Politecnico di Torino

1859

Master’s Degree in Computer Engineering

December 2022

Fairytool

Learn and socialize by making games

Supervisors:
Bottino Andrea
Strada Francesco

Candidate:
Aurigemma Daniele

I would like to dedicate this thesis to my brother Matteo, for teaching me that no matter how difficult life may be, you can always find a way to be happy

Contents

Table of Figures

Acronyms

VPL

Visual Programming Language

VR

Virtual Reality

AR

Augmented Reality

XR

Extended Reality

OS

Operating System

SDK

Software Development Kit

API

Application Programming Interface

UI	User Interface
GUI	Graphical User Interface
HUD	Head Up Display
DOM	Document Object Model
SVG	Scalable Vector Graphics

1. Introduction

1.1 The Goal

Videogames are one of the most successful media today, they quickly spread all over the world and in all age groups, especially among the young people, videogames are so attractive, but they also are a complex piece of technology. Graphics, physics, computer science, math and collaboration are just few of the skills involved in game development, and these make approaching this activity from scratch quite hard and frustrating. **Our goal is to create a System to allows users to create games in a fast and <u>fun</u> way**, a way that's make the game creation the game itself, a way that should intrigue the user and show him the rudiments of all the skills mentioned above, and that allows people to *experiment with technologies such as 3D, VR and AR*. Finally, the experience should **stimulate users to interact with other people, collaborating or showing their work to others**.

Or at least this is just the beginning, as the project has gained another incredible goal: **<u>evaluating the effectiveness of this tool in bringing benefits to autistic people and with other similar diagnostic pictures</u> <u>and being able to provide support to refine the diagnosis</u>**, but we will talk about this in more detail later.

1.2 The Target

Who needs something like this? Of course, the first answer is teenagers and children, young people who are fascinated by videogames, but who still must acquire the skills necessary for their realization. But why should they create one?

On the one hand we certainly have **those who are simply intrigued** by trying, on the other hand there are kids who might have to do with **school projects** or more generally with their **education**. Starting from this, we can extend our target to **everyone who wants to move the first step into the computer science world in a fun way** and having immediate outputs.

Other people that could be interested in this system are the **participants in contests** (e.g., Hackathon, Game jams) or **anyone needing to quickly create a game prototype**, who do not usually have too much time to spend in learning technologies and writing code. Is common that applicants to this kind of competition are not insiders, so they usually deliver documents instead of working prototypes.

And finally, as I mentioned before, another group of people who can benefit from the use of this project are **autistic people or people with similar diagnostic pictures**, although in this case, in addition to the mere use of the software, there would be a **need for a real path to be addressed**.

1.3 The System

How could this be possible? The first idea was to write some sort of plugin/addon to simplify an existing game creating system (e.g., Unity, Unreal), but questions immediately arose: are these solutions too storage consuming? Do they installation require too much time for our target? Is exporting games, especially on mobile or VR systems too complex? Are these systems too complex to learn? All these questions lead to the adopted solution: web application. With a **web-based solution**, user do not need to install anything and sharing games and projects across different devices is easy as sharing a web link.

We figured out how to the system technology should looks like, but what about the user experience? The first thing that came to our mind was to completely avoid coding. Learning how to code requires time and learning how to code a game requires more time. Another problem is that game assets such as 3D models and sounds are really time consuming to create and implement. So, the final idea was to create a **block-based programming system** with **readymade assets**, everything surrounded by the most minimal **user-friendly interface** possible.

We will deeply discuss all these aspects later this reading.

1.4 The Game Engine

Is there already a solution to this problem? Someone could say “what about game engines?”. **Game Engines are software that provides everything needed to create real-time interactive applications**. They are usually commercial products made to allow user to create commercial games and applications, so they do not care too much about installation requirements or ease of use. Of course, some exceptions exist, and there are also products that aim to a similar goal, but we will look at them in the next chapter.

1.5 The Laboratory

Where this software should be developed? Of course, regarding technical and organizational aspects I surely had to mention the *CGVG* department of the *Politecnico di Torino*, but making something like that is not only a technical challenge, but also a complex design task, so to make it possible I need to study and understand my target, and here comes the studio of **Doctor Claudia Amoruso**, the ***Polistudio Amoruso*** (in Italy), that in addition to give professional psychologists support, it also allowed me to follow and study an heterogeneous group of young people (from 10 to 28 years) with different psychological diagnostic pictures, including ***Autism*** and ***High Intellectual Potential***, who are **constantly followed by specialists** in the psychological field, but united by a passion for video games! I had the opportunity to hold **meetings with this group on a bi-weekly basis**.

We will get into details about this later in that reading, but I had to thank her and the boys in advance.

1.6 The Boys

But who are the guys who will accompany me along this tortuous path? To answer this question, I will use the information provided by Dr. Claudia Amoruso, who personally follows all the participants.

Common features found in participants:

- High cognitive potential.
- Great emotional sensitivity (psychological pain and anger).
- Difficulties reported in psychotherapy related to peer attendance.
- Expression of the pleasure felt in gaming.

Characteristics of the subjects:

- All subjects involved have conducted psychotherapy with a *Transactional Analytical* address.
- All subjects have a cognitive assessment.
- All the subjects reported in the interviews, in addition to the pleasure of playing, that of experimenting together in programming.

Common contents:

- Experiences of solitude in childhood
- Personal beliefs:
    - "I will never fall in line"
    - "I usually prioritize things in an original way"
    - "I don't want to become what society (family, peers) thinks of me", "bad"
- explicit pleasure in being of help to others.
- Rigid value systems.
- Anime/manga mirroring.

For the good resurrection of the experience:

- Based on the needs expressed: **<u>fun</u>**.
- Based on needs reported in psychotherapies: **<u>socialization</u>**.

The hypothesis formulated by Dr. Claudia Amoruso is that:

- the pleasure of playing,
- the reflection provided by the group itself to its participants,
- sharing programming tasks,
- the sharing of a ‘coldly’ language that warms up ‘as needed’ in the interaction between participants,

Together with other variables to be identified during the research, the synergy and collaboration of professionals as an effective response to taking charge of subjects with high-potential autism as a complex and adjustable response to complexity.

1.7 The Name

Why is it called ***'Fairytool'***? I would like to end this introductive chapter with the explanation of the name choice for this software. While experimenting with the boys one day I asked them how should I name the project, they did not give me a precise name, but they all agree one thing, they want to express themselves through this *tool*, they want to **tell stories**, and what was the first way to tell stories? *Fairy tales*! So, the name is simply the union of these two concepts.

This is enough, let us start with our analysis.

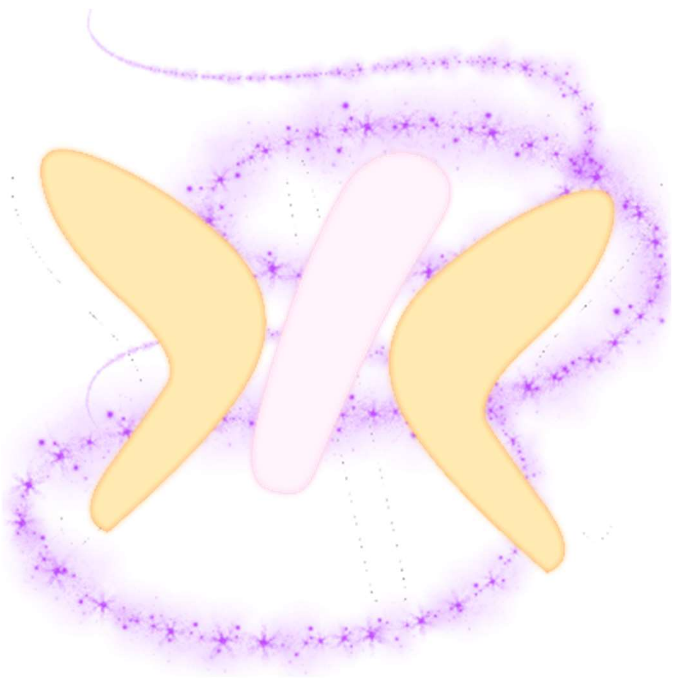


Figure 1: Fairytool Logo

2. State of The Art

At what stage is the research and development on this topic? There are (and were) a lot of projects with similar goals, but luckily just a bunch of them out there are interesting for our purpose, let us see why.

**a full list of the examined tools can be found in the Sources chapter of this work*

2.1 Determine similar tools

Hundreds of games making tools exist out there, are all of them meet our requirements? Of course, they are not. We will analyze our project features to determine which are the ones that should be considered.

The first factor is **graphics engine**, the majority of the similar tools are actually developed thinking about 2D graphics, which is completely fine, but most of the games that people play are actually 3D, so a three-dimensional approach would be more attractive, and it also put the basics for a more scalable and portable environment, where different camera settings and perspectives can be chosen, and different devices and technologies can be targeted.

The second factor is the **learning purpose**. Many game making tools are meant to allow people to create games, not learn how to do it. So, they are not designed to be used from scratch, some of them have complex user interface, some others have a hard to learn programming environment, other ones are difficult to setup and install.

Finally, the way they approach **modern technologies**. Some of the beginner-friendly game engines do not target technologies such immersive Virtual Reality or immersive Augmented Reality, some do not even care about mobile platforms.

Let us finally talk about what exists while I am writing. I will analyze these materials ordered by action done to use the tool itself.

2.2 Platforms

What should be the target platform for a project like that? There are similar tools available for almost every existing platform so let us examine them closely:

- **Desktop:** Of course, almost everyone has at least one PC in their home, so it is not surprising that different developers have targeted this platform with their products. Here, high technical performance can be reached if the software is well designed, but as a downside, we are requiring our users to perform an installation, in addition, software design becomes much more complicated and less scalable and portable.
- **Mobile:** Speaking of things that anyone has, the smartphone could not be missing. Several developers have chosen to create or port their environment to make it available on mobile platforms. Here the user interfaces tend to be very simplified due to the small screen and scarcity of available input systems. So, the overall experience tends to be very limiting, but some clever ideas came out from those limits, especially regarding 3D space object positioning, which is often too complex even on desktop platforms.
- **Game consoles:** Players (especially the youngest ones) are surely an incredibly good target for this kind of project, and which is the best place to find players? Game consoles. It seems perfect, but in fact there are only two valid products available on these platforms, *Dreams* for Sony PlayStation, and *Game Builder Garage* for Nintendo Switch. Game consoles are a great commercial opportunity, but if we want to make something that should be cheap and accessible, they are not the best choice.
- **VR Headset:** Many big companies are investing in these technologies nowadays, so it is natural that some steps are also taken in their applications development. Since these technologies are quite new, there are not many available and stable options. One of the biggest is Meta's *Horizon Worlds*, but It is still under development. Also, VR headsets are still quite expensive and very few people have access to them. But despite this, it is an incredibly attractive technology for people.
- **Web Application:** If I had to think at the most portable technology, I must cite web applications. Web-development is one of the most studied and developed technologies of the last years, so it is not surprising that we can easily found several
inherent products developed for this platform. Unfortunately, it has some limitations, and it is less performing compared to other platforms, but a rich

assortment of libraries and frameworks, its spread and a no-download approach make it a suitable target for these purposes.

2.3 Technologies

Most of the existing tools are intended for 2D graphics. It is easier to implement, easier to be managed by the end user, and it does not affect the possible educational purposes. However, kids and more generally **people who play videogames are used to seeing 3D graphics**, so this kind of approach is more familiar to them. Of course, make 3D games easy to create is a bigger challenge and it needs many compromises, but it surely is an attractive approach, and it is not surprising that some of the most popular tools have chosen to try to follow this path, especially when they are designed for gamers (e.g., *Game Builder Garage, Dreams, Roblox* and more). But they still require the users to learn many things before they can be confident with the tool, and this could be seen as an obstacle from people who want something to get a result as fast as possible. Speaking about other technological aspects, no relevant tool offers a built-in **'technological playground'**, so they are usually focused on some specific platform or technology. Some of the most attractive technological aspects that may or may not be supported by existing tools includes **gamepads, mobile platforms, XR (extended reality), multiplayer games**. These elements are not essential to teach game development or coding, but people are intrigued by them. Game console tools usually includes support for technological peculiarities of their devices (e.g., *Nintendo Switch, Sony PlayStation*), Some are mobile oriented (e.g., Struck), and very few aim to work with XR (e.g., *Horizon Worlds, CoSpaces Edu*).

2.4 Assets Management

How do existing tools handle assets such as 3D models, sounds, and so on?

Most of these tools use a ready-made assets library. This is an understandable choice since many issues must be handled, for example:

- ***How can the user have a custom asset added to their game?***
- ***How to protect users from sensitive content? this brings a new problem***
- ***How can the platform handle copyright issues? (This only applies to online shared content)***

The first issue can be solved in three ways:

- ***Creating a functionality that allows users to create assets within the platform.*** This could give more chances to the platform to prevent sensitive content creation, and it also avoids the possibility of copyright infringement. However, this leads to the problem of how the tool can simplify the assets creation, because for resource such as 3D Models the process is anything but trivial.
- ***Letting the user import their own assets.*** Doing this on an online platform could raise copyright related problems. Furthermore, the user should be conscious of the existence of limitations such as file formats or file size.
- ***Allowing the user only using models from a ready-made asset library.***

The other issues are of a legal nature; therefore, they will not be dealt with in this thesis.

Some offline tools, especially the 2D ones, offer a full control on assets management, but many of them are limited to pre-constructed libraries, or they let the user create certain type of resource, for example Game Builder Garage (*Nintendo*) only allows the user to create 2D images, or Dreams (*Sony*) gives the user a tool for sculpting 3D models but not a way to import them from an external application or device.

2.5 Programming System

Traditional text-based programming languages are surely extremely powerful, but they need to be studied, practiced and they are sensitive to syntax errors, situations that could represent obstacles for newbies.

VPL (Visual Programming Language) can solve and/or mitigate most of these issues.

We can classify VPL using the following taxonomy:

- **Block-based**:
  These languages are characterized from allowing the user to drag and drop blocks from a toolbox containing a predefined set of commands into a workspace area. Here blocks can be nested together in a jigsaw puzzle style.

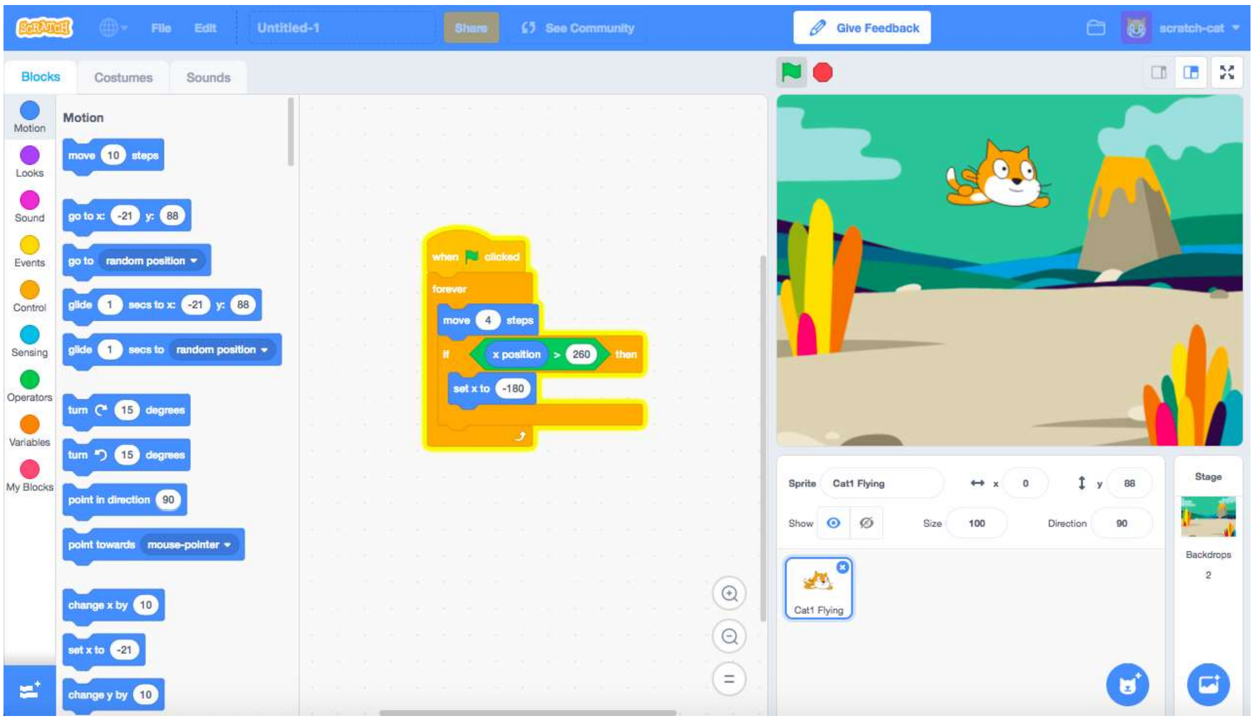


Figure 2: Scratch, example of block-based VPL

- **Icon-based:**
  Icon-based VPLs, as the name suggests, heavily rely on the use of Icons to describe objects and actions. These icons can be elementary, if they represent objects (e.g., file) or actions (e.g., add, remove), or complex, composite icons obtained by assembling elementary ones to create "visual sentences".

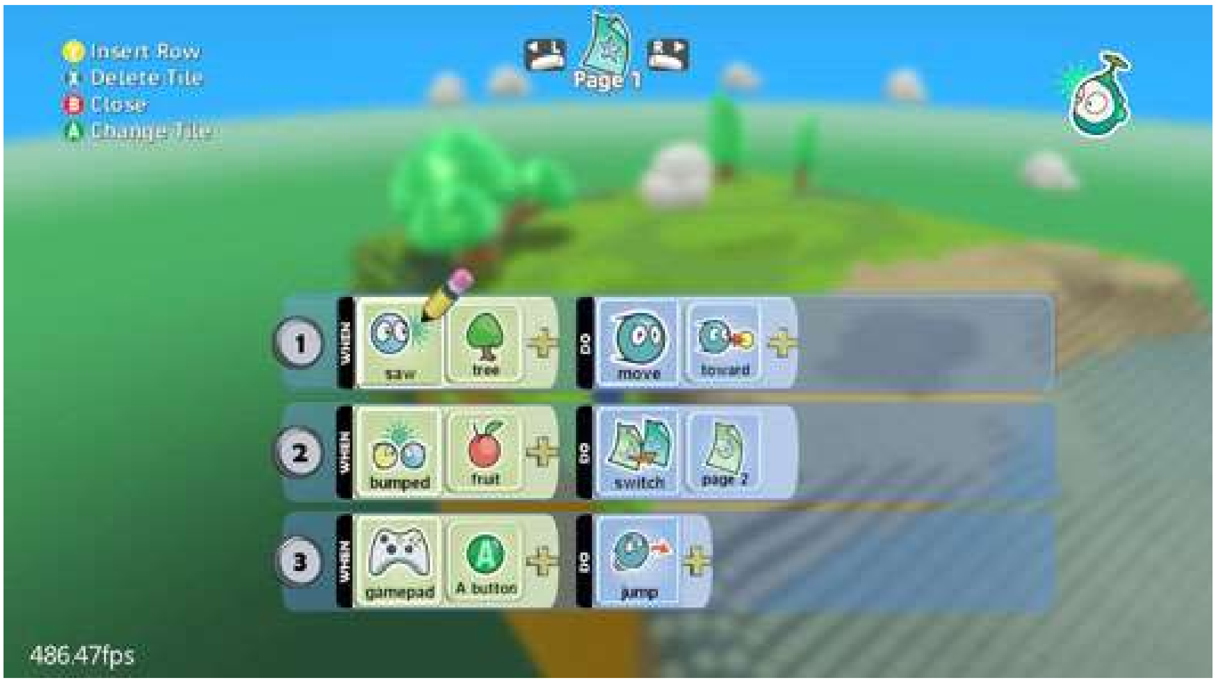


Figure 3: Kodu, example of icon-based VPL

- **Form-based:**
  The end-user is required to compile forms related to specific features or functions, using various kinds of controls, such as drop-down menus, text field, and more. The inserted values can be used to customize the behavior of the current form functionality or to reference others.

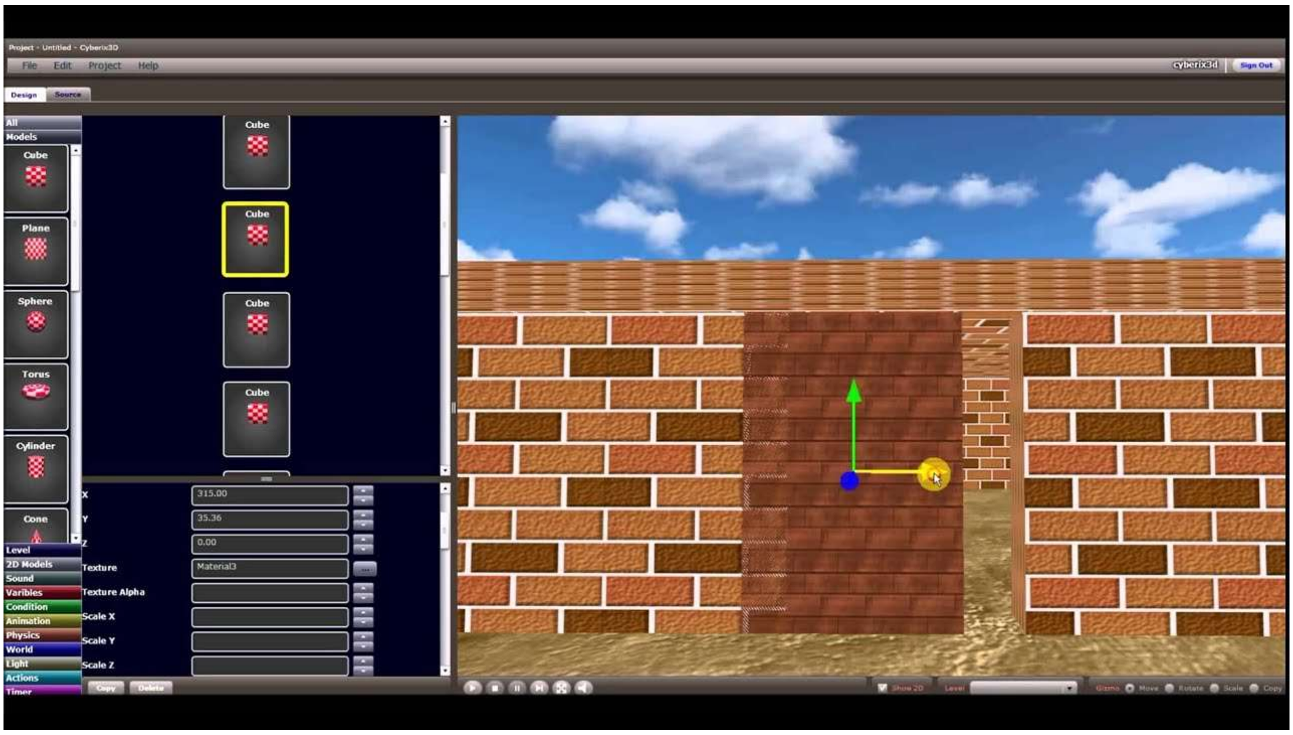


Figure 4: Cyberx3D, example of form-based VPL

- **Diagram-based:**
  They are similar to block-based VPLs, users still drag and drop blocks into a workspace, however here the outputs of a block have to be connected, using a line or an arrow, to inputs of other blocks, creating diagrams or flow-chart. Usually blocks can also be rearranged in the workspace without changing the semantics of the created program.

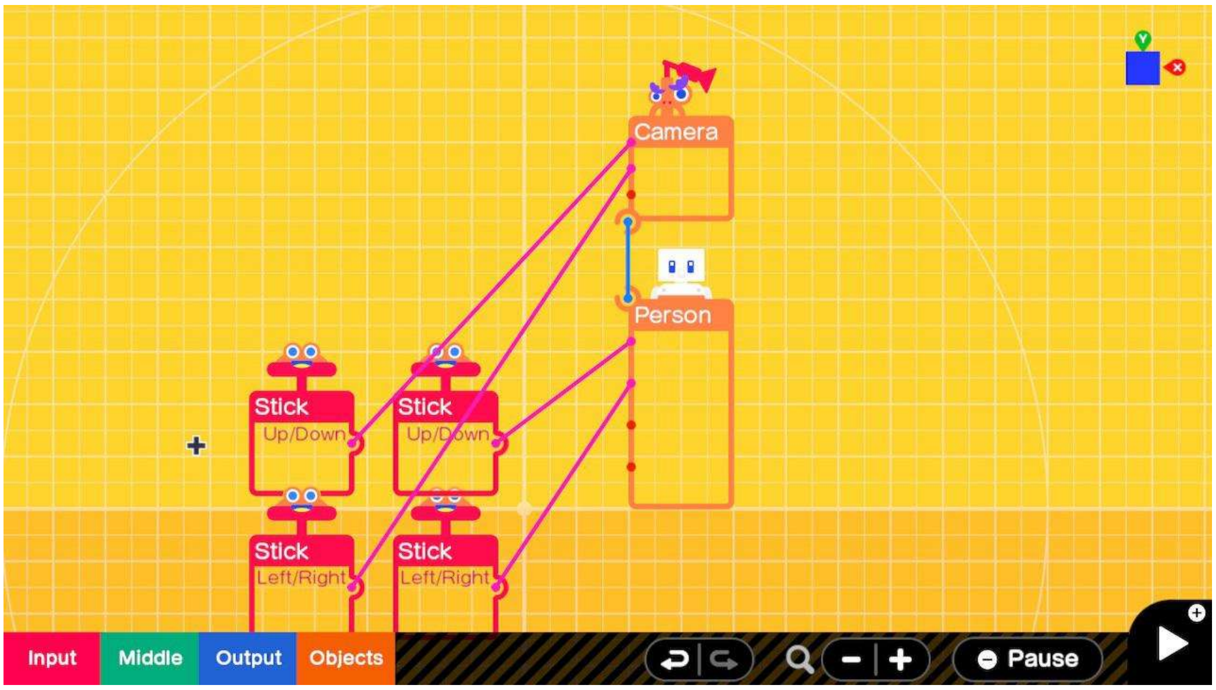


Figure 5: Game Builder Garage, example of diagram-based VPL

Speaking about game engines and game development, the most common VPL style is the *diagram-based* one, also used by major products like *Unity* or *Unreal*, however the huge disadvantage of this approach is the spatial complexity of the created programs quickly increase while inserting and connecting new blocks, due to the arbitrary block positioning and the presence of lines that connect inputs and outputs, making programs confusing and hard to read.

Form-based and *icon-based* VPLs are the simplest ones to use, but they limit a lot the expressivities of the language, they are also the least common to find. However, they have some interesting aspects, icons increase the readability of the programs and help to immediately identifying pieces, and form-style controls further decrease the possibility of user errors, and they are good to describe higher level functionalities.

But keeping apart commercial game development tools, the most common VPL style surely is the *block-based* one, it is intuitive, it reminds the traditional text-based programming, and using some shrewdness (that we will discuss later in the *Visual Programming Language* chapter) readability and spatial complexity can be extremely improved.

3. Evaluating Existing Tool Effectiveness

Surprisingly, for the creation of the existing tools, no formal (or at least non-public) experiments have been conducted and their development and maintenance are based on an empirical method involving a cycle of feedback collection from their users both in terms of aspects and regarding the User Experience, and their possible implementation.

After the preliminary analysis I had to find a way to evaluate the effectiveness of existing tools. As written in the *Introduction* chapter, to do this I collaborated with *Polistudio Amoruso*, which gives me a sample of target audience and the support of professionals in the field of psychology. The group of testers was small, about ten elements, but interesting thing was that they were constantly followed during the whole period of the experiment *from 23rd April 2022 to 26th November 2022* which includes both the existing tools effectiveness evaluation, and the *Fairytool* testing phase.

3.1 Before Starting

Before starting a **preliminary survey** made on Google Forms **was shown to the kids**. This Survey aimed to collect information about people interest in the video games world, including favorite games, playing time, interest in game development, VR and AR experience and prior technical knowledge. The survey was compiled by people while talking, chatting, and discussing about the topics included in the survey. This created a good and comfortable environment and initialized a communication channel between the group members, who, as I wrote in the Introduction Chapter, have different social relationship problems. The role of this survey was to **make people introduces themselves**, so no standard references were taken to write it, it was created based on our goal and revised both by the thesis supervisors and Polistudio Amoruso.

3.2 Method

The idea was to see **how kids react and response to known existing similar tools**. And since the time for the experimentation was not so much, I have decided to examine the most used one, **Scratch** by MIT. *Scratch* is already used in schools as a didactic tool, it has a block-based VPL similar to one I was thinking of based on the previous chapter analysis, and it is free with a huge community behind it, so people can easily get it, on the other hand, it is a 2D oriented tool, but its functionalities are enough to reach our goal. The other tools were statically examined, and the results of these analysis can be found on the previous chapter.

- At first, a **brief introduction on game development** was made. I told them they were going to create a game **prototype**, so they should not care so much about the result. I also explained that a game work thanks to a mechanism called **game loop**. A Game is an infinite loop (that usually iterate at least **60 times per seconds**) that continuously **processes user inputs** (e.g., gamepads, keyboard, mouse), **updates** objects status (e.g., physics, attributes, etc..) and draws them to the user screen (**rendering**)

```
// Game Loop Example
 while (true) {
     processInput();
     update();
     render();
 }
```

- Then people were asked to individually **recreate** a custom version of the well know ***Google Chrome Dino* game using *Scratch***, trying as much as possible to do everything **by themselves** or **collaborating** with others, I would intervene only when necessary.
- Finally, an **end of experience questionnaire survey** was compiled by the group. The goal of this phase was to collect information about system usability, tool effectiveness and overall user experience, including fun and relationships. This survey was created including the standard S.U.S. (*System Usability Scale*), a set of questions adapted from *The Game Experience Questionnaire* [46], and a custom set of questions revised both by thesis supervisors and the psychologists.

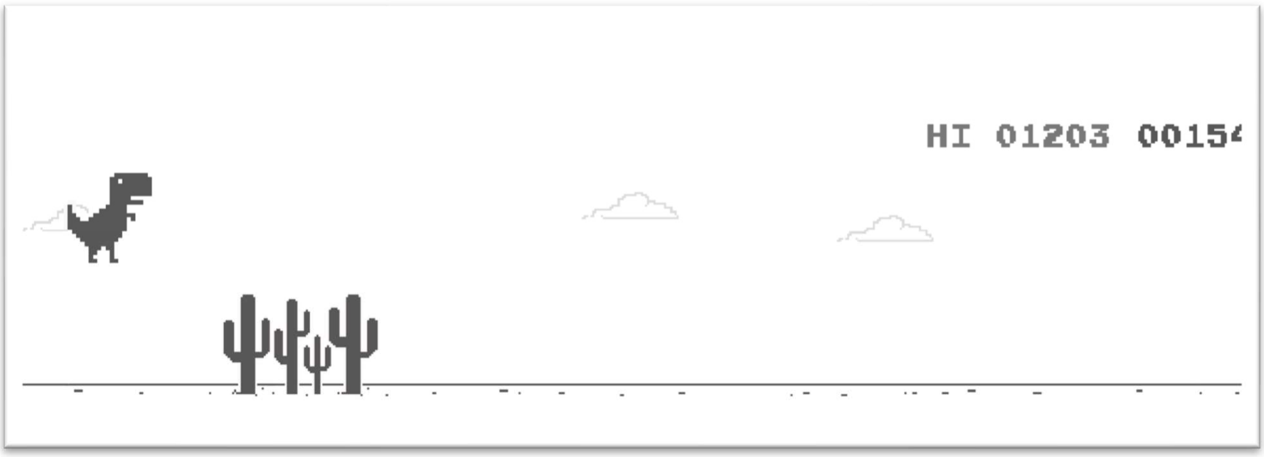
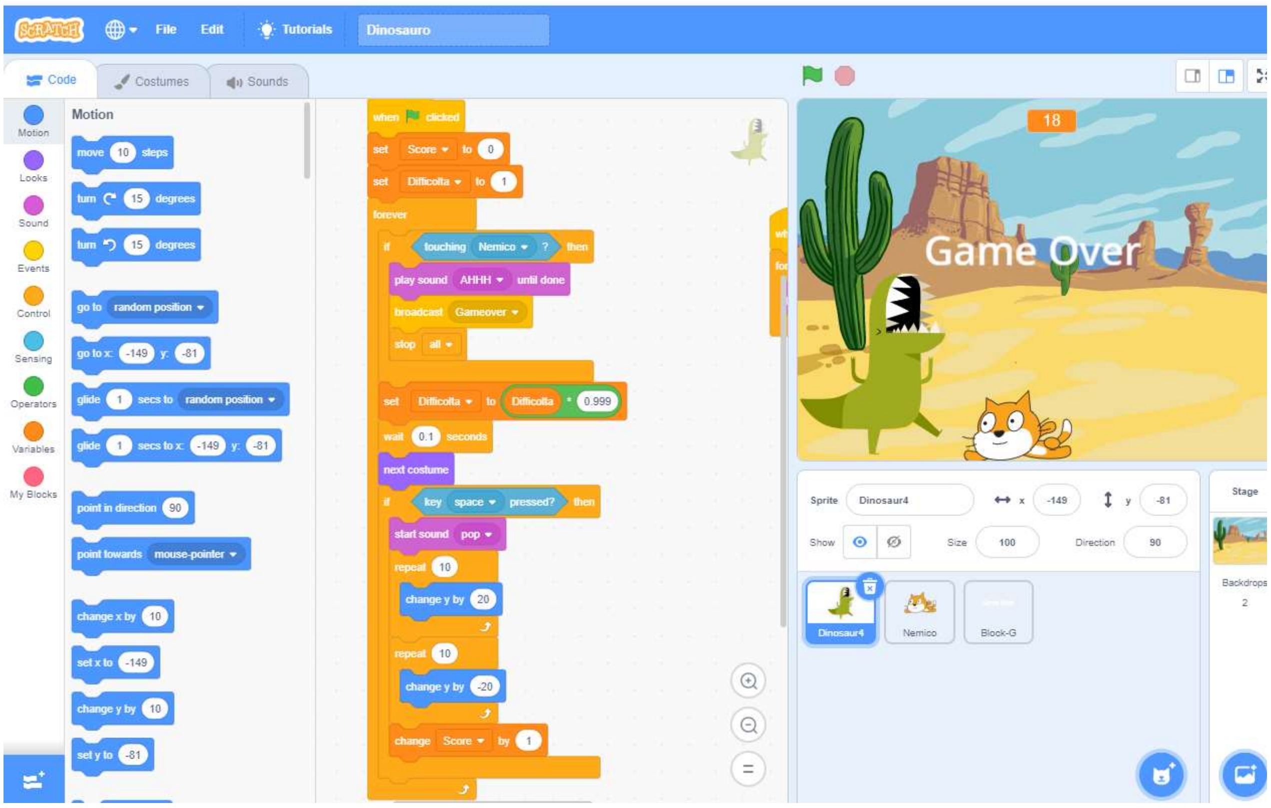


Figure 6: Original Chrome Dino Game



```
when [flag] clicked
set Score to 0
set Difficolta to 1
forever
    if touching Nemico ? then
        play sound AHHH until done
        broadcast Gameover
        stop all
    set Difficolta to Difficolta * 0.999
    wait 0.1 seconds
    next costume
    if key space pressed? then
        start sound pop
        repeat 10
            change y by 20
        repeat 10
            change y by -20
        change Score by 1
```

Figure 7: Chrome Dino Clone made using Scratch

3.3 Results

Here are the **key points** that emerged, based both on **first-person observation** of the group and on the **result of the end-of-experience questionnaire**.

Technical aspects related feedbacks:

- + The group found the **color breakdown of the various categories of blocks** extremely helpful, even though the same color was shared by too many blocks making it more difficult to distinguish between them.
- + A much-appreciated feature was the **library of predefined assets** (sounds and images), as it allowed to speed up the game creation process, without necessarily having to import your own files, an operation that was not found easy due to its confusing graphical interface.
- + The most appreciated part was the **block programming system**, which allowed the students not to have to memorize instructions and to completely avoid syntax errors.
- + Some kids have found the possibility of being able to **import custom images and sounds** amusing.

- - Using *Scratch* was complex for most of the kids, as **they found it difficult to navigate the interface** and find the needed blocks among the many made available by the program.
- - **The flow of instructions in *Scratch* was not particularly clear** to the kids, the event system did not allow to understand in which order the instructions were executed. For example, it was not clear which functions were blocking for the program and which ones allowed it to continue with other instructions while they were still executing.
- - The **multiple workspaces of Scratch caused confusion**; all the boys have made a mistake at least once due to the incorrect placement of a block.
- - **One of the biggest challenges has been spatial reference systems**, which use the pixel (called ‘step’ in *Scratch*) as a unit of measurement. This created a lot of confusion for the boys.
- - **Numerous blocks have required learning a lot of information** before they can be used consciously, for example the ones for managing variables, a simple

concept for insiders, but it can represent a first obstacle for those starting from nothing.

- The guys complained about the **lack of high-level blocks**, i.e., blocks with more elaborate functionality that aim to avoid having to add **too many blocks to the program**, for example there is no prefabricated way to be able to make a character easily jump, which must instead be programmed using a large number of blocks.
- **The semantics of some blocks has been made ambiguous** by the choice of words used to compose the text present on them (the language used for the experiment was Italian, but the original version also has similar defects), for example the blocks related to the concept of 'messages' made all the kids think they were texts to be printed on the screen rather than event handling.
- The boys made **frequent and constant requests for assistance** to know how to use the software and the semantics of almost all the blocks they use.
- *Scratch* allows you to connect the created script to a single object present in the scene, if you want to **create more objects with the same behavior** you must duplicate the script code or use the 'clones' mechanism provided by the software, which however **was not very intuitive**.
- **Code tends to get long and complex quickly** due to using many blocks, even when programming quite simple behaviors.

<u>Experience related feedbacks:</u>

- All group members found it **curious** and **fun** to put yourself in the shoes of a game developer.
- Most of the boys, even the most introverted ones, have found in the group experience a way to **express themselves** by confronting people with common interests.
- The group experience has made it possible to create bonds and **relationships** between the boys, some only within the group meetings, while others have also been exported outside of it.

4. Design Principles

So, what is the best set of choices that can be made to design a tool that can reach our initial goal?

We need to reach people who are approaching coding and game development for the first time, people who have no experience about machine technical specifications or operating systems, so they can own different technical environment setups.

4.1 Platforms

A **web-based** application here is probably the best choice. It does not require any installation, so it is suitable for several contexts such as teaching, challenges, or exploring the technology, situations in which the user want the tool available in the shortest amount of time. Another advantage of this approach is that in this way the tool will be platform independent, allowing it to run on almost every operating system and device. However, a web-based tool is less performant than a native one, but this should not be a big issue since we still want users to create easy games compared to the big industry AAA games they are used to. Another limitation of this approach is that that user must have an internet connection to use the tool, but nowadays this is not a big problem anymore, however a solution could be making available an offline version of the software, which is a quite simple task starting from a web code base.

4.2 Technologies

Games also represent a way to explore new technologies, so the idea is to create a playground for our user to explore different environments and possibilities.

Most of the games our target is used to make use of **3D graphics**, and the whole sample chosen for the experimentation state that they prefer this kind of graphics approach over the 2D one. Also, 3D graphics is less present in the examined existing tools, which usually prefer to support 2D only. Having said that, we can easily choose 3D as our main target graphics technology.

Another technology our target (and most of the people) is used to is **mobile platforms**. So, the created game, and the editor, should also be available on these devices. One of the biggest problems of this kind of platforms is that making an application available for them is a non-trivial task due to several SDKs that differs for OS and version and to the slow compilation process to have the user generated game installed. However, choosing a web-based approach allow us to make this process much simpler, the users can have their games running on several mobile devices with the same simplicity they can **share a web URL**, but we will discuss about this later when talking about the software architecture.

If we have gamers included in our target, then they are surely familiar with **gamepads.** Very few examined tools support them natively, so it can be a nice add to make people explore unusual ways to send input to their game.

A modern technology that we are seen more often these years is **VR** and **AR**. These technologies are intriguing people and making them available on our platform can add an enormous power to the playground effect we want to create. To be effective they should be available both on mobile devices and headsets because the latter are not yet widespread.

4.3 Assets Management

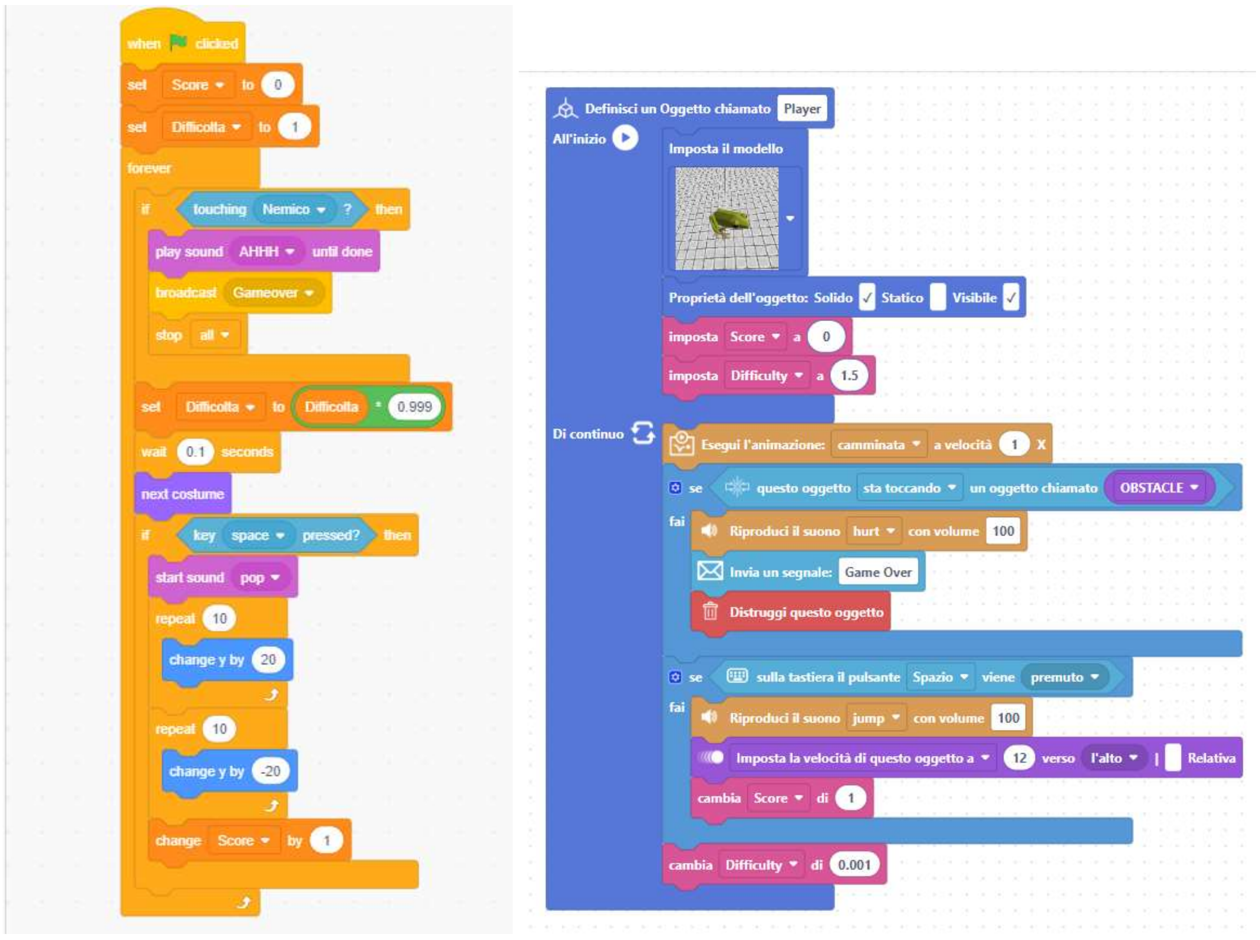
This is one of the most complex parts to speak about, so complex that I have chosen to make it as simple as possible: Allowing the user to only choose assets from **pre-constructed libraries**. We have already spoken about issues related to the assets importing process and evaluating the sample of target audience it turned out that the simply does not care about them so much if the pre-constructed libraries are filled with enough variety. In future developments sounds and maybe images editor/importer can be implemented, but speaking about 3D models, the problem is that the audience must be able to understand them, so we must teach people about 3D file formats, 3D animations and polygon and vertex count, because many of them do not know anything about this technology, so creating and maintaining an importer for several file formats it is not worth it at this stage.

4.4 Programming System

As we have said before, we want to avoid the use of traditional text-based programming languages, instead we want to make use of **VPL**, because they are easier to learn and manage for the users and they are completely syntax error free.

The most diffused approach is the **block-based** one. It is simple, clean, and similar to traditional programming languages. So, I have chosen to adopt it for the project.

One of the biggest issues with VPLs is that the programming environment get messed up soon compared to traditional programming languages. The solution I found to this was to make unused blocks automatically collapsing and rearranging in a clean way. This turned out to be effective during the experimentations, and the code workspace management was never a problem.



```
when clicked
set Score to 0
set Difficolta to 1
forever
    if touching Nemico ? then
        play sound AHHH until done
        broadcast Gameover
        stop all
    set Difficolta to Difficolta * 0.999
    wait 0.1 seconds
    next costume
    if key space pressed? then
        start sound pop
        repeat 10
            change y by 20
        repeat 10
            change y by -20
        change Score by 1
```

```
Definisci un Oggetto chiamato Player
All'inizio
    Imposta il modello
    Proprietà dell'oggetto: Solido ✓ Statico Visibile ✓
    imposta Score a 0
    imposta Difficulty a 1.5
Di continuo
    Esegui l'animazione: camminata a velocità 1 X
    se questo oggetto sta toccando un oggetto chiamato OBSTACLE
    fai
        Riproduci il suono hurt con volume 100
        Invia un segnale: Game Over
        Distruggi questo oggetto
    se sulla tastiera il pulsante Spazio viene premuto
    fai
        Riproduci il suono jump con volume 100
        Imposta la velocità di questo oggetto a 12 verso l'alto | Relativa
        cambia Score di 1
    cambia Difficulty di 0.001
```

Figure 8: Scratch vs Fairytool code for the Chrome Dino Game main character

4.5 Space Handling

As we have seen before, one of the most important difficulties of our target audience is represented by the reference systems, people find it hard to understand even using a 2D space. The main causes of this are the pixel as unit of measurement and the axes orientation that may not be clear from the beginning. To mitigate this, I have decided to adopt a **grid-based** solution both for the scene editing system and the space handling by the VPL. It is clear and it makes easier the placement of the object into the scene, furthermore the axes orientation should be visible into the environment both for the scene and the objects the user is working on.

Finally, the gizmo design should be extremely simplified compared to professional software. The gizmo is a GUI control system for scene's object transformations, including translation, rotation, and scaling. The gizmo appears on the selected object and shows the user arrows and other graphical icons to control the object's geometric transformations along each axis. Usually, its appearance changes depending on the transformation selected by the user, so it looks different when the user selects for example to change the rotation instead of the translation of the object. Gizmos are usually complete, but the process of changing transformation can be cumbersome, so I decided to be inspired by the one used in ***Struckd***, a popular mobile tool that allows users to create and share simple games. The idea here is to not change the Gizmo appearance and make available only Z axis translation, proportional scaling (no scaling available for each axis), and rotation around two axes instead of three. This works because you can move the object on the XY plane just by dragging it, so there is no need to having this translation on the gizmo, and the rotation along the third axis can be created composing the rotation along the other two axes, which are usually the most common ones, so this operation is often not needed.

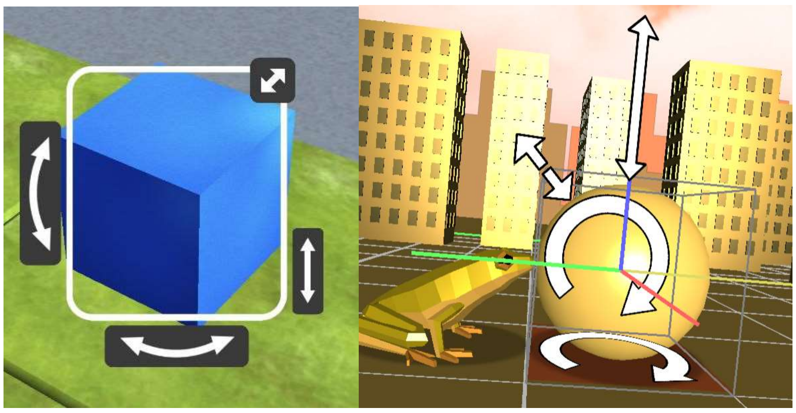


Figure 9: Struckd vs Fairytool Simplified Gizmo

5. Graphical User Interface

This is most critical component of our software, we need to make it complete and intuitive, keeping in mind that our users could be really young. The images related to the GUI will be shown in Italian, the language in which the experiment was conducted.

5.1 Login

The *login page* is the first page you see when you log into the software, it has a single button that allows the user to access *Fairytool* using any valid *Google* account. Once you click on the button a popup will open with a login wizard, which once done will be temporarily stored, avoiding having to log in every time you open the software.

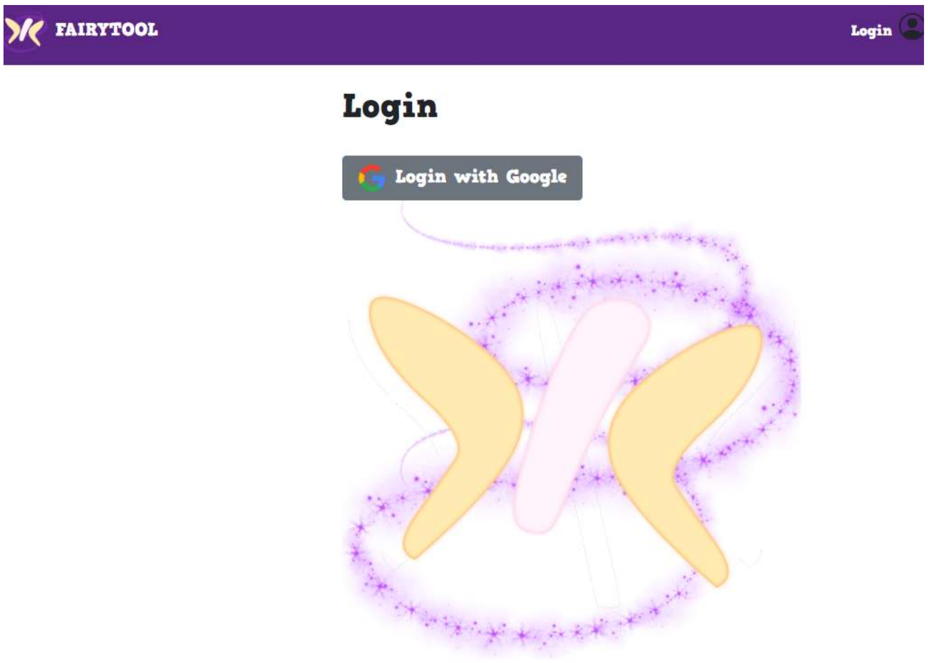


Figure 10: Fairytool login page

5.2 User Games

After logging in (or if a valid recent login is found) you are redirected to the user's personal page. Here you can manage your games, especially you can **play** a game, **delete** it, **duplicate** it, **edit** it or **create a new one**. When you delete a game, a confirmation pop-up appears on the screen to **prevent accidental deletions**.

When you click on the button to create a new game, a popup is shown with the purpose of letting the user choose whether he wants to start from an empty project or from a **template**. The latter are pre-built projects that allow the user to explore and analyze simple video games that vary by genre and technology

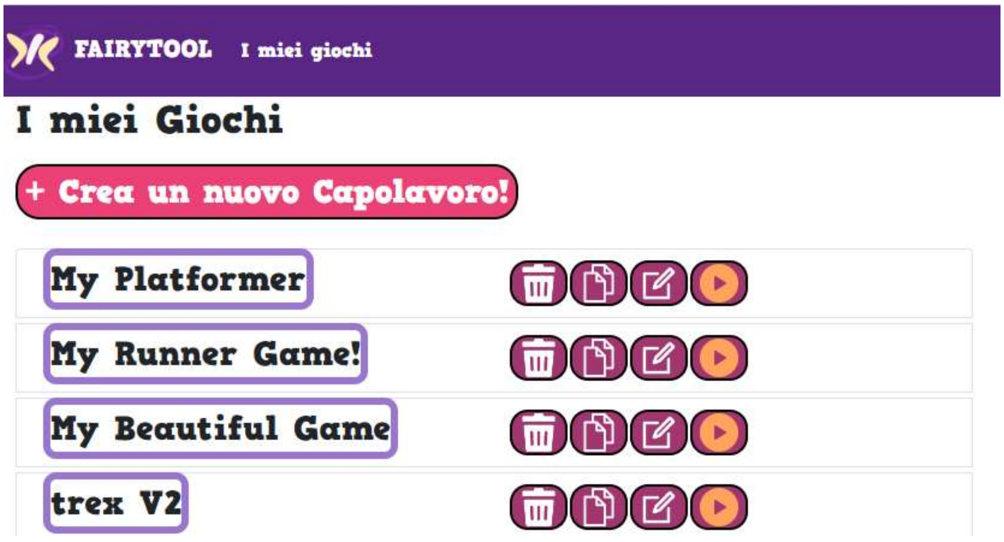


Figure 11: Fairytool User Games page

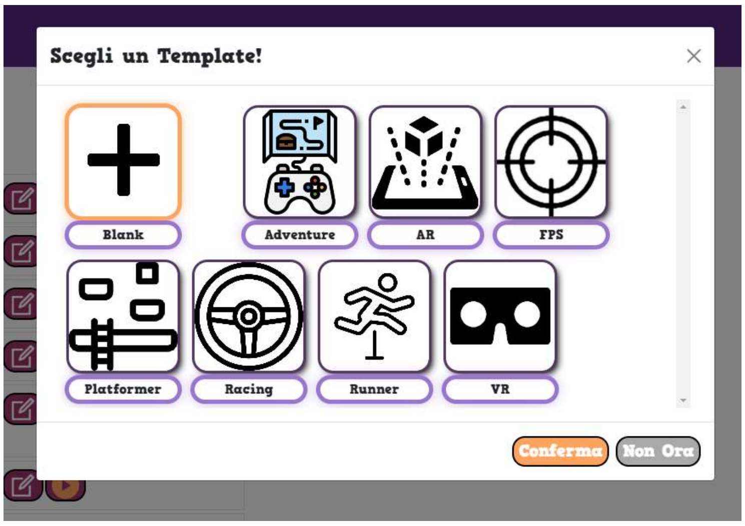
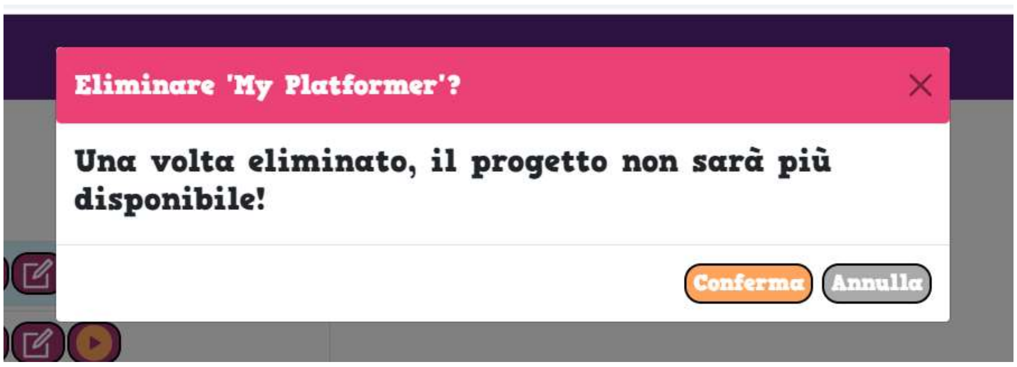


Figure 12: Template Chooser


Eliminare 'My Platformer'?

Una volta eliminato, il progetto non sarà più disponibile!

Figure 13: Delete confirmation dialog

5.3 Game Editor Header

The page header allows the user to perform various actions.

First, you can go back to your personal page, the one that contains the list of your projects, or even to the login screen.

At the top right you can view the **account** with which you are currently logged in and, by clicking on the drop-down menu next to the email used by the account, you can **log out** of the system.

When a game is being edited, the header also shows a button to be able to **play the game** and run it in another browser tab, for each other change made to the game it you can just reload this tab to see the changes applied or press again the button to start the game.

One last feature of the header is to have a ***help* button** to launch a quick guide that will explain the basics of using the software. This quick start guide was created using the *Tango* online tool.

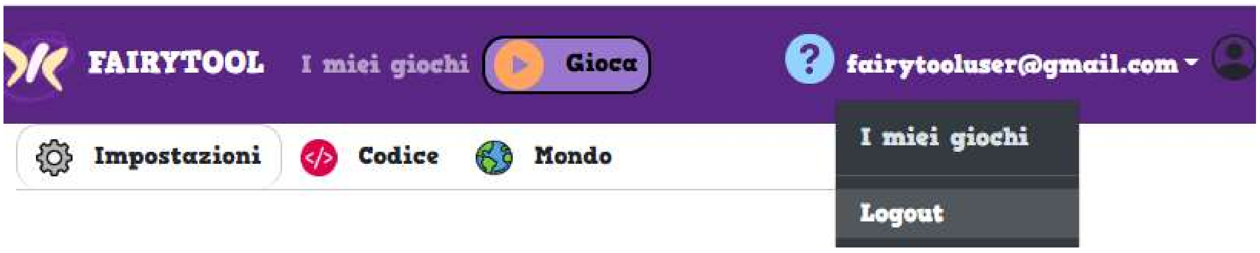
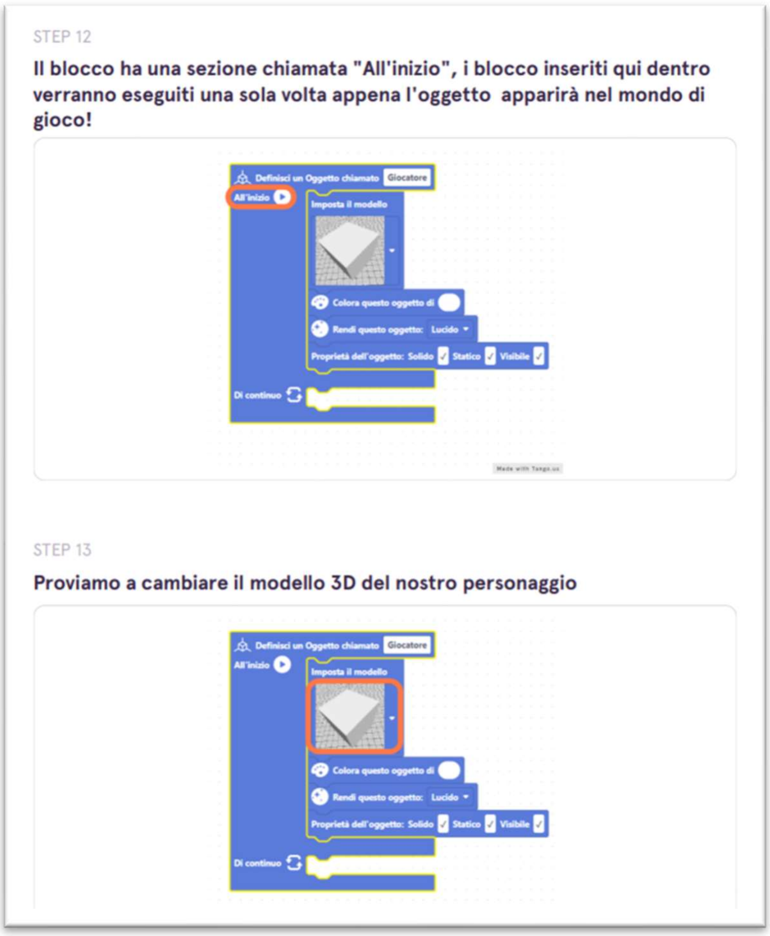


Figure 14: Fairytool Game Editor Header


STEP 12

Il blocco ha una sezione chiamata "All'inizio", i blocco inseriti qui dentro verranno eseguiti una sola volta appena l'oggetto apparirà nel mondo di gioco!

STEP 13

Proviamo a cambiare il modello 3D del nostro personaggio

Definisci un Oggetto chiamato
Giocatore
All'inizio
Imposta il modello
Colora questo oggetto di
Rendi questo oggetto: Lucido
Proprietà dell'oggetto: Solido Statico Visibile
Di continuo
Made with Tango.us

Figure 15: Quick Start Guide

5.4 Game Editor

Clicking a game's edit button or creating a new one takes you to the game editor, the hub of the user experience. Here every change you make to the games will be **automatically saved** on the server so it will therefore be accessible from any device you log into with the same account.

This section is split into three different tabs **Settings**, **Code** and **World**.

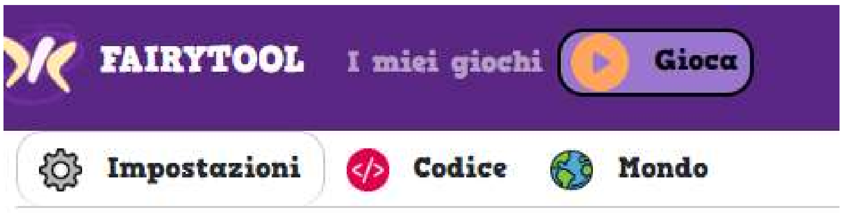
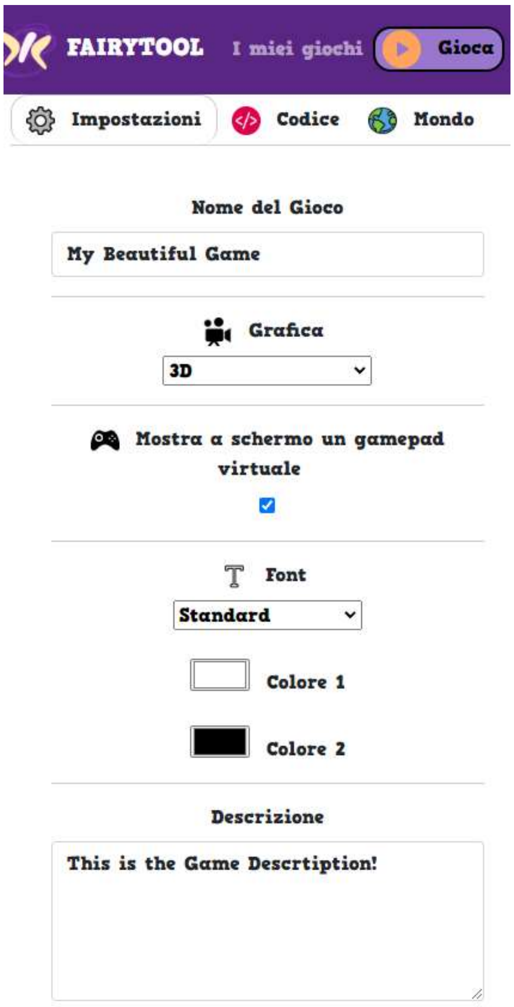


Figure 16: Fairytool game editor tabs

5.4.1 Game Settings

Clicking on ‘Settings’ tab of the game editor the user can change the following general game settings:

- The name of the game
- The graphics engine, choosing between classic *3D graphics*, *Virtual Reality* or *Augmented Reality*, the latter will allow the execution of the game only if the device currently in use is compatible with these technologies.
- The on-screen display of a virtual Gamepad, useful when creating a game with many controls for touchscreen devices such as smartphones or tablets.
- The font that will be used when running the game.
- A simple color scheme made by two colors that will be used for some default elements of the game, such as the game main menu or some elements of the eventual HUD.
- A description of the game being made.



Nome del Gioco

My Beautiful Game

Grafica

3D

Mostra a schermo un gamepad virtuale

☑

Font

Standard

⬜ Colore 1

⬛ Colore 2

Descrizione

This is the Game Descrtiption!

Figure 17: Game Settings Tab

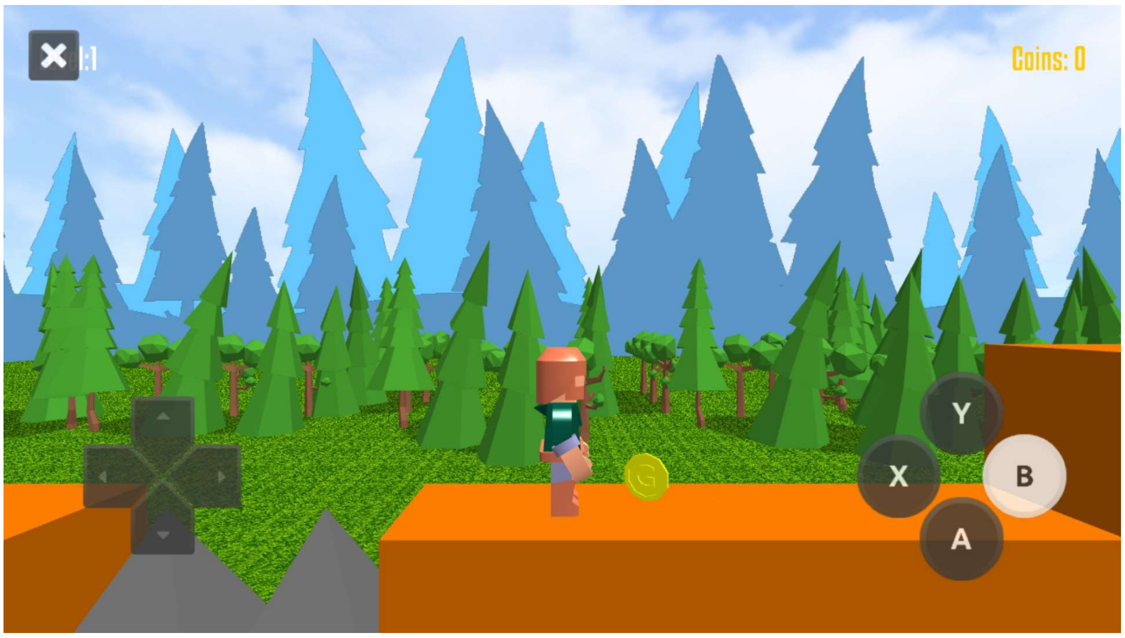


Figure 18: Virtual Gamepad used in a game

5.4.2 Game Programming

The ‘Code’ tab of the game editor allows the user to program the behavior of objects in the game. It consists of a **single code workspace** and an **object outliner** on the right where any programmed object will automatically appear. By clicking on the outliner object icon, you will be redirected to the corresponding piece of code in the code workspace.

Within the code workspace there can be only two types of blocks, **objects,** and **functions**.

- **Objects** allow you to create **templates that you can use multiple times** within your game, a mechanism similar to what happens in real programming. this contrasts with what happens in most of the existing tools as almost all of them allow you to program a single object in the scene and not a class of objects present in the scene. This is done to avoid having to teach kids the concept of scope, but on the other hand it makes it extremely difficult to create even quite simple scenes where there are frequently repeated objects (just think of the obstacles of a platform game).
- **Functions** are programming blocks for creating logic that can be reused multiple times within the program without having to copy and paste the code. They are quite an advanced concept for those who want to try their hand at improving the organization of their work, but they are not a vital feature for the software.

To reduce the spatial complexity generated by block placement, blocks you are not working on will **automatically collapse** leaving only the essential information to identify the block visible (icon and name), moreover blocks are **automatically rearranged** to further optimize the space from them busy. However, each block can be collapsed manually by **right-clicking it to open the context menu** and selecting the corresponding option.

We will take a deeper look at the block programming system in the *Visual Programming language* chapter.

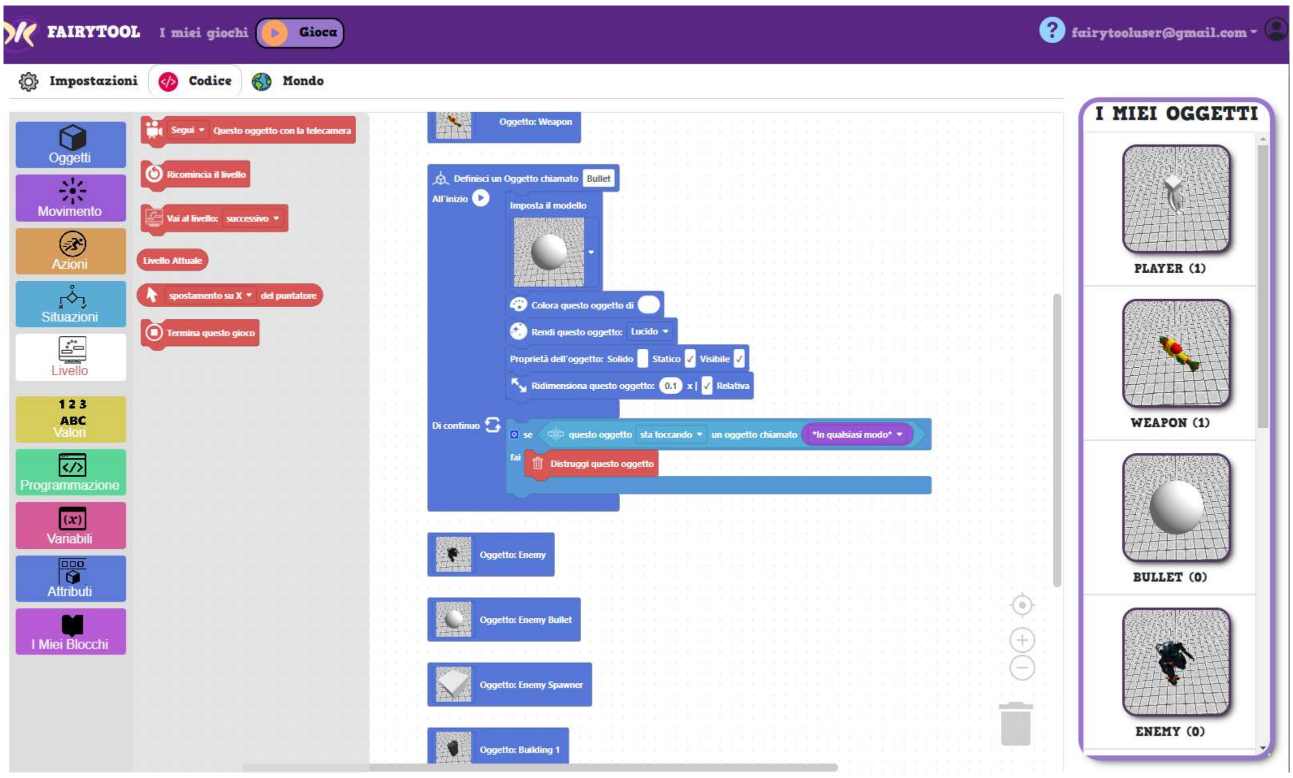


Figure 19: Fairytool Code Tab

5.4.3 Scene Management

By clicking on the 'World' tab of the game editor, you will be taken to the game scene edit screen.

The screen is made up of **four macro-sections**:

- **<u>The scene properties</u>** are settings relating to the single scene being edited. These settings include:
    - **The camera view**, i.e. a drop-down menu that allows you to choose how the scene will be displayed, whether with a view orthogonal to one of the axes, or if in first or third person, the latter automatically also include control of the camera system via input, so you won't need to program it in code. This option is only available when the game's graphics engine is set to '3D', otherwise it will only be viewed from the user's point of view in the real world.
    - **The distance of the camera** from the point or subject framed, this is not available when the view is in first person, or the graphics engine is set to a value other than '3D' because in that case the distance from the subject is always 0 being the framing based precisely on his point of view.

    - **Gravity**, which will affect the physics of that level, in future developments, this feature could be implemented as a programming block, to allow for dynamic gravity changes.
    - **The lighting**, which will modify both the global illumination of the scene, adding both an ambient and a directional light, and the skybox, or the background designed to simulate the presence of a sky in the scene, adapting it to the selected lighting, for example if we choose the 'Sunset' option, the light in the scene would tend to turn orange, it would become slightly less inclined with respect to the ground, and the skybox would adapt, now representing a sunset sky.
    - **The landscape** represents the background visible in the distance, it serves to make the scene more coherent with what you want to represent, for example you can add a city seen in the distance, mountains, a forest, etc.
    - **The terrain** allows you to choose the texture of the base plane of the currently edited scene, to further customize the scene, for example we can make it look like an expanse of grass, earth, sand, etc. scroll the mouse wheel
    - **The background color** allows you to change the color of the texture applied to the predefined support plane.
    - **Music** is the background music to the scene, when you press the button to choose music, it opens a popup where you can navigate to choose the song you want.
    - **Music volume**, this feature together with the previous one could be integrated into the programming blocks during future developments to allow dynamic music programming.

- <u>**The scene editor**</u> allows the user to change the composition of objects within a single scene. Here the inserted objects can be moved along all axes, resized, rotated, or deleted.
  In the scene, the global axes are visible as follows: **X-axis colored red, Y-axis colored green, and Z-axis colored blue**. These axes serve as a reference to check the rotation and position of the various objects in the scene immediately.
  There is also a grid in the scene that indicates the height level you are working at. By dragging the objects on this grid, it will be possible to **translate** them on the plane parallel to the ground (keeping them **hooked to the grid**) while maintaining their position with respect to it.

Clicking with the left mouse button on an object will select it, moving the grid to the height of the base of that object, and causing a simplified gizmo to appear on it. This **gizmo** will allow you to **rotate** the object over two axes (the rotation on the third axis can be obtained by combining the rotations with respect to the other two), to **resize** it keeping its proportions unchanged, and **to translate it over the Z axis.** This last operation will also move the grid that represents the work plane together with the object, to allow you to easily insert other elements at the same height. The last function of the gizmo is to show the **relative axes** of the object and the **direction of the face** of the same (drawn in yellow), to be able to immediately identify its orientation with respect to the scene or with respect to other objects.
By holding down the CTRL key it will be possible to make a **multiple selection**. It is possible to **delete** an object by dragging it to the **trash icon** or by clicking on it when the objects we want to delete are selected. However, all the traditional **keyboard shortcuts** are available for copying, pasting, and deleting items from the scene.
To navigate within the scene, I tried to minimize the commands to be used, which in any case always remain visible in the lower part of the screen. **The rotation of the camera is done by pressing the right mouse button** followed by a mouse drag action, as happens in many other 3D editors, but as during the experiments it was discovered that most of the kids were not used to clicking with the right mouse button mouse button mouse**, the same operation can be performed with the left button** if we are not pointing an object with the mouse, otherwise the object will be selected and dragged. **To translate the camera in the scene you have just to press the middle mouse button** (the one under the wheel) and drag, while **scrolling the mouse wheel will zoom it**.
At the top left there are three **buttons that will allow the camera to position itself orthogonally to the axes** automatically, without having to manually position the camera.
Finally, in the highest part of the scene editor, we find some useful information, such as the name of the selected object, and the coordinates of the last clicked grid cell**, these coordinates will be colored in the same way as the respective axes** to improve the ability to orientate the user and be able to immediately identify the axes. The cell in question will also be colored red within the scene.

- **The scene outliner** allows you to select up to ten scenes to work on. Scenes with objects in them are considered not empty and will display differently from others. The outliner will also highlight the scene on which you are currently working.
- **The object outliner,** here you can see the icons and names of all the objects defined within the 'Code' tab. Dragging objects from this outline to the scene, you will be able to add them to it at the position you release the mouse.

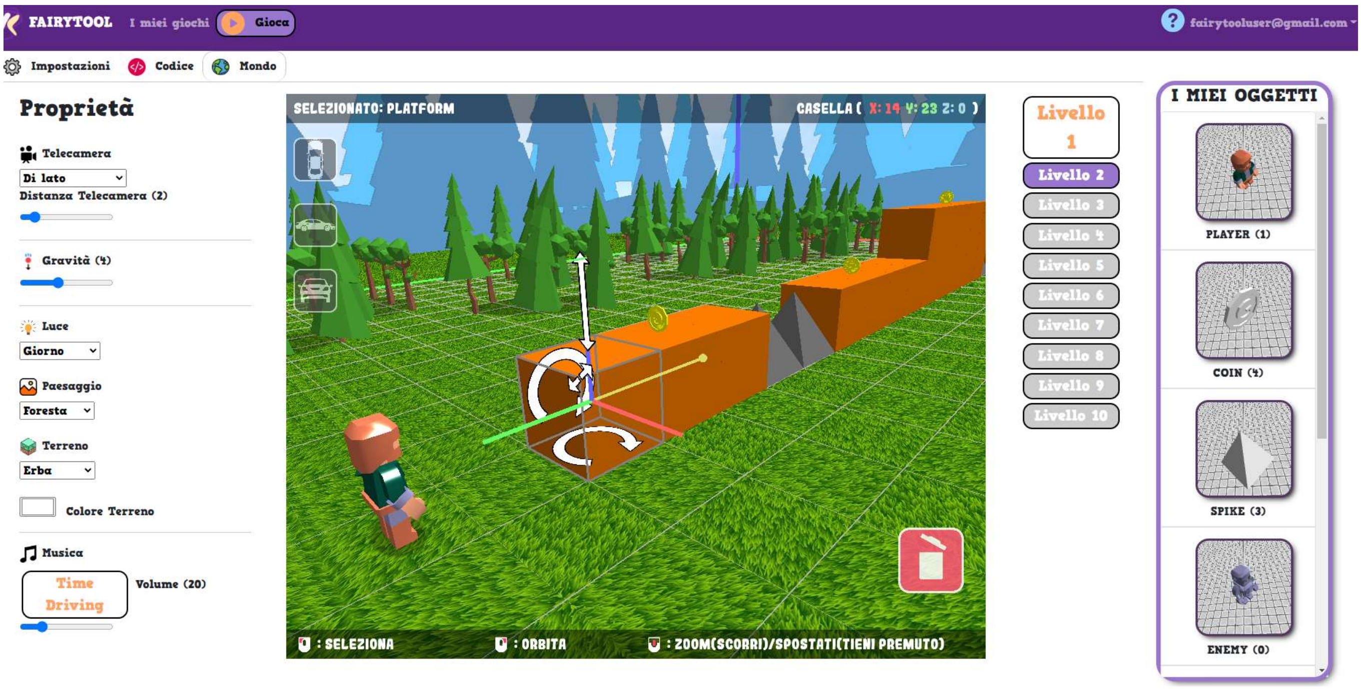


Figure 20: Fairytool Scene editor

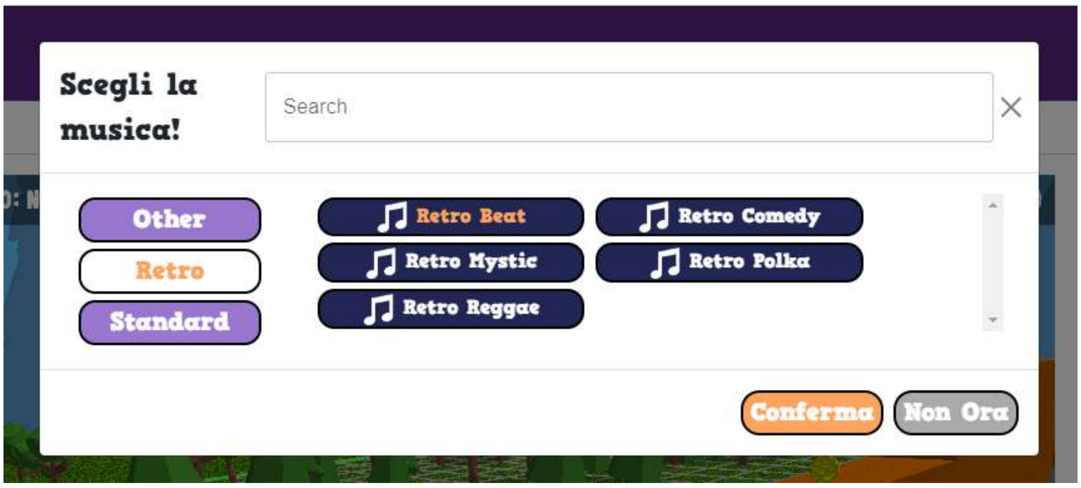


Figure 21: Background Music Chooser

5.5 Running Game

Once the user has pressed the ‘Play’ button, a new browser tab will be opened and here the game will be displayed in full screen mode. By reloading this page, you can update the game view with the latest changes made by the creator.

If the game was created by setting the graphics to ‘Virtual Reality’, the VR viewer used will take the user into the virtual game environment, after clicking on the appropriate button shown on the screen. If, on the other hand, the game was created using the 'Augmented Reality' graphics option, the camera of the compatible device will open, and it will be necessary to touch the surface of the real world on which we want to place our virtual world.

The start menu will be standard for all games, displaying the game name with a simple animation and the option to start the game, all using the color scheme chosen in the game editor's ‘Settings’ tab.

At any point in the game, pressing the ESC key will take you back to the main menu.

It is possible to simply share the URL of the playing game page to make it available on any other device connected to the internet. The game will run even if the user is not logged in *Fairytool,* this I the fastest way to share the created games with friends.

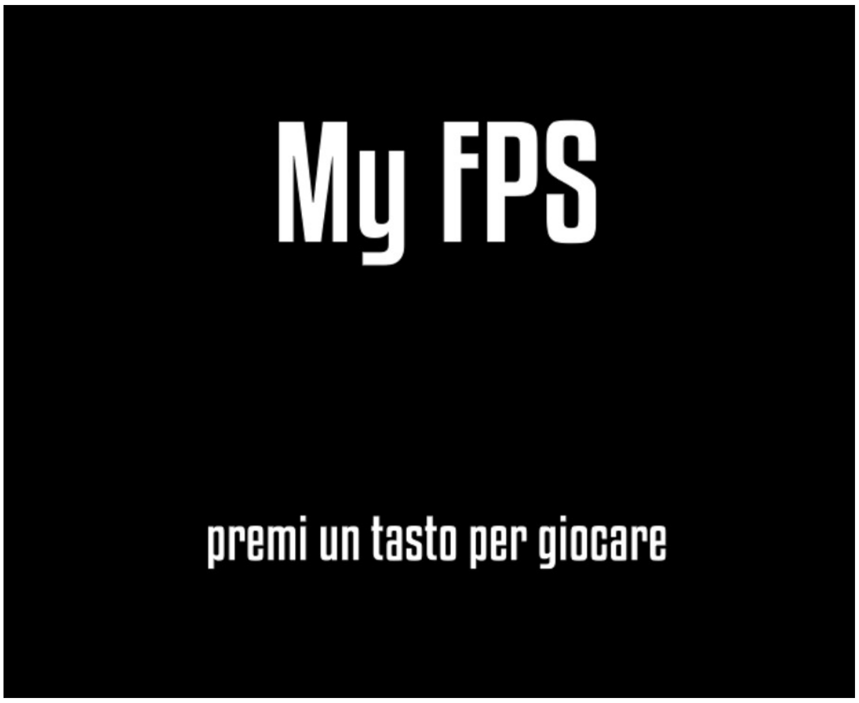


Figure 22: Game Title Screen

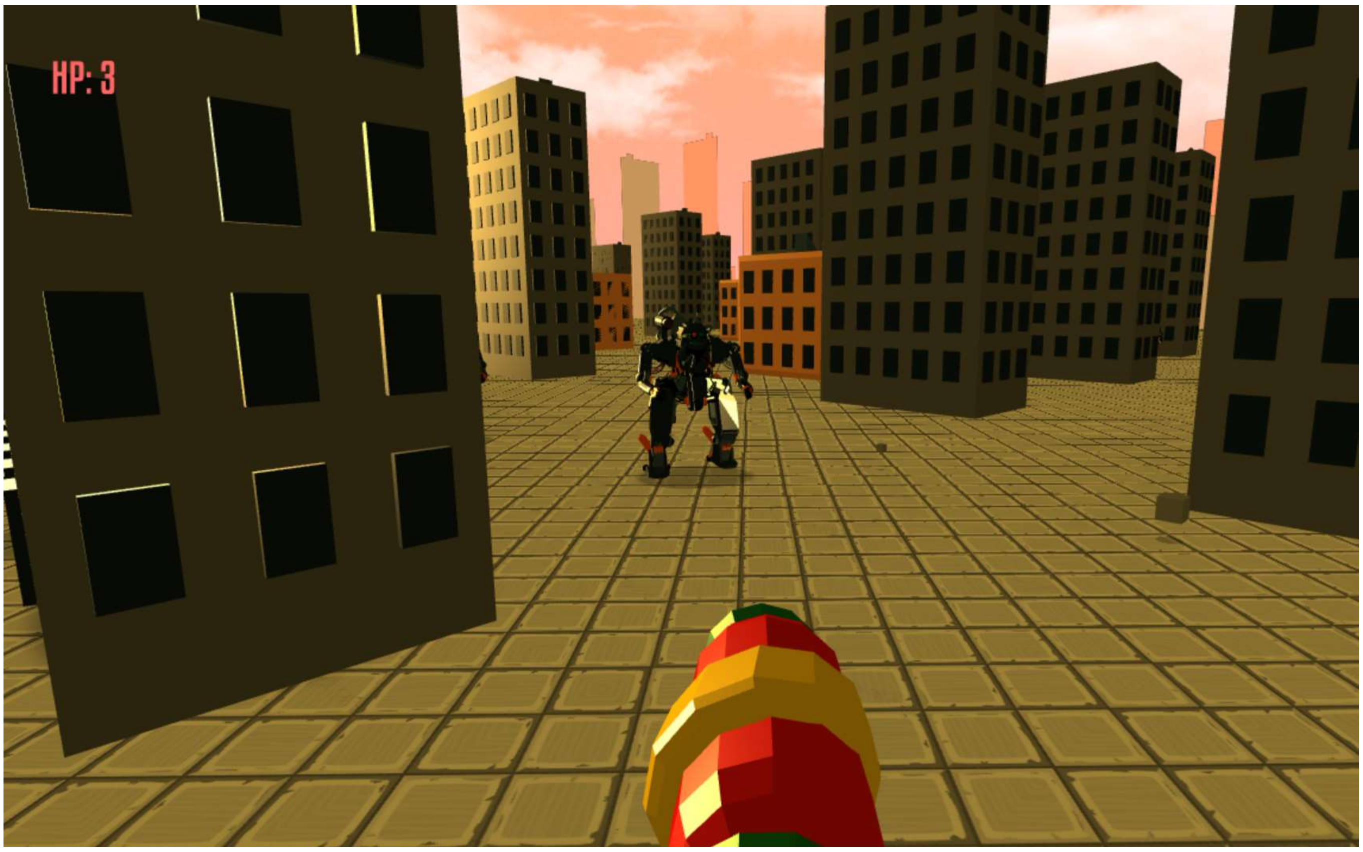


Figure 23: Example of Game made using Fairytool

6. Visual Programming Language

As mentioned above, one of the major problems to be solved is knowing how to manage the quantity of blocks made available to the user, which should not be too high to confuse him, nor too small to make the language less expressive. Also, it should be easy to identify the blocks and figure out how to fit them into each other.

6.1 The language basics

This VPL, as mentioned above, uses draggable **blocks** as a major part of the development process. These blocks are organized into various visible **categories** on the left side of the code workspace in a component called **toolbox**, each category has an **icon**, **name**, and **color**. Clicking on the desired category will open a drop-down menu which will allow you to choose a set of blocks. Each **block is characterized** by a **color**, an **icon** (optional), the **content of the block** (consisting of text and any options), but above all by its **shape**. The shape of the blocks **says where they can be inserted**, and it is only possible to insert blocks that are compatible with the hook, this **avoids any syntax or type errors**.

Variables and **attributes** can be defined and used as parameters for almost all the blocks present in the programming environment to add more dynamism to the game.

Unlike traditional programming languages, user-defined **identifiers** within the code are **case insensitive**, and **spaces are allowed** within identifiers, for example *"Box Count"* is a valid name.

Blocks can be moved using a **drag-and-drop** mechanism, moving a block will also move all blocks that will be stacked below it, to move a single block just hold down the CTRL key.

Furthermore, when blocks are right clicked, they show a **context menu** that allows you to perform various operations on the block itself, such as **collapsing** it, **duplicating** it, adding a **comment,** show documentation, or **deleting** it. A block can also be deleted by dragging it onto the **trash icon**, which, if consulted, can return all the elements present within it to be able to restore them.

At the highest level only two types of blocks, **objects** and **functions** are considered valid, all other blocks will be nested in them, otherwise the software will make them semi-transparent and not calculate them when running the game.

Also, higher level blocks will **automatically collapse** when not in use, and automatically rearrange within the space to optimize it and make it less chaotic.

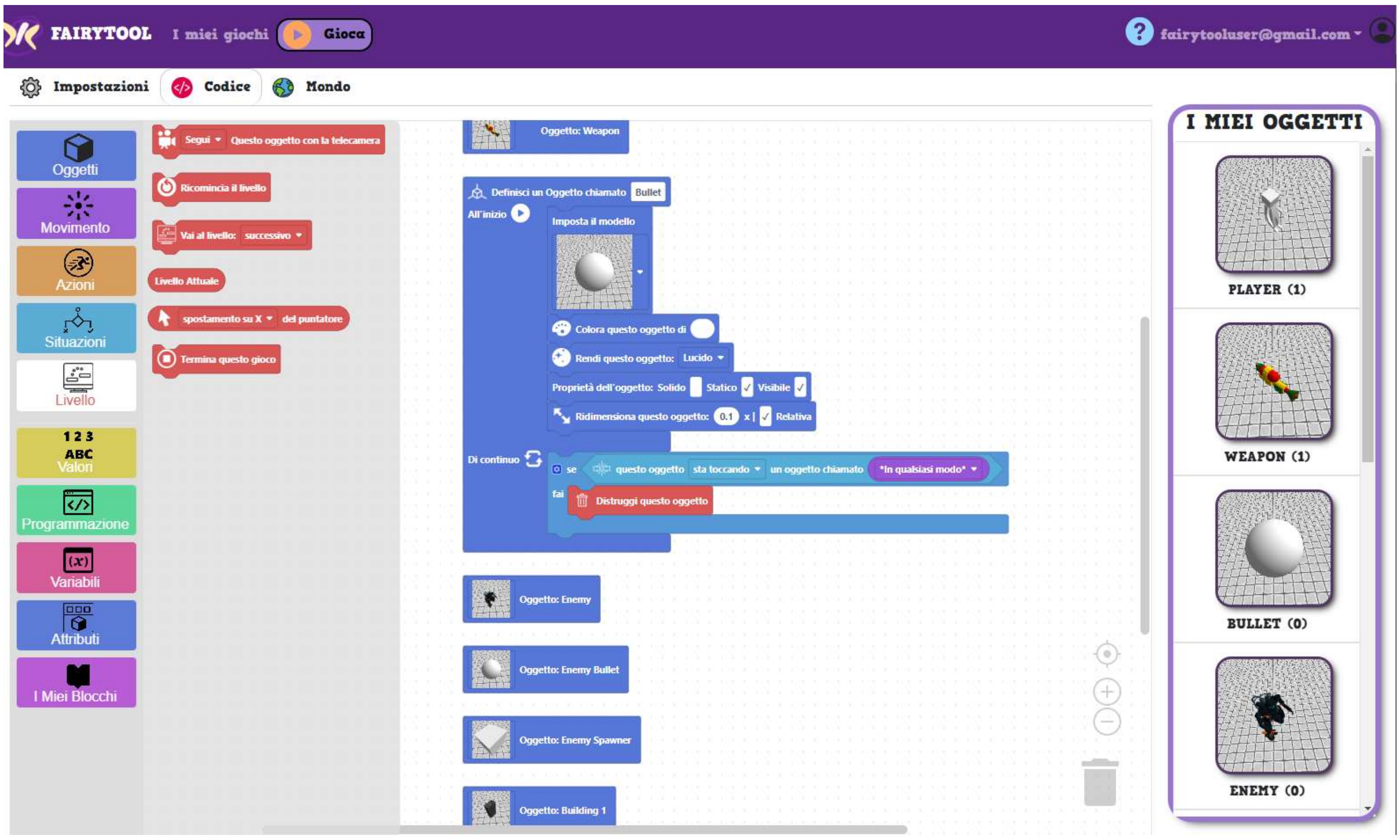


Figure 24: Fairytool Code Editor

6.2 The Toolbox

Let us now analyze in detail all the categories of blocks present in the toolbox and how they are organized inside it.

6.2.1 Objects Category Blocks

This category of blocks contains instructions for editing everything related to the definition of objects.

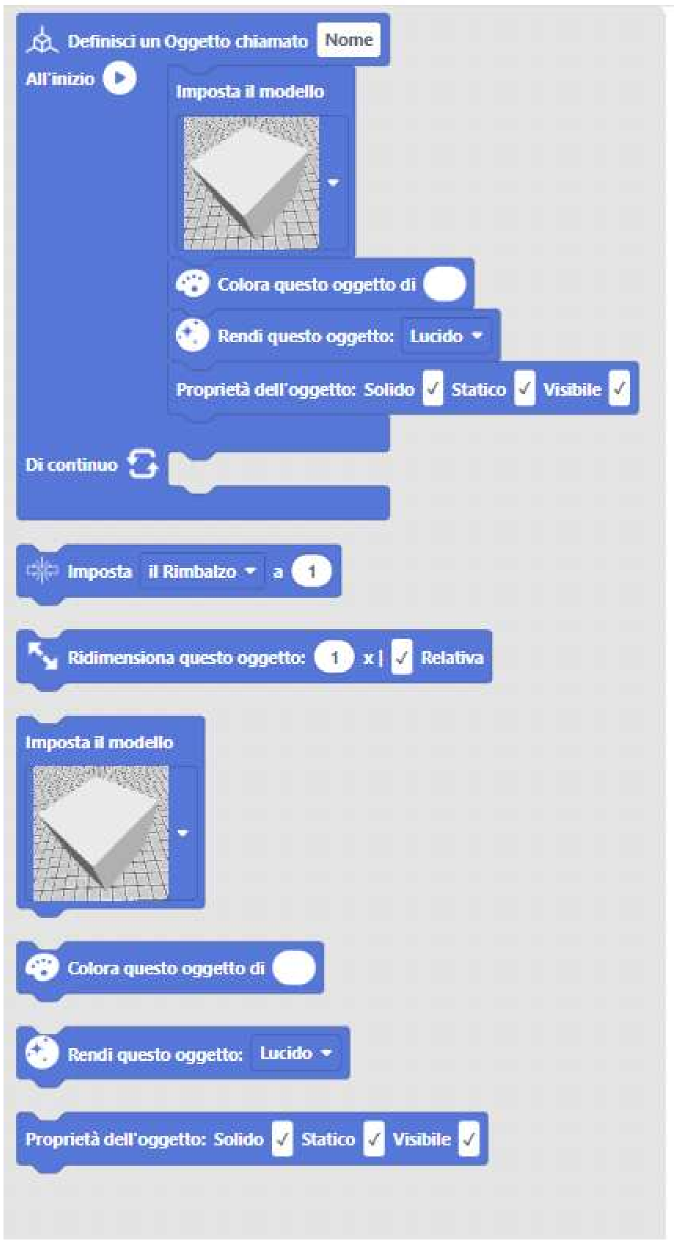

Definisci un Oggetto chiamato Nome
All'inizio
Imposta il modello
Colora questo oggetto di
Rendi questo oggetto: Lucido
Proprietà dell'oggetto: Solido ✓ Statico ✓ Visibile ✓
Di continuo
Imposta il Rimbalzo a 1
Ridimensiona questo oggetto: 1 x ✓ Relativa
Imposta il modello
Colora questo oggetto di
Rendi questo oggetto: Lucido
Proprietà dell'oggetto: Solido ✓ Statico ✓ Visibile ✓

Figure 25: Objects Category

1. It defines an object with the name set in the appropriate field, this name must be unique, and if it is not, the software will make it so. All objects instantiated with this definition will bear the same name. But they will still be distinct if present within the same scene.
   This is the most important block of all, in fact, the definition of this object will appear instantly in the objects outliner, making it available for insertion within the various scenes. It has two **statements**, or two slots where you can insert other blocks, **one to insert instructions that will be executed only once when the object enters the scene**, and **the other to insert instructions to be executed at each update cycle of the game** (about sixty times per second). The block is displayed within the toolbox with already commonly used blocks in it.
2. This block allows you to set elementary physical properties for an object, that is, you can set the bounce, the mass, or the damping (a sort of friction that allows you to slow down the movements of the object).
3. Resizes the object by scaling it by the specified factor. The 'relative' field refers to the original size of the object. For example, if enabled, an object whose size has been halved, scaling 0.5 will halve it further, rather than leaving it unchanged.
4. This block shows a preview of the 3D model to be assigned to the object and if clicked will open a popup that will allow you to choose a 3D model from a library by selecting one of the various categories, or by performing a keyword search.
5. Allows you to change the display color of the object.
6. Changes the material of the object to the selected one, that is, changes the way it reflects or emits light. If it emits it, then it will emit it in the color of the object.
7. Allows you to change the basic properties of the object. If an object is solid, then it cannot pass through other solid objects, if an object is static, then it will not be affected by forces (e.g. collisions response or gravity) but it can still be moved using static velocities or be solid and thus provide a point of contact for other objects, If the object is visible, then it will also be displayed within the scene.

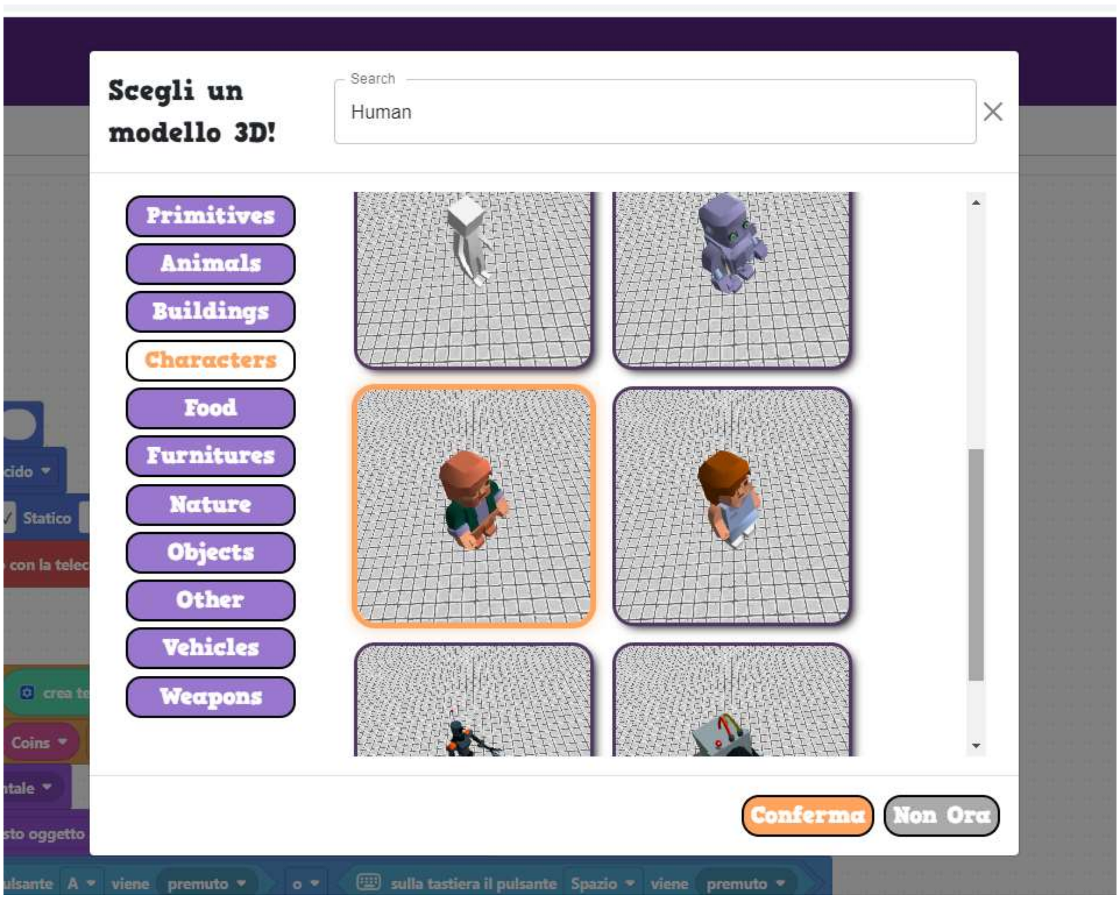


Figure 26: 3D Model Chooser

6.2.2 Movement Category Blocks

This category contains all the blocks for repositioning, moving, or rotating objects.

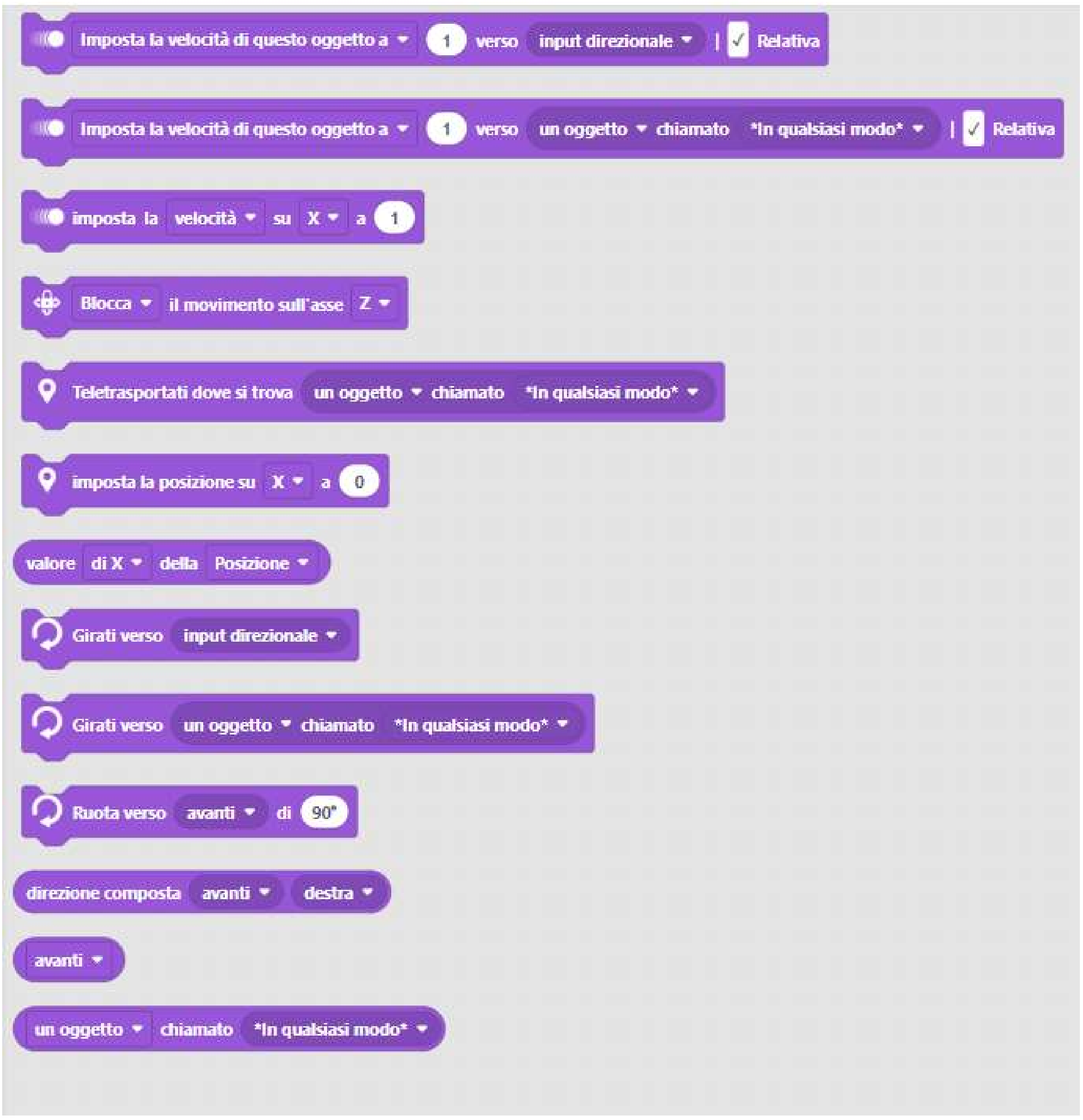

Imposta la velocità di questo oggetto a 1 verso input direzionale | ✓ Relativa
Imposta la velocità di questo oggetto a 1 verso un oggetto chiamato *In qualsiasi modo* | ✓ Relativa
imposta la velocità su X a 1
Blocca il movimento sull'asse Z
Teletrasportati dove si trova un oggetto chiamato *In qualsiasi modo*
imposta la posizione su X a 0
valore di X della Posizione
Girati verso input direzionale
Girati verso un oggetto chiamato *In qualsiasi modo*
Ruota verso avanti di 90°
direzione composta avanti destra
avanti
un oggetto chiamato *In qualsiasi modo*

Figure 27: Movement Category

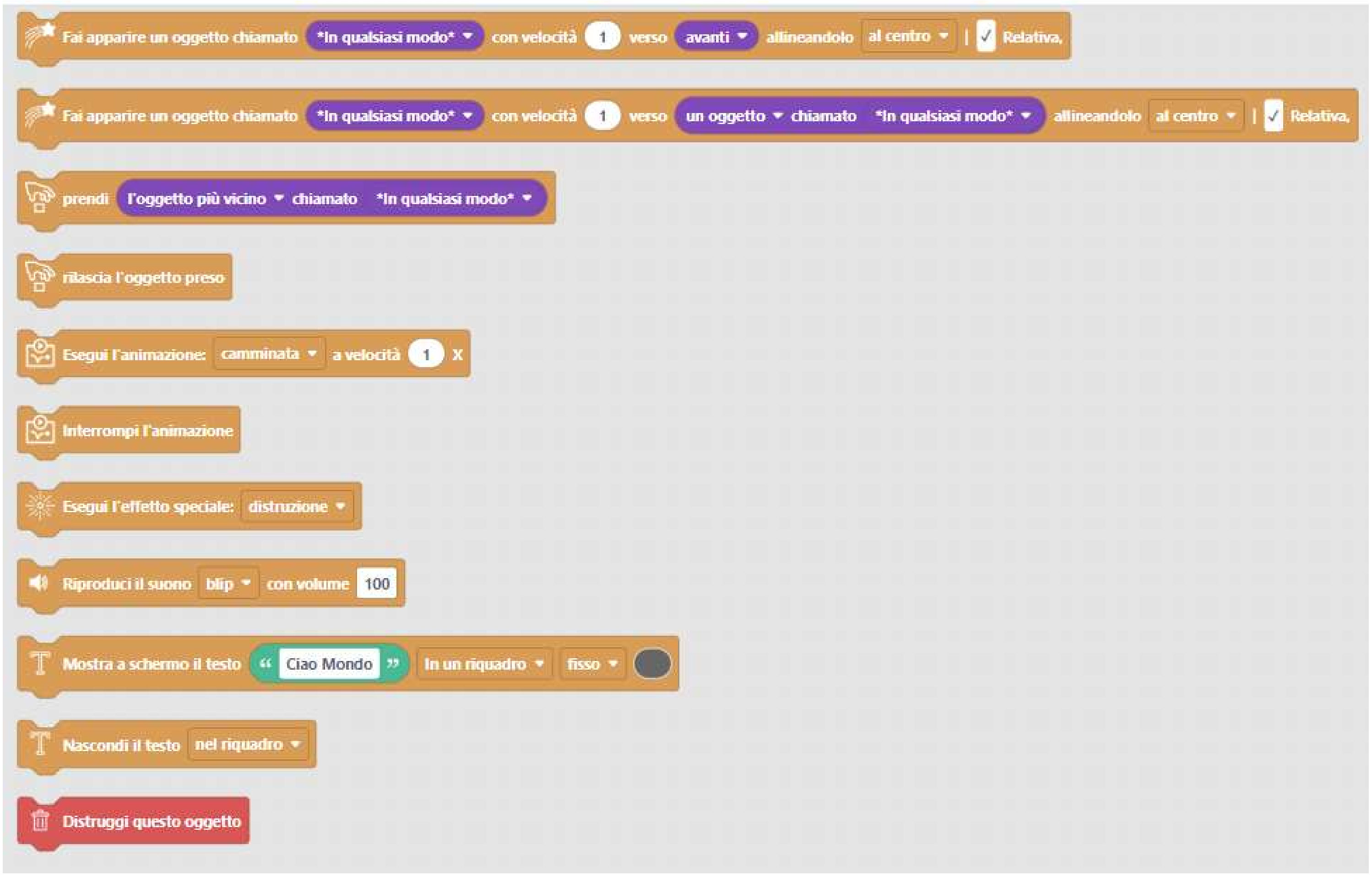
1. Allows you to set the speed or force with which an object should move in the specified direction. The 'Relative' option allows you to specify whether this direction refers to the scene, or to the current rotation of the object, for

example if it is active, the object is rotated to the right, the ‘forward’ direction will point to the right.

2. Same as the previous block, but the direction is defined by the position of another object in the scene.
3. It allows you to set the speed of the object using the individual axes, designed for users who need more control over movements.
4. It allows you to lock the movement on a certain axis, for example if I lock the movement on the Z axis and tell the object to follow another, this will only move on the XY plane.
5. Allows you to place the object at the same position as another.
6. Allows you to manually set the position of an object, using the grid cells as the unit of measurement.
7. Returns the value along a certain axis of the position, velocity, or rotation of the object
8. Causes the object to look in the specified direction.
9. It causes the object to look towards the position of another object.
10. Rotates the objects towards a certain direction by the specified angle.
11. Allows you to create a direction consisting of two directions.
12. It allows you to get a direction, if the direction has ‘input’ in the name, then it refers to the corresponding directional input read by any device capable of modifying it (arrow keys, WASD keys, analog gamepad stick or directional pad).
13. It allows you to get the reference of an object by constructing a simple query and it can be used to specify the direction to that object.

6.2.3 Actions Category Blocks

These blocks allow you to perform several types of actions. Like animating the 3d models, playing sounds, showing particles and more.


Fai apparire un oggetto chiamato *In qualsiasi modo* con velocità 1 verso avanti allineandolo al centro | ✓ Relativa,
Fai apparire un oggetto chiamato *In qualsiasi modo* con velocità 1 verso un oggetto chiamato *In qualsiasi modo* allineandolo al centro | ✓ Relativa,
prendi l'oggetto più vicino chiamato *In qualsiasi modo*
rilascia l'oggetto preso
Esegui l'animazione: camminata a velocità 1 X
Interrompi l'animazione
Esegui l'effetto speciale: distruzione
Riproduci il suono blip con volume 100
Mostra a schermo il testo " Ciao Mondo " In un riquadro fisso
Nascondi il testo nel riquadro
Distruggi questo oggetto
Figure 28: Actions Category

1. The object that calls this block makes another object appear by constructing it from one of the definitions provided, you can also set the speed and direction, as well as the type of alignment with respect to the calling object. The 'Relative' option allows you to specify whether this direction refers to the scene, or to the current rotation of the calling object, for example if it is active, the object is rotated to the right, the ’forward’ direction will point to the right.
2. Same as the previous block, but uses the position of another object in the scene to determine the direction in which the object appears
3. It allows the calling object to be able to ‘pick’ an object present in the scene. The object picked will be positioned in front of the caller and will follow all its

movements. An object can only take one object at a time. The most common application of this block is to make objects to hold in your hand in first-person games.

4. Releases the object previously picked from the caller if it exists.
5. It allows you to run one of the preset animations to the 3D model of the calling object, also setting its playback speed. These animations are not based on ‘bones’ but on a set of transformations, so it will be possible to apply them to any 3D model in the appropriate library.
6. Stops the animation that is currently in progress.
7. Allows you to add different particle effects to the position of the calling object.
8. It allows you to select a sound and play it at the desired volume. To select the sound, a popup like those already seen above will appear.
9. It allows you to show a text on the screen by specifying its position, color, and the possible duration of the event. If a variable or attribute is placed here, both the name of the element and its value will appear on the screen. If you want to show only the content of the variable, you can use the block to create a text in the ‘Values’ category.
10. Hides the text currently shown at a specific location on the screen.
11. It destroys the calling object, removing both his 3D model and all his behaviors from the scene. The effect of this block is not immediate but will be accomplished after performing all the behaviors of all objects, including the caller of this block.

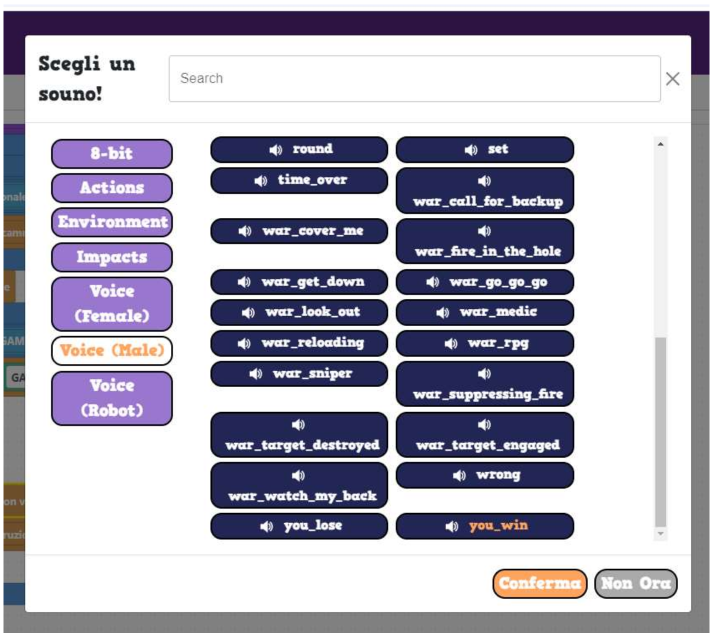


Figure 29: Sound FX Chooser

6.2.4 Situations Category Blocks

The blocks within this section allow you to manage the different events that may occur during the execution of the game.

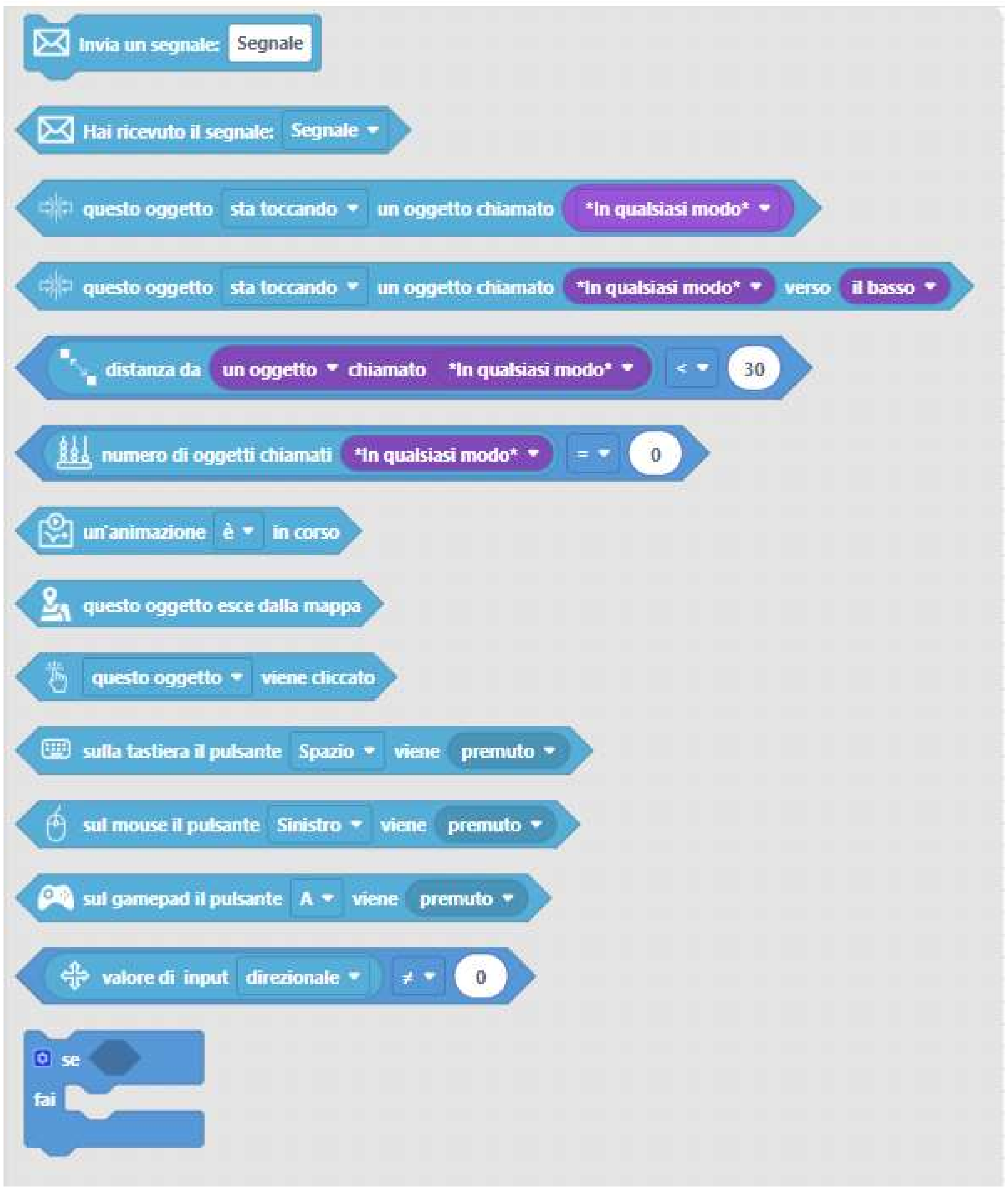

Invia un segnale: Segnale
Hai ricevuto il segnale: Segnale
questo oggetto sta toccando un oggetto chiamato *In qualsiasi modo*
questo oggetto sta toccando un oggetto chiamato *In qualsiasi modo* verso il basso
distanza da un oggetto chiamato *In qualsiasi modo* < 30
numero di oggetti chiamati *In qualsiasi modo* = 0
un'animazione è in corso
questo oggetto esce dalla mappa
questo oggetto viene cliccato
sulla tastiera il pulsante Spazio viene premuto
sul mouse il pulsante Sinistro viene premuto
sul gamepad il pulsante A viene premuto
valore di input direzionale ≠ 0
se
fai

Figure 30: Situations Category

1. Allows you to send a signal by specifying its identifier (in the form of a string) to all other objects currently in the scene. When an object generates a signal, it does so to communicate to other objects that they will have to do something when they receive it. The term 'signal' was chosen instead of the more commonly used 'message' because in the experimentation phase this

confused the boys a lot making them think that it was a message to be shown on the screen.

2. Returns *true* if the object has received a certain signal. This allows you to build custom logic to manage the signal in the way you prefer.
3. Returns *true* if the object has collided in the specified way (touching, just touched, finished touching) with an object named by the given name.
4. As above, but more stringent because it also specifies the direction in which the impact must occur to be considered valid (for example, if I have to program the press of a button, I can check that this is touching an object upwards).
5. Returns *true* if the distance between the calling object and the one specified as a parameter respect the selected comparison.
6. Returns *true* if the count of objects with the specified name respects the selected comparison.
7. Returns *true* if an animation is currently running on the calling object.
8. Returns *true* if the object is no longer within the scene.
9. Return *true* if the selected item (the calling object, or the entire screen) is clicked, also including any touchscreen events.
10. Returns *true* if the specified keyboard input event has occurred.
11. Returns *true* if the specified mouse input event occurred.
12. Returns *true* if the specified gamepad input event occurred.
13. Returns *true* if the directional input value (arrow keys, WASD keys, analog stick, or directional pad) chosen respects the selected comparison. Inputs along a certain direction have a value between -1 and 1, while if the direction is not specified, then the value will be between 0 and 1.
14. This is a conditional block, it was also put here because it has many applications within this category, but we will talk about it in more detail in the ‘Programming’ category.

6.2.5 Level Category Blocks

Here you will find all the blocks that can alter or control the status of the various scenes.

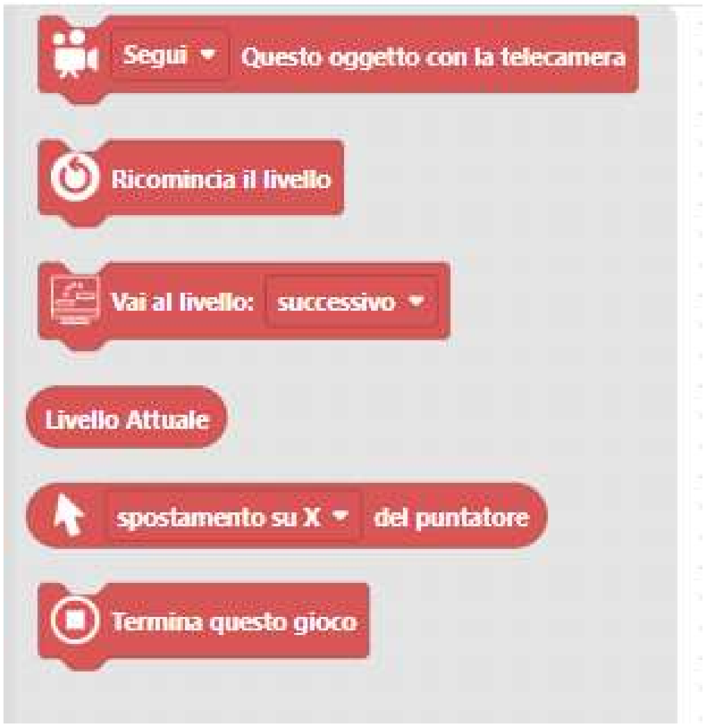


Figure 31: Level Category

1. This block allows you to follow (or stop following) the calling object by the scene camera.
2. Allows you to restart the current level, restoring the initial state of the scene.
3. Allows you to switch to another scene. It can be a specific scene, or the next one, in the latter case, the first non-empty scene is considered next and is found following the numerical order. If there are no more valid scenes, then the game will return to the main menu.
4. Returns the number that identifies the current scene, from 1 to 10.
5. Returns the contents of scene variables related to the pointer, such as position or movement (position change from the previous frame).
6. Immediately end the game by taking it to the main menu.

6.2.6 Values Category Blocks

This category deals with all the value types that can be set within the code.

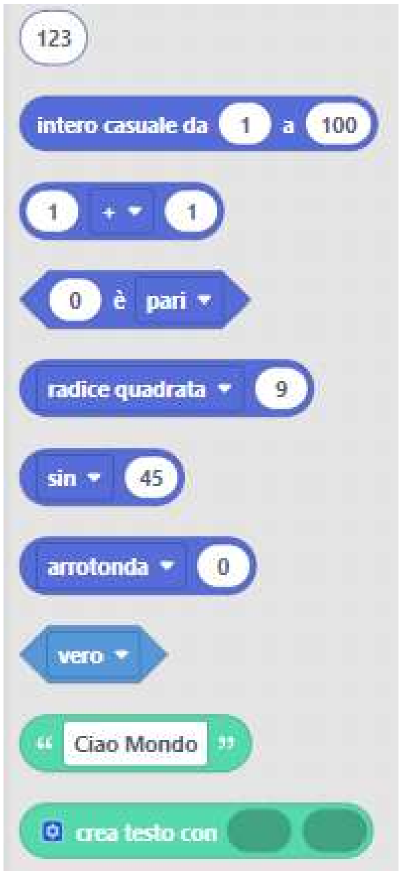


Figure 32: Values Category

1. A numeric value.
2. Integer value chosen randomly between two extremes (inclusive).
3. Returns *true* if the number passed as a parameter respects the chosen mathematical property (even, odd, prime, integer, positive, negative, divisible by).
4. Returns the result of the mathematical function to which the specified numeric parameter is passed (square root, absolute value, unary minus, natural logarithm, base 10 logarithm, power base *e*, power base 10).
5. Returns the result of the trigonometric function (sin, cos, tan, asin, acos, atan) to which the specified numeric parameter is passed.
6. Returns the numeric parameter rounded with the selected method.
7. Boolean value (true/false).
8. Textual value (string).
9. Constructs a string by concatenating several variables together. By clicking on the blue wheel on this block you can change the size of the list.

6.2.7 Programming Category Blocks

In this category we find all the blocks that help to define the logical flow of the program.

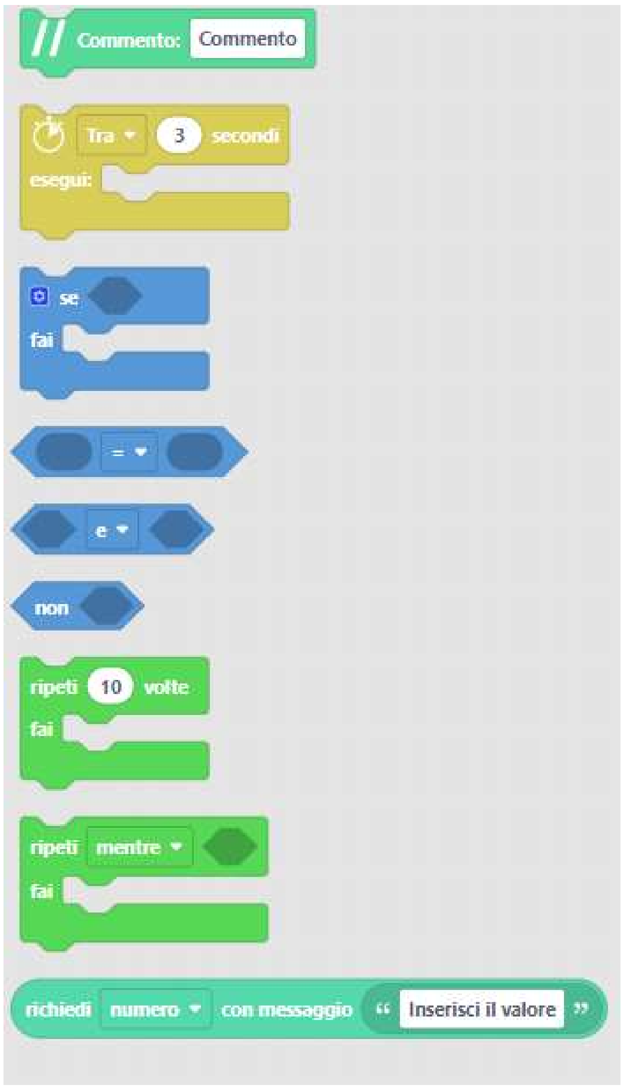


Figure 33: Programming Category

1. This is a comment, you can specify a text to be able to better organize your code, but it will not affect in any way the execution of the final program. Another way to write a comment is to use the context menu of blocks activated with the right mouse button, but the display of the comment in this case will be a speech bubble (which can also be hidden) that points to the block. This block has the function of leaving more conspicuous comments that cannot be hidden.

2. This block allows you to perform a sequence after a certain amount of time. The block can be set either to execute instructions only once, or to execute with a regular time interval. If a variable is used as a parameter to specify the time interval of a repetition, this period will depend on the current value of that variable at each iteration, and not just on its initial state.
3. This block allows you to execute a statement only if the condition passed as a parameter is verified. Using the blue wheel, you can change the number of conditional statements that the block can support (‘else’, ‘else if’ statements).
4. It returns *true* if the comparison is verified.
5. Returns to *true* if the logical operation is verified.
6. Defines a cycle to be repeated a finite number of times.
7. Defines a cycle to repeat until a certain condition is verified.
8. Returns input of the specified type that the user wanted. To ask for input, a pop-up is shown on the screen containing the text passed as a parameter.

6.2.8 Variables Category Blocks

In this category we could manage the global variables of our program. A global variable will be shared among all objects in the system, so if one object changes its value, for example, all other objects will see that change.

This is a dynamic category, i.e., the number of blocks depends on how many variables have been defined.

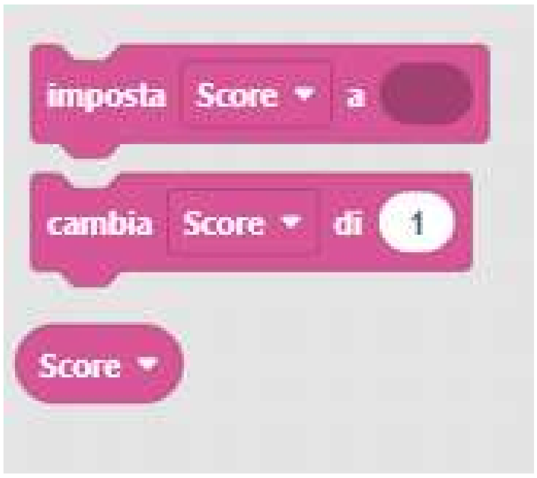


Figure 34: Variables Category

1. Sets the value of the variable to the one passed as a parameter, whatever its type.
2. Allows you to change the current value of the variable by incrementing it by the value specified as a parameter. Useful with numeric values.
3. Returns the current value of the variable.

6.2.9 Attributes Category Blocks

This category is very similar to the variables one, but this time the values refer to the single object that declares them, so each object has its own copy of this value and only that object can modify it; in fact, this category has the same color as the ‘Objects’ category, but has been separated since it concerns a rather optional concept that is not essential to make simple games.

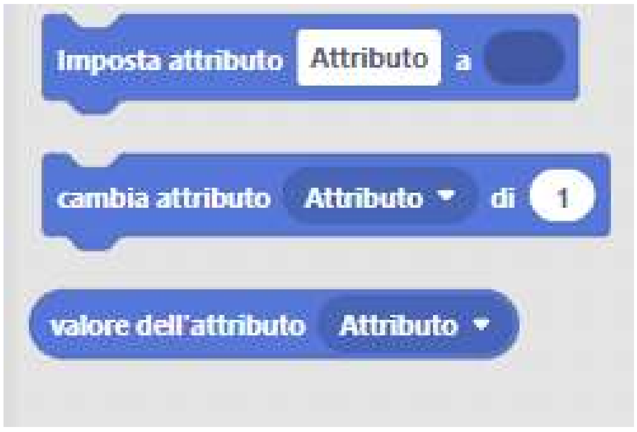


Figure 35: Attributes Category

1. Sets the attribute value to the one passed as a parameter, which can be of any type. Also, if the attribute has never been set within that object instance, it is also declared.
2. Allows you to change the current value of the attribute by incrementing it by the value specified as a parameter. Useful with numeric values.
3. Returns the value of the attribute.

6.2.10 Custom Blocks Category Blocks

Here the user can define his own custom blocks, which at the computer level are equivalent to function definitions. This is also a dynamic category, so the exact number of blocks depends on how many functions have been defined.

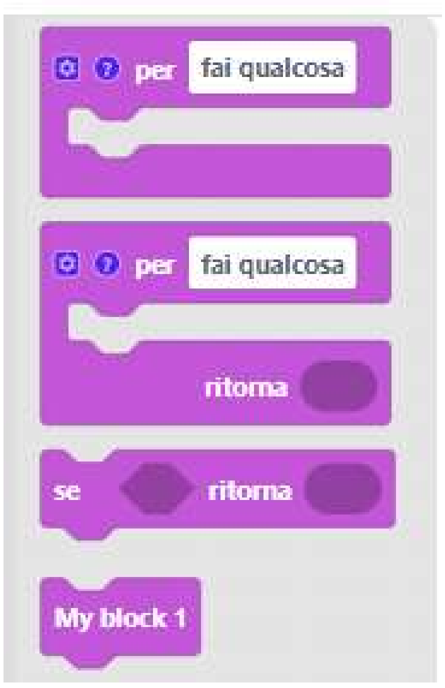


Figure 36: Custom Blocks Category

1. Creates a block with the specified name that executes statements without returning any values. Using the blue wheel, you can set optional parameters that can be passed to the block when called.
2. Creates a block with the specified name that executes statements to return a value of any type. Using the blue wheel, you can set optional parameters that can be passed to the block when called.
3. This statement should be used within the definition of a custom block. If the specified condition is verified, the function immediately returns the value set as a parameter.
4. Call a custom block.

7. Software Architecture

We now need to choose the main programming language for the project. We want to build. The huge quantity of libraries and frameworks available for this language is enough to make it also the choice for this project. Let us see how we can build this software.

7.1 Libraries and frameworks

- **Main UI:** If the idea is to create a web application, then we firstly need to pick up a front-end library, in this case ***React.js*** is right for us. It is well supported and easily extensible, a feature that we really need to create this tool.
- **Game Graphics and Assets:** here the final choice was to rely on ***p5.js***, a library that supports image and models loading, 3D graphics, sounds, and has an extension for VR and AR. This library has also the advantage of being beginner-friendly designed, which leaves the door open for future user-writable extensions using this library. However, some little adjustments had to be made at the time I am speaking*, p5.js* support textures but not when the 3D model is loaded from an URL, so I added this feature. Furthermore, ***p5.xr***, the extension that make AR and VR rendering possible on p5.js, had some problems on axis orientation and input handling while using immersive AR on a mobile device (which is actually caused by ***WebXR***, the native web library for extended reality), so I had to find workarounds to fix this issues.
- **VPL:** this is the core of the user experience, and luckily a framework by Google called **Blockly** exists. This framework allows us to create a block-based programming environment, and it is fully customizable. It provides primitives to design blocks and coding workspace, and it also offers methods to create a custom compiler for the code "written" by users. It is simply perfect for us.
- **Game-Engine:** having seen the nature of this project, I have chosen to implement a custom one myself, keeping it as simple as possible, I already have worked on game engines before, so it is not a new field for me. We will discuss the architecture of this Engine later in this chapter.

- **Back-end:** creating a web application implies that data must be processed and stored somewhere. In order to simplify the server-side application building I have chosen to rely on ***Firebase*** by *Google*, which allows you to access back-end functions such as user authentication, data storage and real time messages (that is exactly what we want to do for our project) without having to write server-side code. All we need to do is create and setup the project on the *Firebase* website and import the *Firebase* API in the client-side project to starting using it.

7.2 Project Structure

How are all these pieces linked together?

The main GUI of the software was created using *React*.js, but within it I had to insert custom containers both to be able to render the workspace created using *Blockly*, and to be able to render the 3D scene editor created with *p5.js*, since the first uses a rendering based on a 'Virtual DOM', the second based on the SVG format, while the third on *WebGL.*

The software defines an API to be able to communicate with *Firebase*, which is used to allow users to **authenticate** and to **save projects** in the **database**.

Every time the user modifies a game or scene, the change made to the project enters a queue, and every 3 seconds, if there are changes in the queue these are saved on the server.

Projects are serialized in a JSON format and stored in the database. This format includes all the **metadata** for the project (e.g., user, date), the *settings* the game will be run with, the serialization of the **scenes** (always in JSON format) and the serialization of the **code workspace** (in XML format).

Each user can edit a project only if he is authenticated and only if they own the project, if the user tries to edit a project created by another user, this is automatically copied into their account.

When a game is created, a **URL is generated to identify it**, and all users with that URL (even those who are not authenticated) will be able to play it.

Asset previews are uploaded from public files located at the same URL at which the project is hosted. And to **optimize the number of calls made to the server**, a **caching mechanism** was implemented for the most frequent and heavy asset types (3D models and sounds).

As for 3D models, previews are dynamic, so they are rendered when you load the model into memory. Also, the 3D models loading method provided by *p5.js* did not support textures. I therefore rewrote the loading logic from its parent project Processing, which supports them.

To populate the **3D Models library**, I downloaded most of assets from *PolyPizza*, while others were created using *Asset Forge.*

Sounds and **music** mostly came from the amazing *Kenney* work (who is also the creator of *Asset Forge*), while others were synthetized.

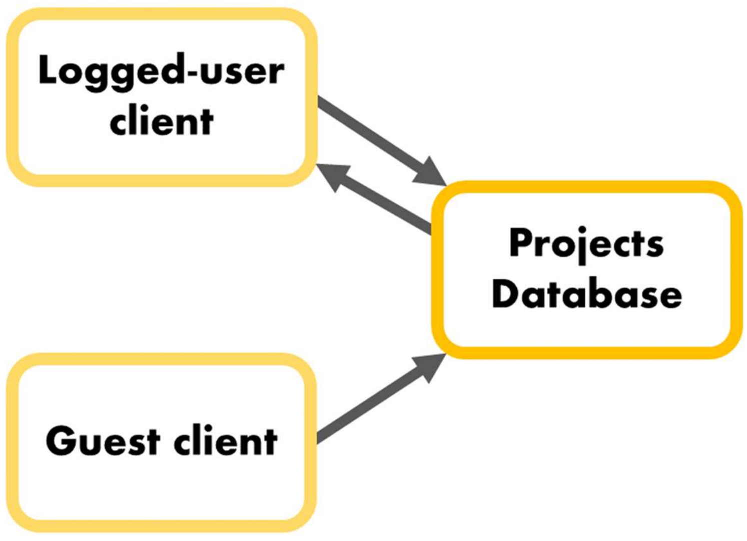


Figure 37: Main Project Structure

7.3 Game Engine Architecture

Many of the **3D physics engines** currently available for the JavaScript programming are not consistently supported and have a number of incompatibilities with the architecture chosen for VPL. In addition, they have features that are far too advanced for the purpose of this project that only risk weighing down performance and unnecessarily complicating even the simplest actions.

I evaluated the cost to pay to adapt these engines to the project compared to the benefits they would bring, and in the end, I decided to write a simple 3D physics engine for this project with the aim of being as light and synthetic as possible.

Outside of this thesis, I have been working on game engines for years, so the complexity of this choice is lowered even further.

The engine consists of **only five classes**:

1. **<u>Vector</u>:** defines a three-dimensional vector and all the mathematical functions that involve it.
2. **<u>AABB</u>:** this is the object collider, which is an object used to calculate collisions with other colliders, implements a *continuous collision detection* algorithm, based on axis-aligned bounding box (AABB).
3. **<u>Body</u>:** represents a physical body within the engine, also implements the physical properties of the object and functions to calculate integrals to get the position of the body in space.
4. **<u>World</u>:** represents the place where objects live, in fact, manages their life cycle and implements an algorithm to compute collisions, which uses a **broad phase** (algorithm that is used to determine which collisions could occur, and which not) to reduce the number of iterations to be carried out, after which it calculates the **position and the normal** of the impact for colliders that have passed the broad phase, and for the collisions that have actually occurred, it calculates the **response** and communicates it to the objects.
5. **<u>Contact</u>:** contains contact information between two objects, such as the objects involved in the collision, the contact normal and the velocity of the collision.

To **bridge** physics and graphics, **two main classes** are used:

1. **GameObject:** which collects the outputs of the physics engine from a single *Body* and defines both methods to be able to display it on the screen, and high-level logic.
2. **Scenes:** embed a *World* and abstracts its logic to a higher level as well as providing functions to be able to draw it on the screen.

To make the engine **communicate with the various graphics engines** (*p5.js* for 3D and *WebXR for XR*) I created a collection of classes that represent the various **cameras** for all different situations, including the different possible settings for the 3D graphics engine. Which are hooked to a scene and use interfaces written specifically to set and obtain the desired information from the reference graphics engine. In addition to all this, classes have also been defined to manage **lighting** and **shadow casting**.

An **input management system** has also been created, to allow the game engine to interface in the same way to the different supported input devices (mouse, keyboard, touchscreen, and several types of gamepads).

The **sound API** was written as an abstraction of the *p5.sound.js* one.

Finally, an interface called ***EngineAPI*** has been defined, which will allow the **VPL compiler** to interface more easily with the key features offered by the game engine, abstracting them at a higher level.

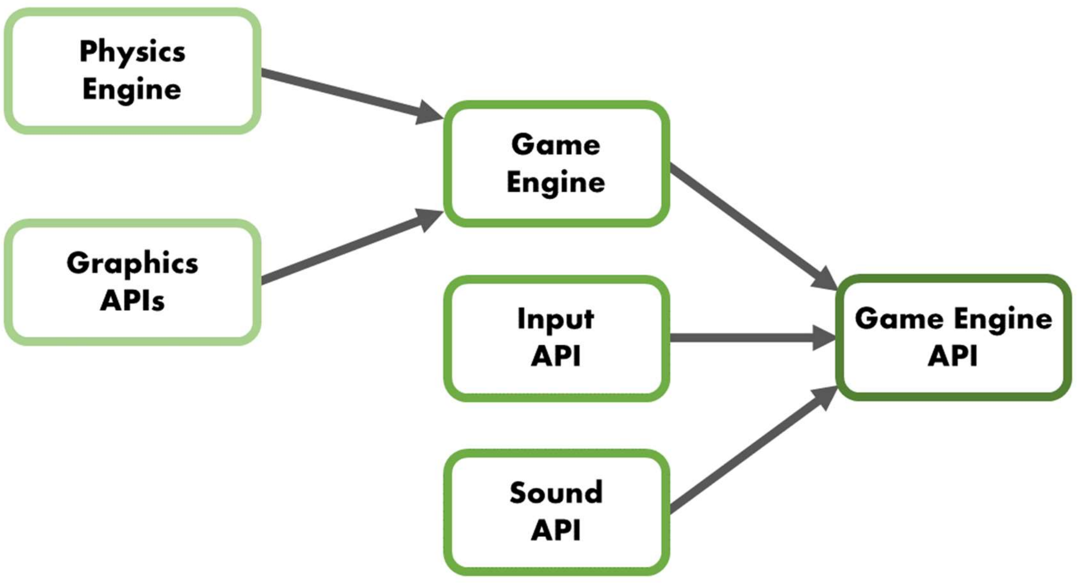


Figure 38: Game Engine Architecture

7.4 Games and Compiler Architecture

To make the game you create playable, the following steps are performed:

1. The user presses the 'Play' button from *Fairytool* or accesses the URL of a game.
2. The client requests from the server the project file saved in the database, and if everything is valid, it receives it.
3. The project file is passed to a **compiler**, which will use scene serialization and user-created code (present within the project file), and the ***EngineAPI*** to build a *JavaScript* file.
4. The built JavaScript file is not stand-alone, and is therefore passed to a component called ***GamePlayer,*** which will take care of preparing the rendering environment and will execute the JavaScript code once it is ready.

I do this way to make sure that any changes in the *EngineAPI* or system architecture allow games to remain more resilient and always runnable.

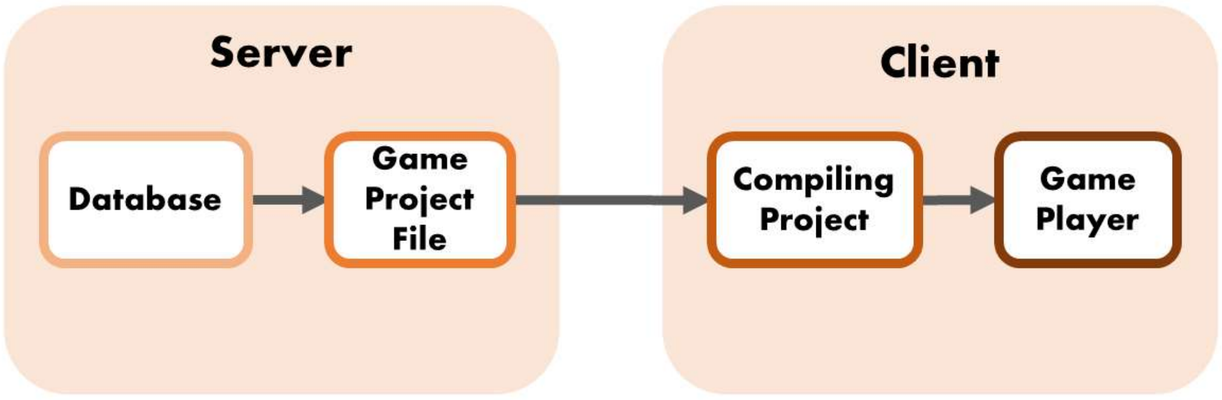


Figure 39: Running a Game

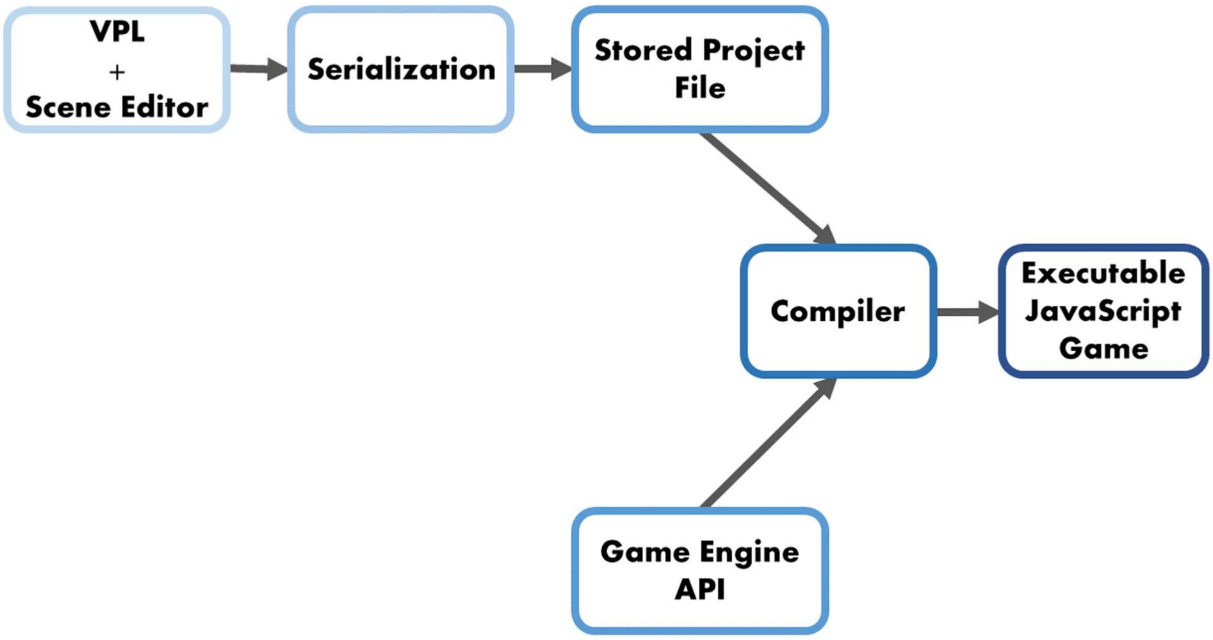
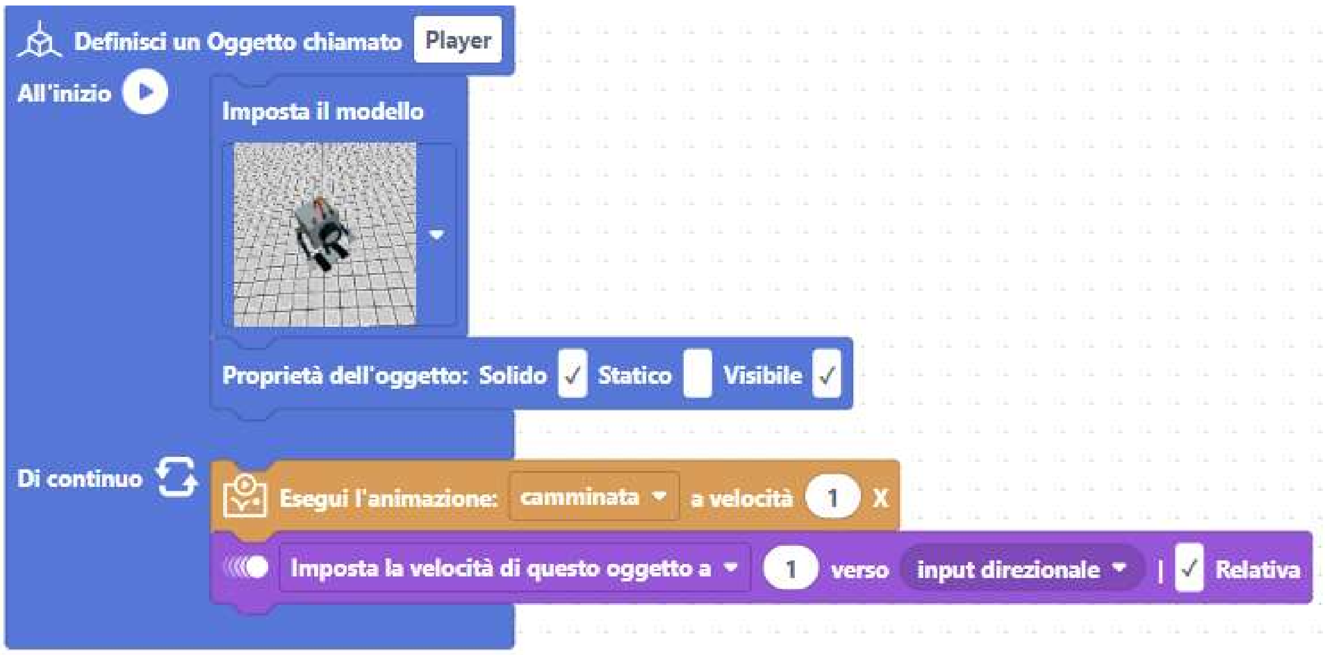


Figure 40: Compiling Architecture


Definisci un Oggetto chiamato Player
All'inizio
Imposta il modello
Proprietà dell'oggetto: Solido ✓ Statico Visibile ✓
Di continuo
Esegui l'animazione: camminata a velocità 1 X
Imposta la velocità di questo oggetto a 1 verso input direzionale | ✓ Relativa

```
let $models = {};
let $gameObject;
$EngineAPI.loadModel('/assets/models/Characters/Mini Robot/mini_robot.obj')
    .then(result => $models['/assets/models/Characters/Mini Robot/mini_robot.obj'] = result);
{
    let $gameObjectDefinition = new $EngineAPI.ObjectDefinition('PLAYER');
    $gameObjectDefinition.setup = function () {
        $gameObject = this;
        $gameObject.model = $models['/assets/models/Characters/Mini Robot/mini_robot.obj']
        $gameObject.solid = true;
        $gameObject.static = false;
        $gameObject.visible = true;
        $gameObject = null;
    }
    $gameObjectDefinition.update = function () {
        $gameObject = this;
        $gameObject.animation = $EngineAPI.Animations.WALK;
        $gameObject.animationSpeed = 1;
        $gameObject.setVelocity(
            $gameObject.absoluteDirection(
                $EngineAPI.currentScene.directions.INPUT.scale(1)
            )
        );
        $gameObject = null;
    }
}
```

Figure 41: Result of Fairytool VPL compilation

8. Experimentations

To evaluate Fairytool effectiveness, I had organized a new cycle of experimentations meeting, always collaborating with *Polistudio Amoruso.* Most of the group members were the same as the *Scratch* evaluation meetings, but new members were added, allowing me to follow both, people who had this little experience with Scratch and who have not.

8.1 Method

- At first, **I collected suggestions from the group members to create some simple games** based on their interest and wishes. In this phase I was the one that created those games using *Fairytool,* this was done **to check the software completeness and stability**, furthermore this was a first step to identify patterns and common game design goals whose implementation process should be facilitated inside the game engine.

- After a couple of weeks of software improvements based on the previous step result, I made a 10-minute introduction to *Fairytool* to the group, but nothing too specific was told to the kids to see how much information they can figure out by themselves. After being divided into small groups, **they were left free to explore the game engine without having to perform a specific task**, collaborating with each other to share information and start creating their first experiments independently. In this phase, I limit myself to observing the groups and intervene only upon explicit request from them. This allowed me **to start evaluating the usability and intuitiveness of the system**, as well as to note any technical problems of the software.

- After this first phase of exploration of the tool, the members of the various groups were reshuffled, this to allow everyone to be able to interact and bond with each other. The task of the groups has changed at this point, the aim is to **think of a game** that you would like to make and consequently try to **build it** with the aim of letting **the other groups play it**. No constraints or guidelines were imposed, but it was recommended to start from a simple idea, and then enrich it over time. Once again, the work was conducted independently by the various groups.

- **The previous step was repeated several times during several meetings**, where improvements were made to the software based on feedback from group members. The games the groups worked on have always remained the same and they have improved them from time to time, also trying to experiment with various technologies such as porting their games to smartphones or integrating different input devices such as gamepads

- **At the end of the experience**, students were given a questionnaire created with *Google Forms* to be able to summarize everything that emerged during the experience. Like the one used for the *Scratch evaluation*, this questionnaire was also created by combining the ***SUS*** (System Usability Scale) standard with a series of questions customized for Fairytool and another one adapted from *The Game Experience Questionnaire* [46]. The novelty compared to the previous one, however, is the introduction of a new set of questions of a psychological nature elaborated by the professionals of *Polistudio Amoruso*, with the aim of having a clearer picture of the lived experience both individually and in a group.

8.2 Results

Technical aspects related feedbacks:

- **Requests for assistance from all group members were significantly lower**. Furthermore, the requests focused only on the more advanced technical aspects such as for example some more complicated physics concepts or the relativity of reference systems, and the explanation necessary to be able to describe the functioning of these blocks was more concise than the one that was necessary to explain blocks causing problems in *Scratch*.

- **The time to make a game in *Fairytool* was much less** than that achieved with *Scratch*. The students were able to create more complicated programming logics in less time, also thanks to the reduction of the time spent asking for clarifications or documenting themselves. Some members of the group have used *Unity* in the past, these guys reported that according to them creating games in *Fairytool* is much easier and faster than this game engine.

- The presence of **templates** to explore integrated into the game engine was appreciated as it allowed you to immediately see the software in action.

- The presence **of custom icons on each single block** has allowed them to be immediately identified, as well as in some cases being able to receive further clarifications on its behavior.

- The **single workspace** page and the **automatic collapsing and rearranging** blocks has highly decreased the confusion caused by the interface. No member of the group ever expressed doubts about the placement of the blocks or made a mistake by editing the wrong block

- **The part of the software that required the most changes and improvements** across iterations **was the level designer**. Orienting yourself within a 3D environment is decidedly complex compared to a 2D one, above all because professional 3D editing software normally uses many commands and shortcuts to manage the scene, while in our case everything needs to be as simple and intuitive as possible. The 3D scene editing system had worked quite well since day one, but some shortcomings emerged as the levels created by the groups became more complex, such as the lack of controls that bring the view of the scene to predefined angles, or the possibility to create multiple selections, or the need for a simplification of the commands related to the use of the mouse buttons, which have been heavily decreased in favor of a more GUI-oriented approach (much more intuitive). In the last few iterations all groups were able to easily do what they wanted.

Experience related feedbacks:

- The kids found the idea of **developing a 3D game** much more **attractive**, which made them more motivated and in a good mood.

- Being able to **experiment with different technologies** on the created games **intrigued the kids** and kept their attention high, for example they were happy to be able to bring their own gamepads and use them, or to be able to easily bring their game to their smartphone, or even for having discovered the functions related to the XR world. Some young people have also expressed a willingness to experiment with other technologies such as online multiplayer for example.

- We observed that after the experience and the reshuffling of the groups, most of the people **who had extreme relationship difficulties took the initiative and opened up to the other kids**, becoming in some cases really chatty. However, in other cases the improvement of the relational skills was less evident. However, all the people were able to communicate at least a little with the others, both in moments of work and in moments of chat, and given the starting conditions of the boys, this result is by no means obvious.

- Every member of the group has **improved their critical thinking skills**, at the beginning of the path, during the *Scratch* effectiveness evaluation, people had a tough time understanding coding concepts, but when we moved to *Fairytool* they have become more independent, and after several tries and errors, they all were able to think original solutions to different kind problems. Of course, the result depends on the initial condition of each single person.

Finally, **all the guys who took part in the experiment expressed their preference for *Fairytool* over *Scratch***. They justified this preference both in terms of graphic and stylistic results of their works, and in terms of freedom of action within the software due to the greater clarity of what they were doing.

9. Conclusions

Can the creation of videogames be considered a tool to learn something new both in the technical and relational fields? The answer to all of this is: **Yes**, but you must pay attention to several aspects.

The complexity of the game creation tool plays a big part in this**, if the tool is too complex, less motivated people lose interest** or develop severe frustration at not being able to build what they wanted. During the first evaluation of *Scratch* this happened, all the kids were not able to finish the work independently, and in most cases, they were not able to really understand the concepts needed to build the game as they were presented in the software.

When we switched to *Fairytool* the situation drastically changed, most of the **kids managed to obtain satisfactory results in complete autonomy** by requesting assistance only for more complex concepts and they were able to also learn the reasons through brief explanations and apply them later independently without requiring a call of assistance yet.

The attraction towards technologies such as 3D graphics or the XR world played a fundamental role in the enthusiasm of the boys which, as confirmed by *Dr. Claudia Amoruso,* who constantly followed the boys from a psychological point of view, was high when they have even heard of it.

Remaining on the subject of a psychological profile, the transformation of the boys, who we remember **had enormous relationship difficulties**, was nothing short of satisfactory, both for the enthusiasm shown during the process of making the video game and for the teamwork. The boys' psychotherapy sessions showed how **they finally felt part of a whole that mirrored them**. Furthermore, **all the boys were able to establish relationships with other members of the group**, and some of them were even exported outside of it, creating real **friendships**.

9.1 Future Developments

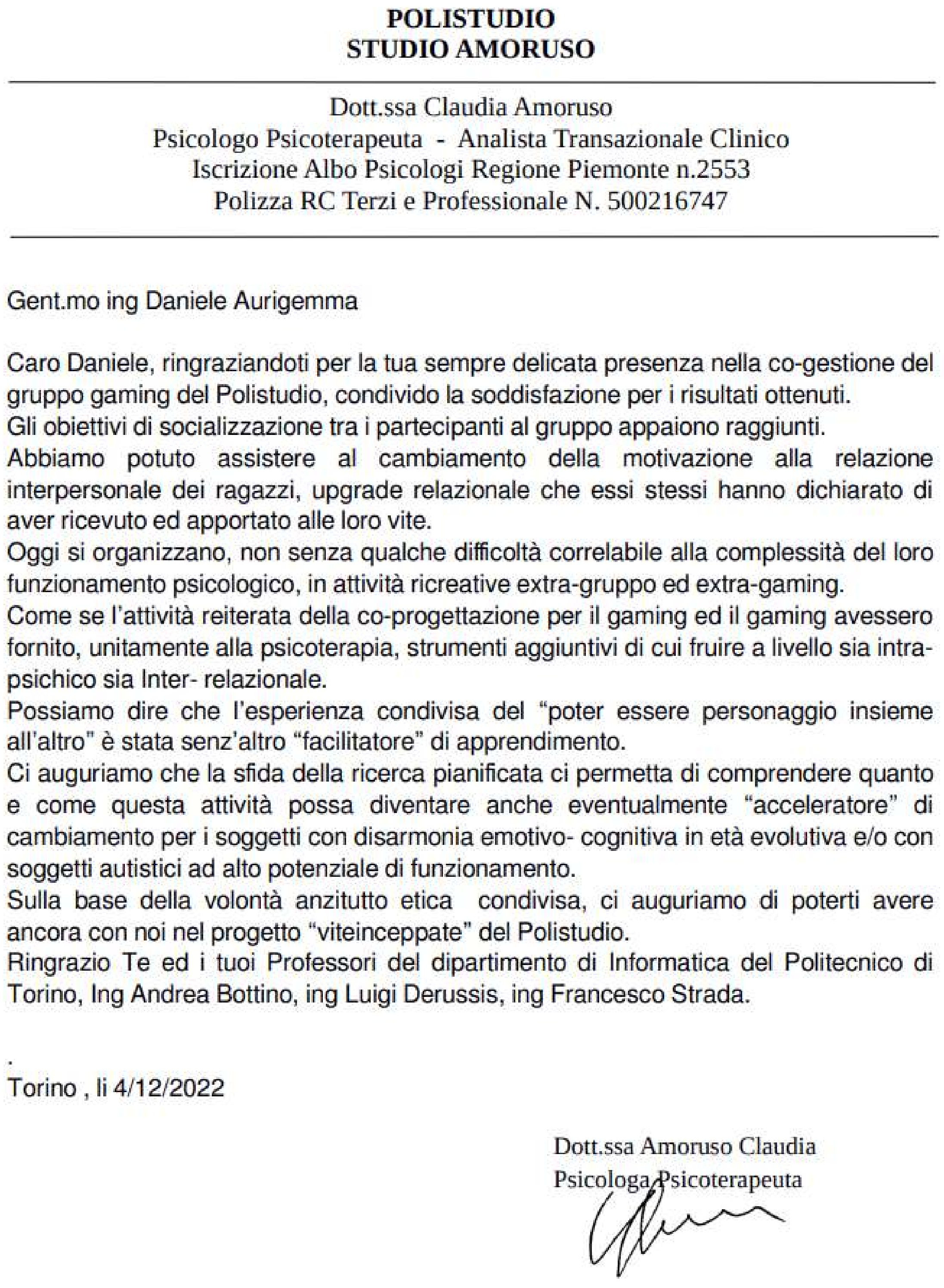
This project turned out to be incredible, and when we started, these were exactly the results we wanted to achieve. However, **the project is only at the beginning** and there is still a lot of work to be done to complete it. And both the supervisors of this thesis and the Polistudio Amoruso have expressed the desire to carry it forward.

These are the main points that should be explored in the future.

- **Localization**. The software was written in Italian because a group of Italian guys would have used it, but everything should be translated into as many languages as possible in order to be effective, since one of the advantages of VPLs is precisely that they can be displayed in the user's language.
- Make the project **Open-Source**: ensuring that a community is created around the project is fundamental for its development given its enormous complexity.
- **Add support for creating multiplayer games**. This possibility had already been explored at the beginning of this study, but there was simply no time to build the infrastructure. As we had already assumed, and as it also came out of the experiments, the world of online multiplayer is very appreciated by kids and would also be another possibility of relationship obtainable through *Fairytool*.
- Development of a **social aspect**. It would be nice if **young Fairytool users could inspire each other** by showing everyone their work and being able to play and study those made by others.
- **Continue the experiments and regarding the benefits on subjects with autism and other similar psychological diagnostic pictures** with the collaboration of *Polistudio Amoruso*. This point is fundamental and has been one of the pillars of this work, as well as one of the greatest satisfactions.
- Another of the developments that had been thought at the beginning of this work was to **extend the experimentation to schools**, this was also not possible within this thesis due to lack of time. But given the results obtained, we all really believe in the educational potential of this project.

9.2 From Polistudio Amoruso

I would like to close this thesis with the words taken from the conclusions of *Dr. Claudia Amoruso* herself. I leave the text she wrote in the original format and language:



POLISTUDIO
STUDIO AMORUSO

Dott.ssa Claudia Amoruso
Psicologo Psicoterapeuta - Analista Transazionale Clinico
Iscrizione Albo Psicologi Regione Piemonte n.2553
Polizza RC Terzi e Professionale N. 500216747

Gent.mo ing Daniele Aurigemma

Caro Daniele, ringraziandoti per la tua sempre delicata presenza nella co-gestione del gruppo gaming del Polistudio, condivido la soddisfazione per i risultati ottenuti.
Gli obiettivi di socializzazione tra i partecipanti al gruppo appaiono raggiunti.
Abbiamo potuto assistere al cambiamento della motivazione alla relazione interpersonale dei ragazzi, upgrade relazionale che essi stessi hanno dichiarato di aver ricevuto ed apportato alle loro vite.
Oggi si organizzano, non senza qualche difficoltà correlabile alla complessità del loro funzionamento psicologico, in attività ricreative extra-gruppo ed extra-gaming.
Come se l'attività reiterata della co-progettazione per il gaming ed il gaming avessero fornito, unitamente alla psicoterapia, strumenti aggiuntivi di cui fruire a livello sia intra-psichico sia Inter- relazionale.
Possiamo dire che l'esperienza condivisa del "poter essere personaggio insieme all'altro" è stata senz'altro "facilitatore" di apprendimento.
Ci auguriamo che la sfida della ricerca pianificata ci permetta di comprendere quanto e come questa attività possa diventare anche eventualmente "acceleratore" di cambiamento per i soggetti con disarmonia emotivo- cognitiva in età evolutiva e/o con soggetti autistici ad alto potenziale di funzionamento.
Sulla base della volontà anzitutto etica condivisa, ci auguriamo di poterti avere ancora con noi nel progetto "viteinceppate" del Polistudio.
Ringrazio Te ed i tuoi Professori del dipartimento di Informatica del Politecnico di Torino, Ing Andrea Bottino, ing Luigi Derussis, ing Francesco Strada.

.
Torino , li 4/12/2022

Dott.ssa Amoruso Claudia
Psicologa Psicoterapeuta
[illegible]

9.3 Thanks

We are at the end, and this is the moment where I allow myself to abandon the formalities and say that this was a ***crazy*** experience, which since it started, has changed many faces until we get to all this. This experience has combined everything I have always worked on inside, but especially outside the *Polytechnic of Turin*, and I find it to be the ***simply perfect*** conclusion of this path.

Therefore, I feel I must thank the supervisors of this thesis Bottino Andrea and Strada Francesco for the support provided during the thesis and professor De Russis Luigi for the interest shown in the project.

A 'thank you' never big enough to the staff of Polistudio Amoruso and especially to Dr. Amoruso Claudia for reasons that there is no need to even mention.

I thank my sweetheart Nuovo Caterina for supporting me during the realization of this project.

But above all I sincerely thank all the boys who took part in this experience!

10. Sources

3D game tools:

[1] AppOnboard Inc. *Buildbox*. https://signup.buildbox.com/
[2] Carnegie Mellon University. *Alice*. https://www.alice.org/
[3] Cyberix3D. *Cyberix3D*. https://www.gamemaker3d.com/
[4] Delightex. *CoSpaces*. https://cospaces.io/
[5] Epic Games Inc. *Unreal Engine*. https://www.unrealengine.com
[6] Google Llc. *Google Game Builder*. https://github.com/googlearchive/gamebuilder
[7] HypeHype Inc. *HypeHype*. https://hypehype.com/
[8] *Kodu Game Lab*. https://www.kodugamelab.com/
[9] Meta Platforms. *Horizon Worlds*. https://www.oculus.com/horizon-worlds/
[10] Nintendo. *Game Builder Garage*. https://www.nintendo.com/store/products/game-builder-garage-switch/
[11] Roblox Corporation. *Roblox*. https://www.roblox.com/
[12] Sony Interactive Entertainment Europe, Media Molecule. *Indreams*. https://indreams.me/
[13] Struckd AG. *Struckd*. https://struckd.com/
[14] Unity Technologies. *Unity*. https://unity.com/

2D game tools:

[15] Code.org. *Code.org*. https://code.org/
[16] Cosmo Myzrail Gorynych. *ct.js*. https://ctjs.rocks/
[17] *Flowlab*. https://flowlab.io/
[18] *Gamefroot*. https://make.gamefroot.com/
[19] GameSalad Inc. *GameSalad*. https://gamesalad.com/
[20] Google Llc. (2014). Dinosaur Game [Video game].
[21] Hopscotch Technologies. *Hopscotch*. https://www.gethopscotch.com/
[22] Microsoft Corporation. *MakeCode*. https://www.microsoft.com/makecode
[23] Neuron Fuel. *Tynker*. https://www.tynker.com
[24] Scirra Ltd. *Construct*. https://www.construct.net
[25] Scratch Foundation, DevTech Research Group. *ScratchJr*. https://www.scratchjr.org/
[26] Scratch Foundation. *Scratch*. https://scratch.mit.edu/
[27] YoYo Games Ltd. *GameMaker*. https://gamemaker.io

Technologies involved:

[28] Google Llc. *Blockly*. https://developers.google.com/blockly
[29] Google Llc. *Firebase*. https://firebase.google.com/
[30] Kenney. *Asset Forge*. https://assetforge.io/
[31] Kenney. *Kenney*. https://www.kenney.nl/
[32] Meta Platforms. *React*. https://reactjs.org/
[33] Mozilla Foundation. *MDN*. https://developer.mozilla.org/
[34] *p5.xr*. https://p5xr.org/
[35] *PolyPizza*. https://poly.pizza/
[36] Processing Foundation. *p5.js*. https://p5js.org/
[37] Processing Foundation. *Processing*. https://processing.org/
[38] Tango Technology Inc. *Tango*. https://www.tango.us/
[39] W3C Immersive Web Working Group. *WebXR*. https://immersiveweb.dev/

Books and articles:

[40] Bischoff, R., Kazi, A., & Seyfarth, M. (2002). The MORPHA style guide for icon-based programming. 11th IEEE International Workshop on Robot and Human Interactive Communication, pp. 482-487, https://doi.org/10.1109/ROMAN.2002.1045668

[41] Brendan, L.K. (2013). Swept AABB Collision Detection and Response. GameDev.net. https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/swept-aabb-collision-detection-and-response-r3084/

[42] Desurvire, H., Wiberg, C. (2009). Game Usability Heuristics (PLAY) for Evaluating and Designing Better Games: The Next Iteration. In: Ozok, A.A., Zaphiris, P. (eds) Online Communities and Social Computing. OCSC 2009. Lecture Notes in Computer Science, vol 5621. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-02774-1_60

[43] Google Llc. Custom Blocks: Best Practices. https://developers.google.com/blockly/guides/app-integration/best-practices

[44] Handley, L. D., Foster, S. R. (2020). *Don't Teach Coding: Until You Read This Book*. John Wiley & Sons Inc.

[45] Hu, Y., Chen, C.H., & Su, C.Y. (2021). Exploring the Effectiveness and Moderators of Block-Based Visual Programming on Student Learning: A Meta-

Analysis. Journal of Educational Computing Research, 58(8), 1467–1493. https://doi.org/10.1177/0735633120945935

[46] IJsselsteijn, W. A., de Kort, Y. A. W., & Poels, K. (2013). The Game Experience Questionnaire. Technische Universiteit Eindhoven. https://research.tue.nl/en/publications/the-game-experience-questionnaire

[47] Kuhail, M. A., Farooq, S., Hammad, R., & Bahja, M. (2021). Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. IEEE Access, vol. 9, pp. 14181-14202. https://doi.org/10.1109/ACCESS.2021.3051043

[48] Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M.N., Kumar, A.N., Ott, L.M., Paterson, J.H., Scott, M.'., Sheard, J., & Szabo, C. (2018). Introductory programming: a systematic literature review. Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. https://doi.org/10.1145/3293881.3295779

[49] Noone, M., Mooney, A. (2018) Visual and textual programming languages: a systematic review of the literature. *J. Comput. Educ.* 5, 149–174. https://doi.org/10.1007/s40692-018-0101-5

[50] Shiffman, D. (2012). *The Nature of Code: Simulating Natural Systems with Processing*.

[51] Vahldick, A., Farah, P.R., Marcelino, M.J., & Mendes, A.J. (2020). A blocks-based serious game to support introductory computer programming in undergraduate education. https://doi.org/10.1016/j.chbr.2020.100037

[52] Whitley, K. N., Blackwell, A.F. (1997). Visual programming: the outlook from academia and industry. In Papers presented at the seventh workshop on Empirical studies of programmers. Association for Computing Machinery, New York, NY, USA, 180–208. https://doi.org/10.1145/266399.266415