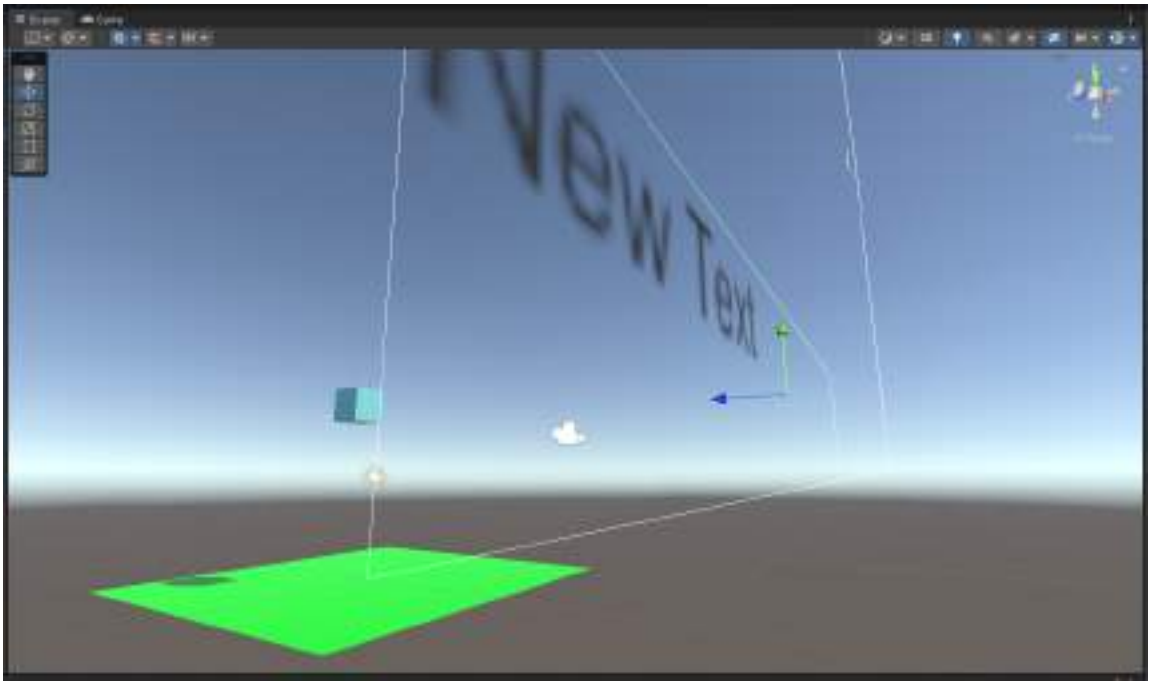
(tip: Instead of holding down the left mouse button and moving using the WASD keys you can select an object from the "Hierarchy" panel and press SHIFT+F to focus the view directly on that object).

Let’s add the overlaid text to communicate with the user:

GameObject -> UI -> Legacy -> Text



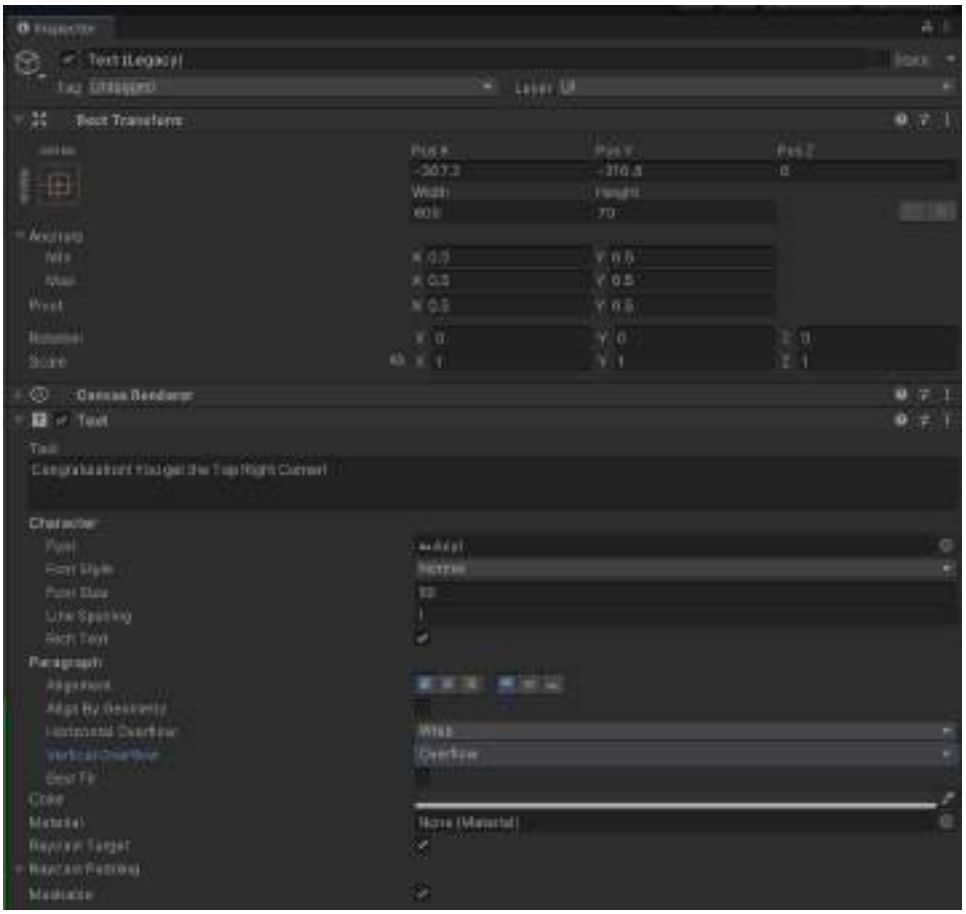
A very large "Canvas" (Canvas) will appear where we can add texts and other elements for the graphic user interface.

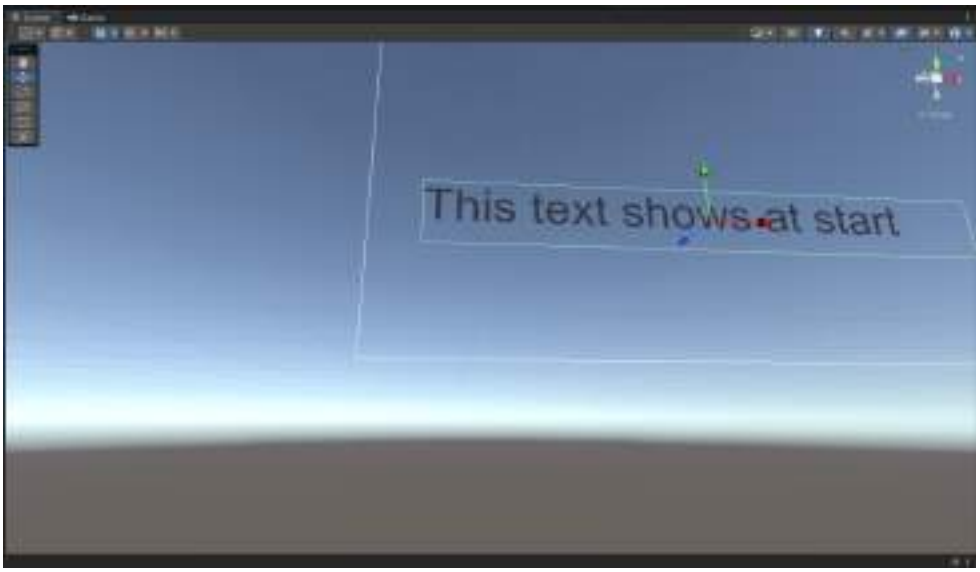


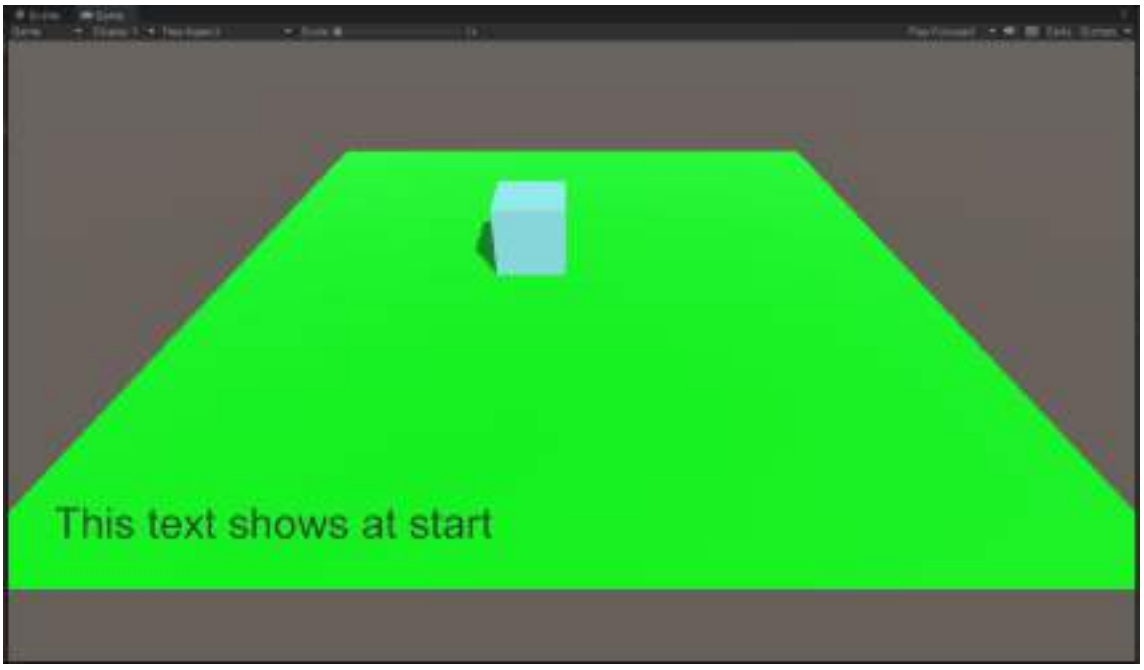
Select Text (Legacy)

Change width, height, text e font size from the Inspector panel;

Move the text to the position that you want.



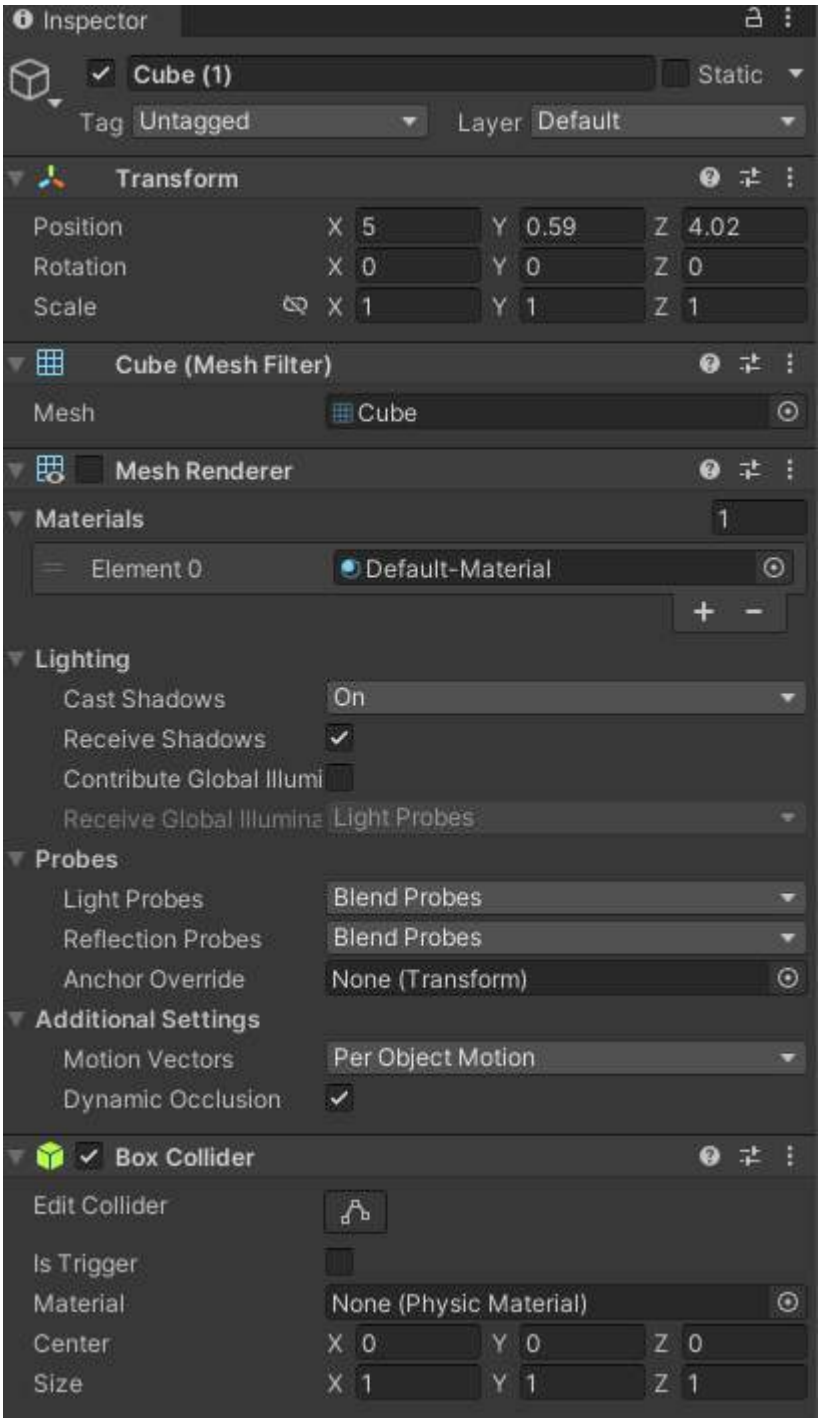


The position of the text in the canvas corresponds to the position on the screen during the game. 

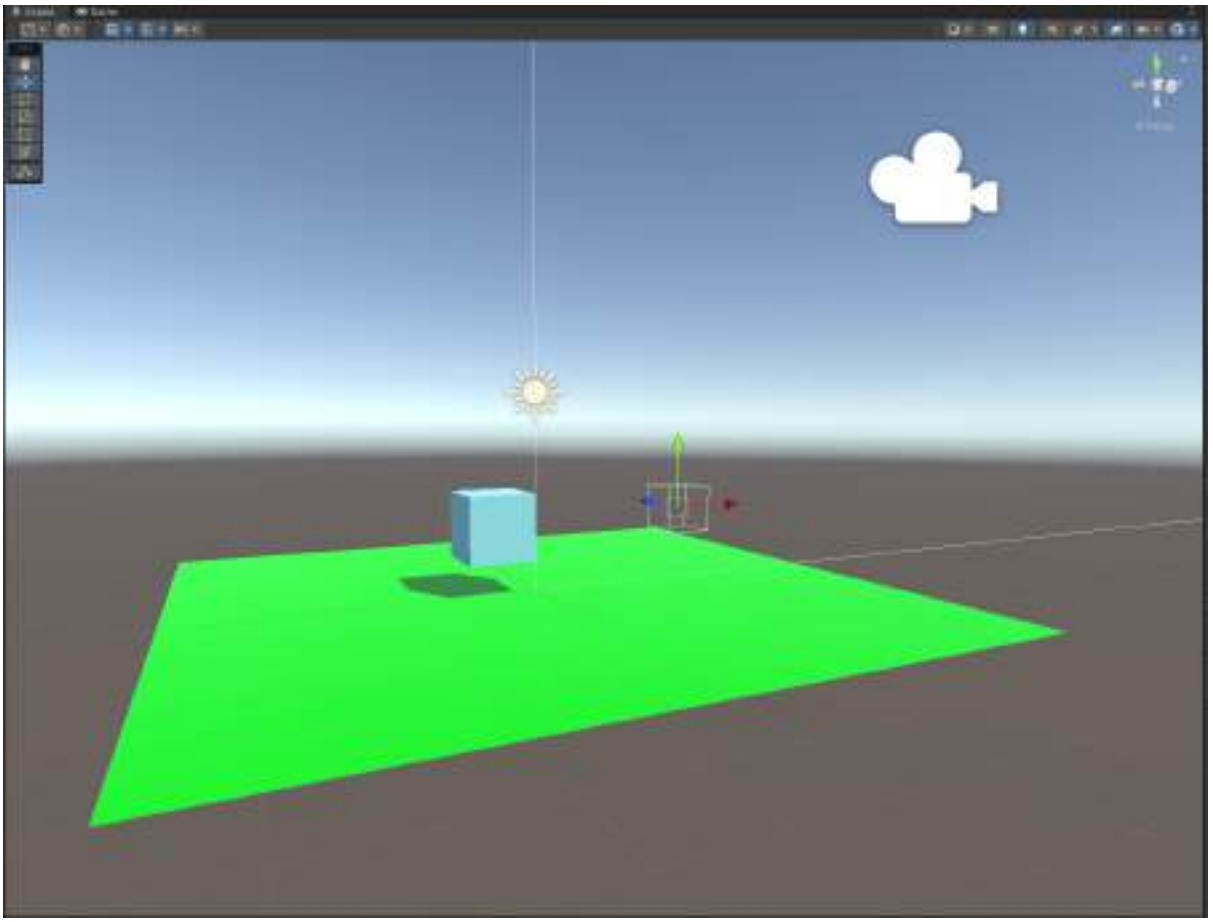
Let’s create another cube:

In Inspector -> Box Collider check the “is Trigger” box to transform the cube in an area that starts an event.

Uncheck the Mesh Render box to make the area that has been created invisible.



Locate the new cube in the top right corner of the plane.



Add a script component named TEXT.

Double click on the file that appear in assets and add the code instruction at line 8.

Public string textToShow;

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class TEXT : MonoBehaviour
{
    public string textToShow;  //pubblic variables can be used from other scripts

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

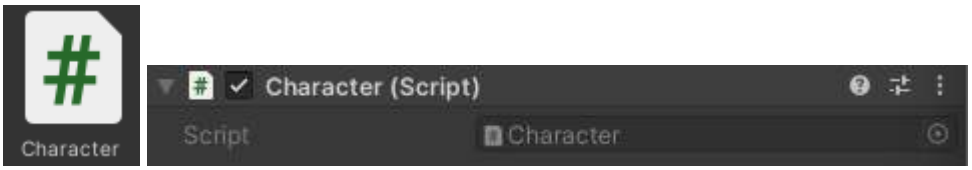
Save and close the code editor.

The added instruction enables us to use and set the textToShow string from the inspector panel.

Change the Text To Show value in *“Congratulations! You get the Top Right Corner!”*

Select the light blue cube and add a script component to it as well.

Name the script “Character” .



Open the script and add this line of code after the first brace before void Start()

```
    public float forceAmount = 0.8f; //questa riga imposta il valore della forza
```

Add the following lines of codes between the two braces of the Update function.

```
        Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
//ottiene la direzione utilizzando gli assi “Horizontal” e “Vertical” mappati sui tasti WASD o sulle levette analogiche
        direction=direction*forceAmount; /moltiplica la direzione per la forzat
        rigidbody.AddForce(direction); //applica la forzqa utilizzando il vettore direzione
```

These instructions will enable us to move our light blue cube using the WASD keys on the keyboard.

The code will be similar to this:

SAVE

Enable the ability to use UI related instructions by adding the following in line 4

```
using UnityEngine.UI;
```

Insert the following string after the declaration of "forceAmount"

```
public Text textToChange;
```

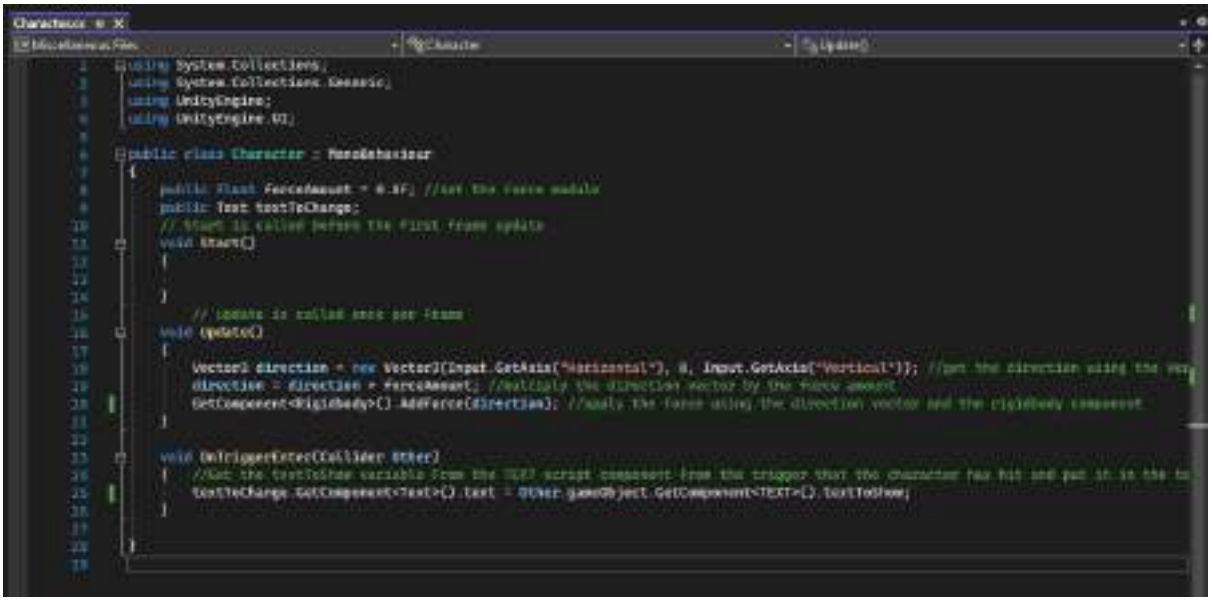
The above line of code will allow us to link textToChange to text in the Canvas using the Inspector panel.

Add this code before the closing of the last brace.

```
void OnTriggerEnter(Collider Other)
    {
        textToChange.GetComponent<Text>().text =
Other.gameObject.GetComponent<TEXT>().textToShow;
    }
```

The previous function will change the overlay text to the one in the invisible cube code set as the trigger area.

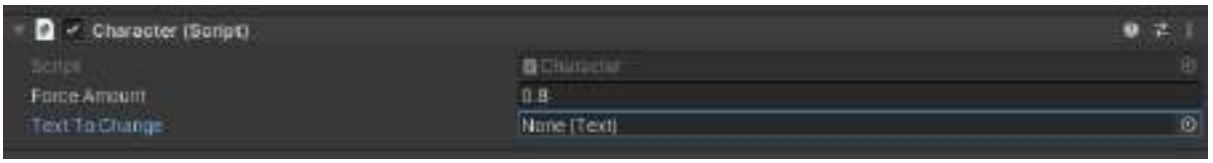
The code will be similar to this:



SAVE and close the code editor.

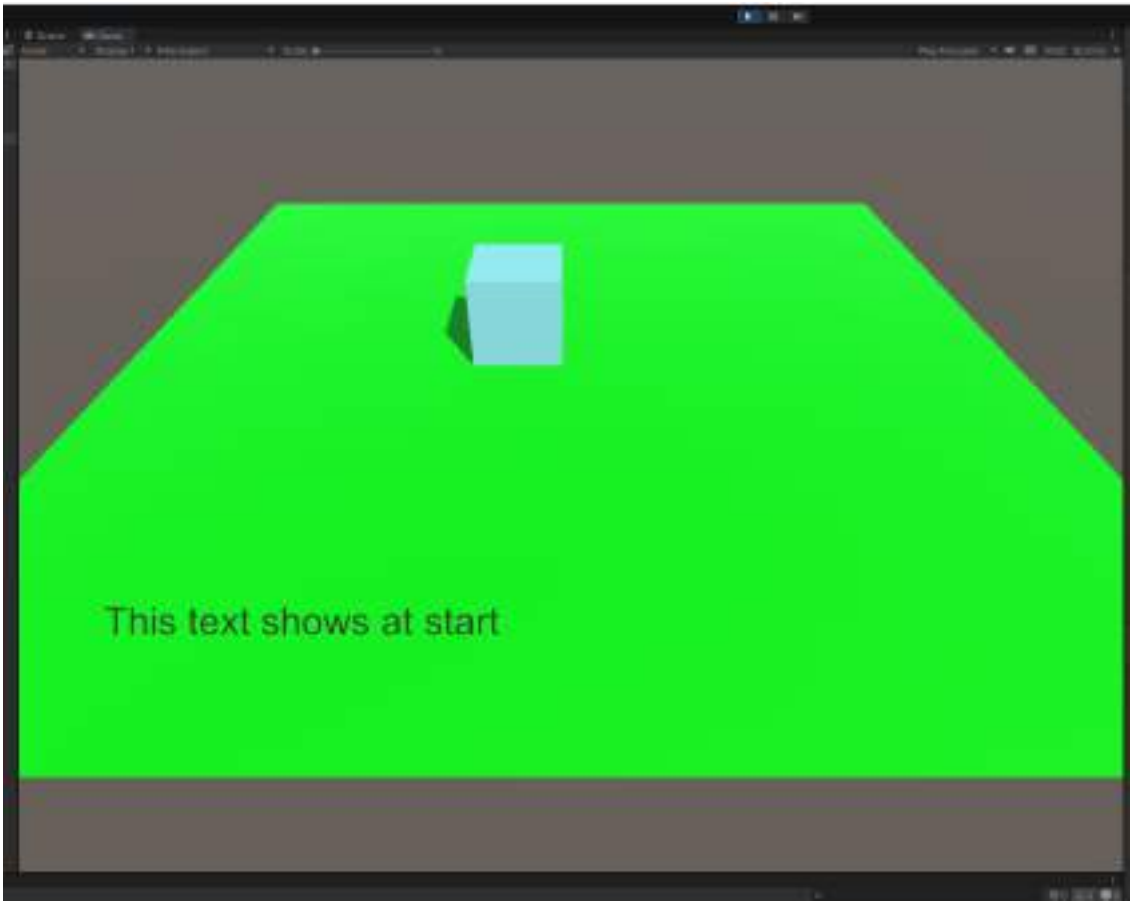
In order to change the text of the UI, it must be linked to the light blue cube using its Inspector panel.

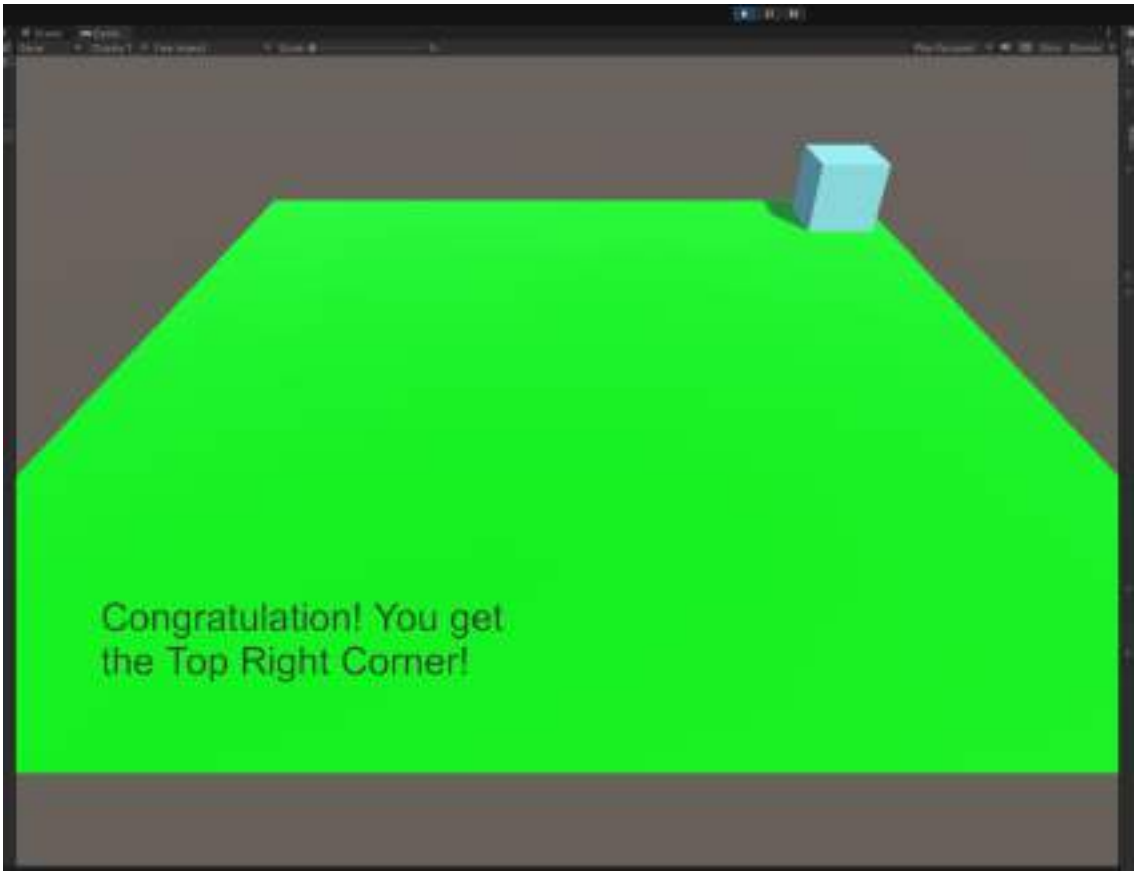
We set Character(script) -> Text To Change.



Drag Text (Legacy) from the Hierarchy panel to Character(script) -> Text To Change in the inspector panel of the light blue cube.

SAVE the project and press play to try our game.





We can copy and paste the invisible cube several times to create different areas that show different texts.


Cube (1)
Cube (2)
Cube (3)
Cube (4)
Cube (5)