Silesian
University
of Technology

FINAL PROJECT

Desktop application supporting learning of stereometry

Adrian BEDNORZ
Student identification number: ab306018

Programme: Informatics
Specialisation: Computer Graphics and Software

SUPERVISOR
dr inż. Alina Momot
DEPARTMENT Katedra Informatyki Stosowanej
Faculty of Automatic Control, Electronics and Computer Science

Gliwice 2026

Thesis title

Desktop application supporting learning of stereometry

Abstract

The aim of the work is to design and implement a desktop application that supports children in learning geometry. The user of the application will have several types of tasks to choose from, in which it will be necessary to calculate the areas and volumes of three-dimensional geometric figures. There will also be an educational component where it will be possible to visualize the changes in the areas and volumes of the figures as the parameters describing the figure are altered.

Key words

application, education, geometry, stereometry

Tytuł pracy

Aplikacja na komputer stacjonarny wspomagająca naukę stereometrii

Streszczenie

Celem pracy jest zaprojektowanie i wdrożenie aplikacji desktopowej wspomagającej dzieci w nauce geometrii. Użytkownik aplikacji będzie miał do wyboru kilka typów zadań, w których będzie musiał obliczyć pola i objętości trójwymiarowych figur geometrycznych. Aplikacja będzie również zawierać część edukacyjną, umożliwiającą wizualizację zmian pól i objętości figur wraz ze zmianą parametrów opisujących figurę.

Słowa kluczowe

aplikacja, edukacja, geometria, stereometria

Contents

1 Introduction **1**

1.1 Introduction into the problem domain 1

1.2 Objective and scope of the thesis 1

1.3 Short description of chapters 3

2 Problem analysis **5**

2.1 Mathematical description of selected solids 5

2.1.1 Prism 5

2.1.2 Pyramid 6

2.2 Existing solutions 8

2.2.1 GeoGebra 3D Calculator 8

2.2.2 Cabri 3D and Cabri Express 9

2.2.3 Shapes 3D Geometry Learning & Drawing 11

3 Requirements and tools **15**

3.1 Functional requirements 15

3.2 Non-functional requirements 16

3.3 Used tools 17

3.3.1 Godot Engine 17

3.3.2 Neovim 18

3.3.3 Git and GitHub 18

4 External specification **19**

4.1 System requirements 19

4.2 Installation 20

4.2.1 Windows 20

4.2.2 Linux 20

4.2.3 macOS 20

4.3 User manual 20

4.3.1 Main menu 21

4.3.2 The 3D view 21

4.3.3 The navigation bar . . . 23
4.3.4 Task exploration . . . 23
4.3.5 Task solving . . . 28
4.3.6 Playground . . . 35
4.3.7 Settings . . . 39

5 Internal specification **41**
5.1 Nodes and scenes . . . 41
5.2 GeoApp structure . . . 42
5.3 The Main scene . . . 42
5.4 Autoloads . . . 43
5.5 Signals . . . 44
5.6 Representation of solids . . . 44
5.6.1 Figure . . . 44
5.6.2 Polyhedron . . . 45
5.7 Tasks . . . 46
5.8 User interface . . . 48
5.9 Polyhedron files . . . 49
5.10 Project file structure . . . 50

6 Verification and validation **51**
6.1 Identified bugs . . . 51
6.1.1 Unintentional camera zooming . . . 51
6.1.2 Issues related to edge cylinder transform . . . 53

7 Conclusions **55**
7.1 Obtained result . . . 55
7.2 Evaluation of used tools and technologies . . . 55
7.3 Further development . . . 56
7.4 Encountered difficulties . . . 56
7.4.1 Acquired knowledge and experience . . . 56

Bibliography **59**

Index of abbreviations and symbols **63**

List of additional files in electronic submission **65**

List of figures **68**

List of tables **69**

Chapter 1

Introduction

1.1 Introduction into the problem domain

With the development of technology, education changes from classical approaches to ones using tools and methods previously unavailable. As computers become more accessible and widespread, schools introduce more digital ways of teaching students. One of the goals of educational applications is to aid teachers in conveing knowledge more effectively than using classical methods. A well-designed educational application should be able to provide all the tools necessary to teach students a certain subject. It may also be installed on the student's personal computer, allowing them to study and practice efficiently even outside of school.

The thesis is concerned with applications which allow the user to gain and improve their skills in the field of stereometry. Stereometry (or solid geometry) is geometry in three-dimensional space. Stereometry is concerned with the measurement of surface areas and volumes of solid figures, such as polyhedrons, spheres, cylinders, and cones.

1.2 Objective and scope of the thesis

Learning stereometry may prove difficult to a student without adequate tools. There are several tools to learn stereometry without the assistance of a computer application. These include:

Blackboard and chalk Offer only static representations of figures which are difficult to draw precisely. The drawings are usually only wireframes and do not have shading. This might make it hard to visualize what the drawing is meant to represent. Drawing the figures is also time-consuming.

Printed textbooks Include shading, but are still limited to static images which can be viewed from a single angle.

Physical models Present a true three-dimesional representation which can be rotated. The dimensions are constant and not every figure might be available as a physical model.

The objective of the thesis is to design and implement a desktop application which helps the user learn stereometry.

Nowadays, web applications are more popular than desktop applications. Despite that, desktop applications:

- do not require an internet connection after installation, while web applications require constant connection to the server hosting the service,
- are faster and more responsive as no data needs to be transfered between the client computer and a remote server,
- give the user control over software updates - the user may use an older version if desired.

The application will allow the user to solve solid geometry tasks and create polyhedrons from vertices, consequently expanding their skills and knowledge in this field. Its target group is students aged 12 to 15. The application can be used as a didactic tool in the classroom to help the teacher in teaching stereometry. It can also be used at home by the student to practice stereometry outside the classroom. The application will feature tasks. A task will contain one or more three-dimensional figures. Each task will be split into several steps. Each step will require the user to calculate one portion of the problem described by the task. After providing an incorrect answer the user will receive a tip to help them solve the task successfully. A module allowing the user to explore the tasks and view the correct answers will be available. In this module, the user will have the possibility to change the task data and see how the shapes change dynamically. An important part of the application is its customizability - the user should be able to tweak the settings of the application such that they get the best experience tailored to their needs.

The application will be built using the Godot game engine. While designed primarily with video games in mind, it can be used to create other types of software. The application requires a modern graphical user interface (GUI) and 3D rendering capabilites. Godot provides a suite of tools which simplify working with both of those systems. Other solutions like Flutter[1] and Electron[2] were considered but ultimately rejected. They provide a wide range of tools for developing desktop applications but do not have first-class 3D rendering support which would make developing the application more difficult.

[1] `https://flutter.dev`
[2] `https://www.electronjs.org`

1.3 Short description of chapters

The first chapter introduces the topic of the thesis and its problem domain. It presents the objective and scope of the thesis. The second chapter analyses the problem and presents already existing solutions. The third chapter lists the functional and non-fuctional requirements, and presents the tools used in the project's development. The fourth chapter serves as the user manual. It specifies the sytem requirements and provides instructions on how to run and use the application. The fifth chapter is focused on the implementation of the project. It explains the architecture of the application. The sixth chapter describes how the project was tested. It also presents the bugs that were encountered and subsequently fixed during the project's development. The seventh chapter concludes the thesis. It compares the achieved results to the objectives of the thesis and explores prospects of the project's further development.

Chapter 2

Problem analysis

Stereometry is an essential part of mathematics education. It develops a student's ability to understand and reason about three-dimensional space. Knowledge of stereometry is important not only for further studies in mathematics, but also for fields such as physics, engineering, architecture, and computer science. In those fields, spatial thinking and geometric reasoning are frequently required. Through the study of three-dimensional solids, students learn to work with concepts such as volume, surface area, and spatial relationships between objects.

Despite its importance, stereometry is often regarded as a difficult subject by students. One of the main challenges lies in the need to imagine three-dimensional objects based on two-dimensional representations. Two-dimensional representations include drawings on a board or in a textbook. Static images and traditional teaching methods may not sufficiently convey the spatial structure of solids. This may lead to poor understanding and memorization without true comprehension.

An interactive application capable of visualizing three-dimensional solids can help with these difficulties. By allowing users to observe objects from different perspectives, inspect their components, and manipulate their parameters, such an application supports the development of spatial imagination. Visual feedback enables students to better understand how changes in dimensions or structure affect the properties of a solid. This strengthens the connection between geometric theory and its practical interpretation.

2.1 Mathematical description of selected solids

2.1.1 Prism

A prism is a polyhedron consisting of two congruent n-sided polygon bases and n joining faces. Every joining face is a parallelogram. A prism is called after its base, e.g. a prism with a hexagonal base is called a hexagonal prism. An n-gonal prism has $n + 2$ faces, $3n$ edges, and $2n$ vertices.

Special cases

An *oblique prism* is a prism whose joining faces are not perpendicular to the bases. A prism whose whose joining faces are perpendicular to the bases is a *right prism*.

A right prism with a rectangular base is also called a *cuboid* (see figure 2.1). A right prism with a square base is also called a *square cuboid*. A square cuboid whose height is equal to the side length of its base is a *cube*.

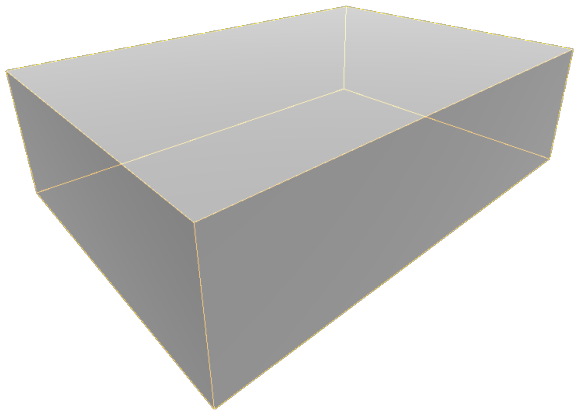


Figure 2.1: A right prism with a rectangular base.

Properties

Volume The volume of a prism is the product of the base area and the height. The formula for the volume is:

$$V = Bh, \tag{2.1}$$

where B is the base area and h is the height.

Surface area The surface area of a prism is the sum of all its faces' areas. The formula for the surface area is:

$$A = 2B + Ph, \tag{2.2}$$

where B is the base area, P the perimeter, and h the height.

2.1.2 Pyramid

A pyramid is a polyhedron consisiting of one polygonal base face connected to a point, called the apex. Each of the base's edges is connected to the apex, forming a triangle, called a lateral face. A pyramid with an n-gonal base has $n + 1$ faces, $2n$ edges, and $n + 1$ vertices.

Special cases

A pyramid is called a *regular pyramid* when its base is a regular polygon. The definition of a *right pyramid* varies between sources. Some sources define it as a regular pyramid whose height falls on the center of the base [6]. According to other sources, a right pyramid is a pyramid, whose height falls on the center of a circle circumscribed about the pyramid's base [11]. A right pyramid with a hexagonal base can be seen in figure 2.2. A pyramid with a triangular base is also called a *tetrahedron*.

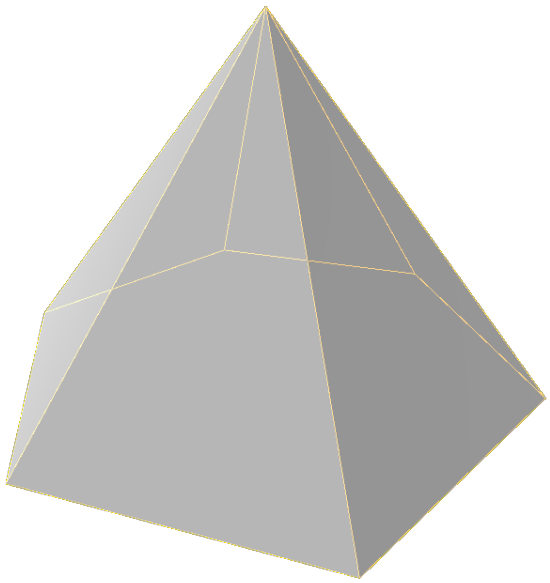


Figure 2.2: A right pyramid with a hexagonal base.

Properties

Volume The volume of a pyramid is one third of the product of the base area and the height. The formula for the volume is:

$$V = \frac{1}{3}Bh, \tag{2.3}$$

where B is the base area and h is the height [9].

Surface area The surface area of a pyramid is the sum of the area of the base and the area of the lateral faces. For a regular right pyramid with an n-gonal base and height h, the surface area can be calculated as follows:

Each regular n-gon can be divided into n congruent triangles. The base of the triangle is the side s of the pyramid. The triangle's height h_B is calculated as:

$$h_B = \frac{1}{2}s\tan(\frac{n-2}{n}90°). \tag{2.4}$$

Then, the polygon base area can be calculated by multiplying the area of one triangle by the number of sides:

$$A_B = \frac{sh_B}{2}n. \tag{2.5}$$

Height of lateral triangles can be obtained by using the Pythagorean theorem:

$$h_L = \sqrt{{h_B}^2 + h^2}. \tag{2.6}$$

The lateral area is calculated in a similar way to the base area:

$$A_L = \frac{sh_L}{2}n. \tag{2.7}$$

Finally, the total area is the sum of the base area and the lateral area:

$$A = A_B + A_L. \tag{2.8}$$

2.2 Existing solutions

2.2.1 GeoGebra 3D Calculator

The 3D Calculator[1] by GeoGebra is an extensive application which provides many tools for exploring 3D geometry. It allows the user to create and manipulate solids, points, lines, and polygons. It also provides tools for measuring angles, distances, areas, and volumes. The 3D Calculator can be of great help when exploring stereometry, but it does not provide didactic features. It is not suited for structured, guided learning. The measuring tools allow the user to calculate certain characteristics but do not teach how to do so. GeoGebra 3D Calculator does not provide support for keyboard control, the mouse being the only allowed input method. The application is only available on web and mobile platforms with no downloadable version for desktop computers. The user interface of the GeoGebra 3D Calculator is visible in figure 2.3.

[1] `https://www.geogebra.org/3d`

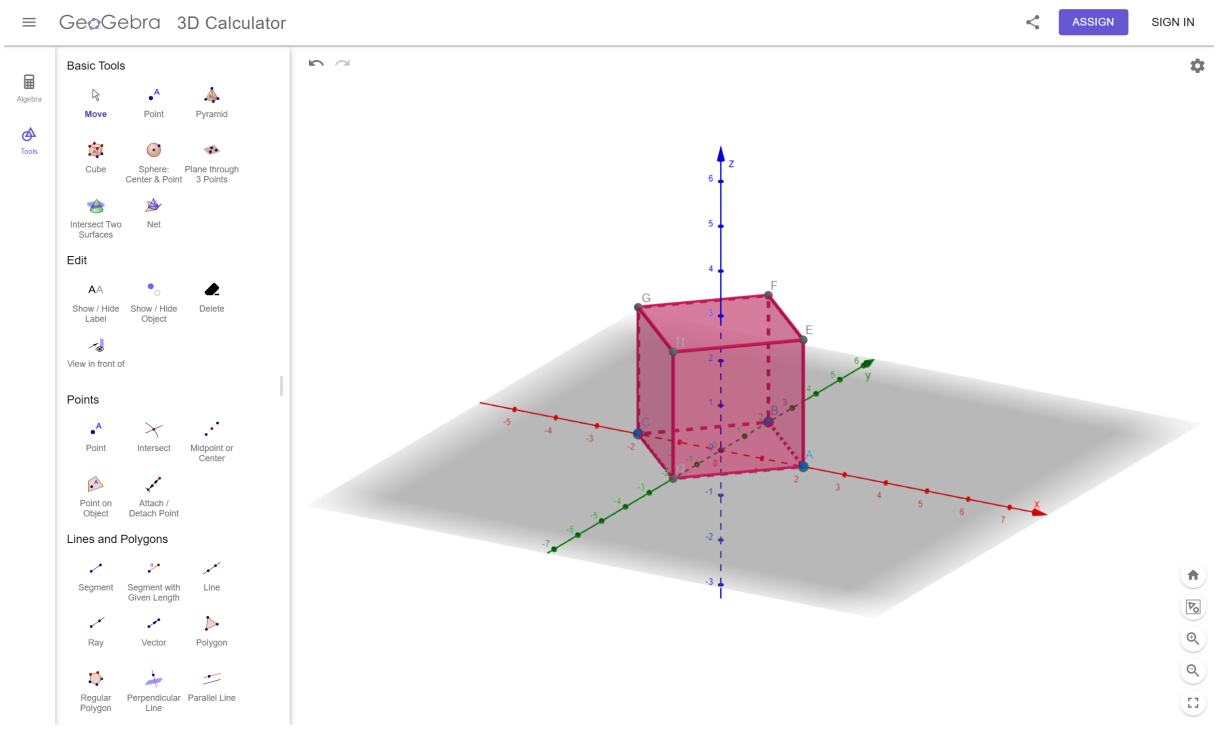


Figure 2.3: GeoGebra 3D Calculator.

2.2.2 Cabri 3D and Cabri Express

Cabri 3D[2] and Cabri Express[3] are 2 applications for learning three-dimensional geometry developed by Cabri. Both applications are designed with teaching in mind. Cabri products are closed-source, which means that the software will no longer be able to receive updates after the original creators stop developing them.

Cabri 3D

Cabri 3D is an application for the Windows and Mac OS operating systems. It is commercial software which requires the purchase of a license. The application is used primarily to create documents consisting of text areas and 3D views. This makes it suitable to be used by teachers to prepare lesson materials, which can be printed on paper. It is not suited to by used by students to explore 3D geometry or solve tasks. The user inteface has an outdated look and has some bugs when running on a system with multiple displays. When running on a system with two displays, the *Tool Help* window produces visual artifacts on the other display when it is moved. The user interface of Cabri 3D is visible in figure 2.4.

[2] `https://cabri.com/en/enterprise/cabri-3d/index.html`
[3] `https://cabri.com/fr/enterprise/cabri-express/`

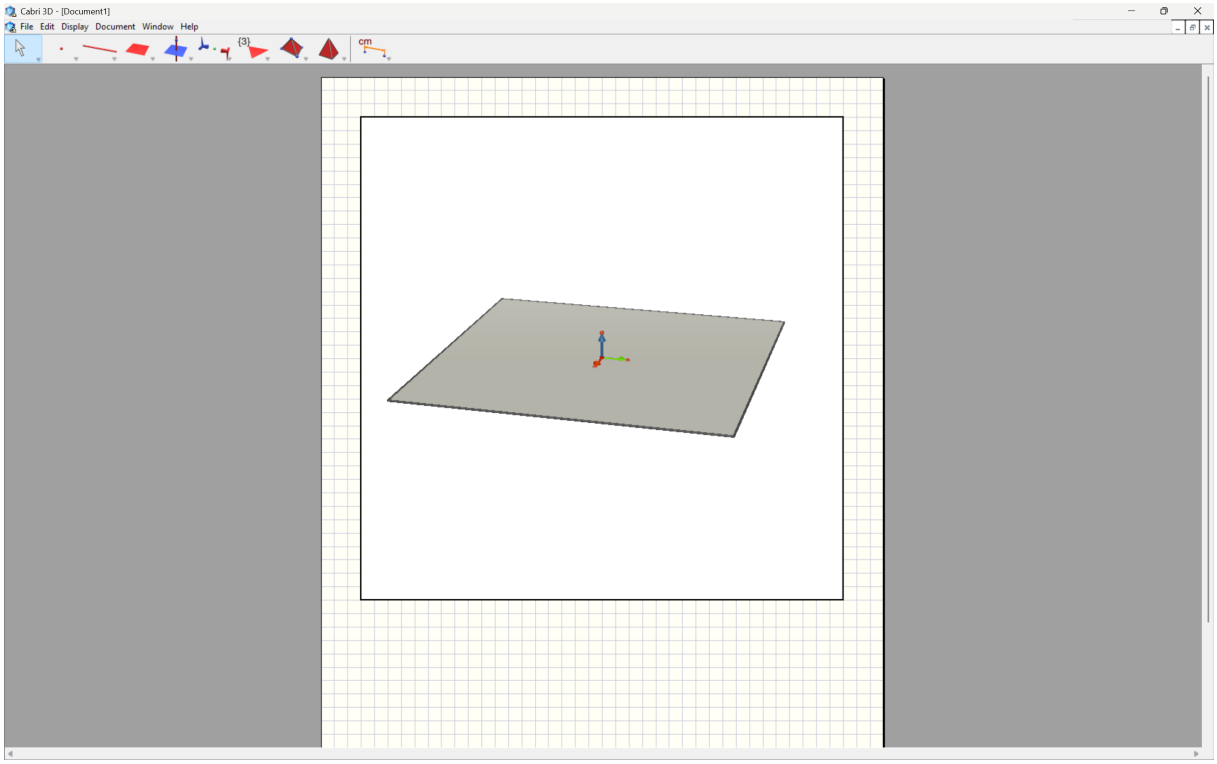


Figure 2.4: Cabri 3D.

Cabri Express

Cabri Express is a general mathematical application and features tools not limited to 3D geometry. It has tools related to stereometry but it is not specialized in this field. This, combined with it being closed-source introduces a problem. If one doesn't find the desired 3D tool in Cabri Express, it may never be added by the developers. Learning how to use the applications may be problematic, as most of the official tutorials for the applications are in French. Moreover, there are few unofficial learning resources in other languages. The user interface of Cabri Express is visible in figure 2.5.

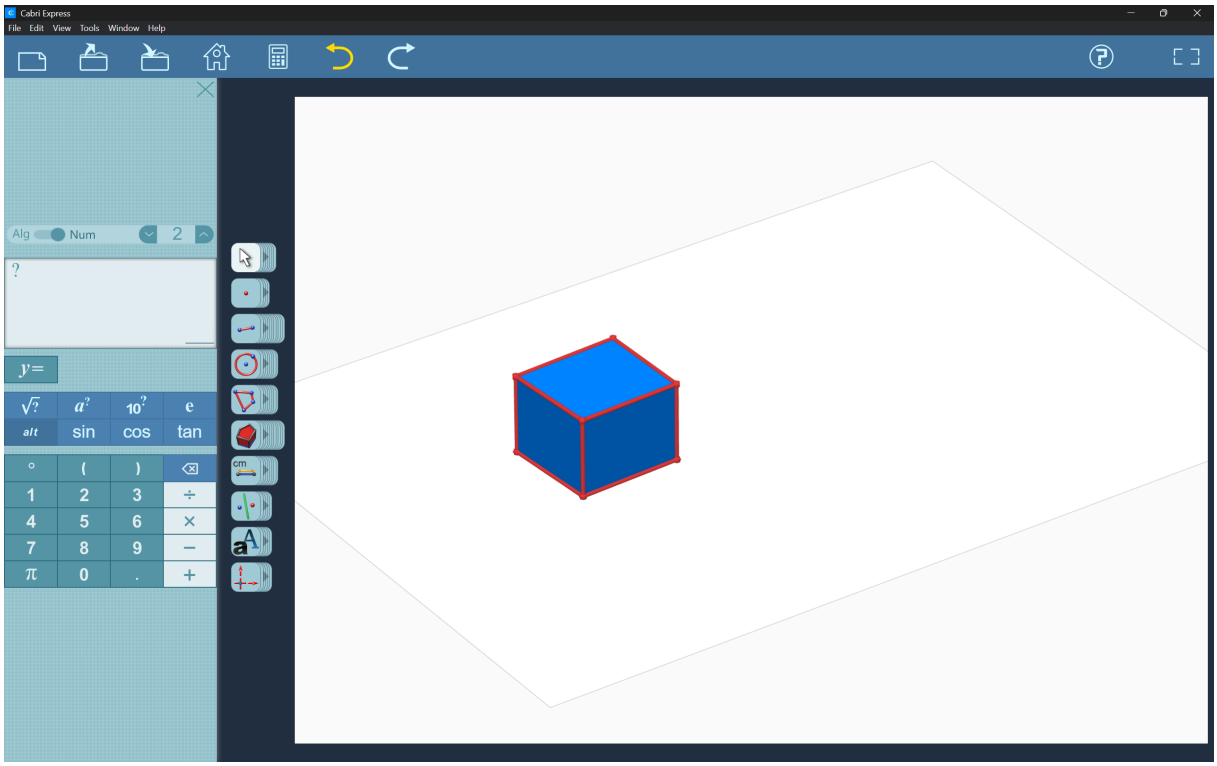


Figure 2.5: Cabri Express.

2.2.3 Shapes 3D Geometry Learning & Drawing

Shapes 3D Geometry Learning[4] and Shapes 3D Geometry Drawing[5] are two similar applications whose main feature is viewing solids. Both applications are available in a subscription model. One may also buy a perpetual license for the Shapes 3D Geometry Drawing application on the iOS platform. A free demo accessible from the browser exists for both applications. Unfortunately, the author of the thesis was unable to run the Shapes 3D Geometry Drawing demo due to an error present on the website.

Shapes 3D Geometry Learning

Shapes 3D Geometry Learning is available on all major desktop and mobile operating systems. The target group of the application is students of grades 4-6. The application allows the user to select from a set of predefined solids. When a solid is selected, the user can view its edges, faces, and vertices. The solid can be unfolded it into a net. The user may also build the net themself, using the solid's faces. The net can be printed and subsequently cut out of the paper, to create a paper net. The paper net can be glued to obtain a paper version of the solid displayed in the application. The user interface of Shapes 3D Geometry Learning is visible in figure 2.6.

[4] `https://shapes.learnteachexplore.com/shapes-3d-geometry-learning/`
[5] `https://shapes.learnteachexplore.com/shapes-3d-geometry-drawing/`

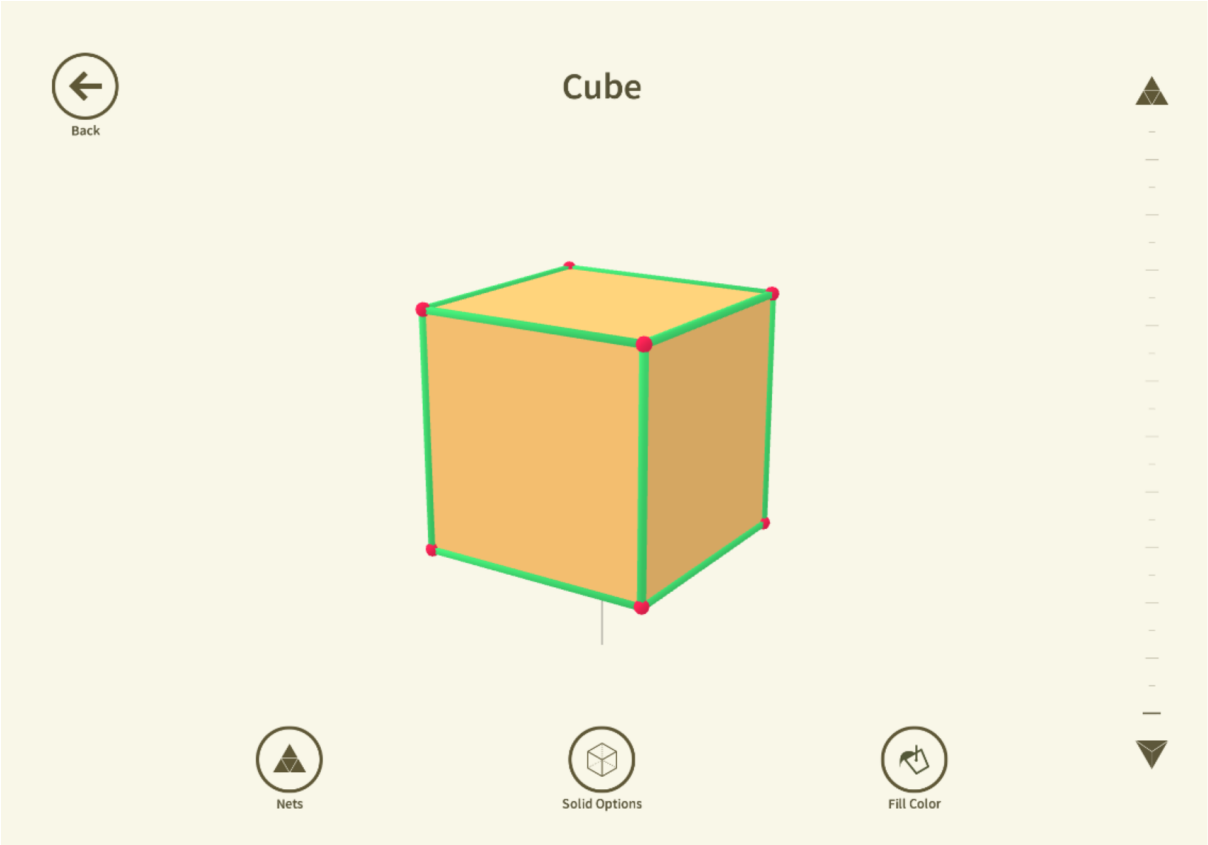


Figure 2.6: Shapes 3D Geometry Learning.

Shapes 3D Geometry Drawing

Shapes 3D Geometry Drawing is available only on the iOS operating system. The target group of the application is students of grades 7-8. It has a user interface and application structure similar to Shapes 3D Geometry Learning. The main differences is the tools available when a solid is selected. Shapes 3D Geometry Drawing allows the user to place line segments and cross sections between points on the solid's faces. The user interface of Shapes 3D Geometry Drawing, along with the error that appears,, is visible in figure 2.7.

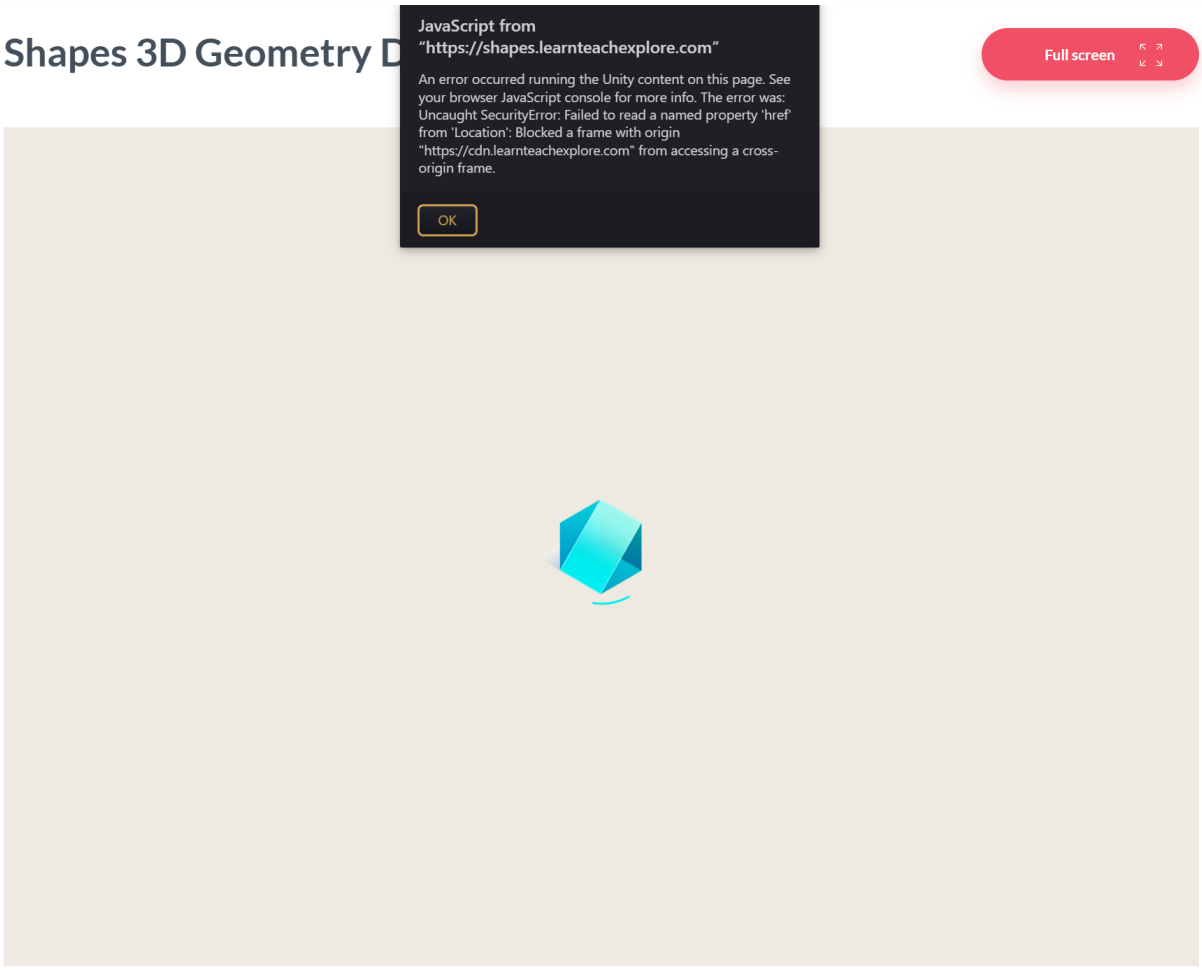


Figure 2.7: Shapes 3D Geometry Drawing. An error occurs when trying to run the demo.

Summary of Shapes 3D applications

Both applications contain many features which help visualize and explore 3D geometry. The applications lack features related to calculation of solids' characteristics like area and volume. They also do not allow the user to create their own solid and only give access to predefined solids. The predefined solids cannot be manipulated by the user, e.g the scale of the solid cannot be changed.

Chapter 3

Requirements and tools

This chapter defines the requirements for the developed application. The requirements describe both the functionality provided to the user and the properties the application must satisfy. They serve as a basis for the design and implementation of the application.

3.1 Functional requirements

Functional requirements describe the behavior and capabilities of the application from the user's perspective. They specify what the application is expected to do, which features it must provide, and how the user can interact with the system. In the context of this thesis, the functional requirements focus primarily on the visualization, manipulation, and educational use of three-dimensional geometric objects. The functional requirements of the application are:

Visualization of 3D objects The application will be able to display three-dimensional geometric solids, such as cubes, prisms, spheres, and cylinders. The displayed objects will be rendered in a three-dimensional scene that allows the user to clearly perceive their shape and spatial properties. The camera will support rotation around the scene and zooming, enabling observation of objects from different angles and distances.

Inspection of solids The application will allow the user to inspect individual components of displayed solids. When hovering over a solid or its elements, such as vertices or edges, the corresponding object will be visually highlighted to indicate that it is selected. Upon clicking a selected object, an informational label will be displayed. The label will present relevant geometric information, for example the coordinates of a vertex or the length of an edge.

Manipulation of solids The application will allow the user to modify parameters related to geometric tasks and objects. Any changes in parameters will be immediately reflected in the three-dimensional view, providing visual feedback. The application

will include a module for creating and manipulating custom polyhedrons, allowing the user to define their shape and structure.

Educational support The application will include a task module containing a set of predefined stereometry problems with varying levels of difficulty. Each task will be divided into individual steps. Each step will focus on the computation of a specific part of the solution. In addition, the application will provide a playground module that allows users to freely experiment with three-dimensional geometry. Within this module, users will be able to create polyhedrons by specifying a list of vertices, save the created polyhedrons to files, and later load them to restore the saved state.

User interface The application will provide an intuitive graphical user interface. The application will support mouse and keyboard interaction. User interface elements will include tooltips and visual cues to help users understand the available controls and functionality.

3.2 Non-functional requirements

Non-functional requirements describe the quality attributes and constraints of the application. They do not define specific features, but instead specify how well the application should perform and under what conditions it should operate. These requirements ensure that the application is reliable and suitable for its intended educational purpose. The non-functional requirements of the application are:

Usability The user interface will be designed to be easy to use and understand. Users will not be required to have prior experience with 3D modeling software in order to effectively use the application. The interface will aim to minimize unnecessary complexity and cognitive load.

Performance The application will provide smooth interaction on hardware capable of running similar three-dimensional educational applications. Camera movement, object manipulation, and user interaction will not cause noticeable stuttering or delays during normal operation.

Reliability The application will operate reliably during runtime and will not crash under normal usage conditions. Invalid or unexpected user input will be handled gracefully without causing application failure.

Correctness The application will correctly compute geometric properties of 3D objects, such as lengths, areas, and volumes.

Maintainability The application source code will be published as open-source. The internal structure of the application will support future extension. Extension will be possible in the form of new stereometry tasks, new geometric objects, or additional features.

Security The application will not require internet access after installation. Files created or used by the application can be manually edited without compromising system security or application stability.

Educational suitability The application will be suitable for use both in classroom environments and for individual study at home. The provided tasks will correspond to typical problems found in school-level stereometry curricula and support the learning process.

3.3 Used tools

3.3.1 Godot Engine

Godot[1] is a cross-platform, free and open-source (FOSS) game engine. It is designed to create 2D and 3D games mainly for the PC (personal computer), mobile, and web platforms. Despite its main focus being the creation of games, it can be used to develop other types of software.

The engine itself is written mostly in C++. The primary language used in Godot to create software is GDScript. Other than GDScript, C# and C++ are also officially supported. Aside from the officially supported languages, support for other languages is provided by community extensions. Some of the community-supported languages are Rust, Swift, and Java.

Godot introduces the concept of *nodes*. A node is the basic building block of the application. A node encapsulates a specific functionality related to 2D or 3D graphics, user interface, etc. Nodes are organized in a tree structure. Each node can have child nodes. This allows for the creation of complex behaviors through the composition of simple elements.

A *scene* is a reusable composition of nodes. Scenes are typically used to represent logical parts of the application, such as user interface screens or 3D objects. A scene can be instantiated multiple times and can be composed of other scenes.

Through the use of nodes and scenes, Godot enables modularity and maintainability. This model is well suited for application which require interactive user interfaces and dynamic scene management.

[1] `https://godotengine.org`

3.3.2 Neovim

Neovim[2] is a modern and open-source text editor that runs in the terminal. Neovim was used as the primary editor for the application's source code.

Neovim provides features which include syntax highlighting and code completion. Plugins add additional functionality which increases the speed of editing, refactoring, and navigation in the codebase.

The support for Godot projects in Neovim is enabled by the Language Server Protocol (LSP) [7]. The Godot editor runs an LSP server process to which Neovim connects. This provides features like syntax highlighting and code completion for GDScript in Neovim.

3.3.3 Git and GitHub

Git[3] is a FOSS distributed version control system [2]. It is used to track changes in files during software development. It allows developers to track the complete history of changes, revert to previous versions, and manage development in a team.

In this project, Git was used to manage project files during the development. It allowed for tracking of changes and easy rollback in situations where new changes introduced issues. Access to the history of file modifications made locating the sources of bugs in code easier.

GitHub[4] is an online platform which hosts Git repositories and provides additional tools for project management. It was used to provide a remote backup of the project and allow access to the project from multiple devices.

[2] `https://neovim.io`
[3] `https://git-scm.com`
[4] `https://github.com`

Chapter 4

External specification

4.1 System requirements

A PC (personal computer) is required to run the application. Versions of the application were prepared for the Windows, macOS, and Linux operating systems. Due to hardware limitations, only the version for the Windows operating system was tested. The application requires approximately 100 MB of storage space. The application was tested on a laptop device with the following software and hardware configuration:

CPU	AMD Ryzen 7 5800H
GPU	NVIDIA GeForce RTX 3050
RAM	16 GB
Operating system	Windows 11

The application performed smoothly on the device. Due to limited access to other hardware, the application could not be tested on low-end devices to establish minimum system requirements accurately. The Godot Engine documentation specifies the following as the minimum system requirements for a simple 2D or 3D project [12]:

CPU	x86_32 CPU with SSE2 support, x86_64 CPU with SSE4.2 support, ARMv8 CPU
GPU	Integrated graphics with full Vulkan 1.0 support, Metal 3 support (macOS) or Direct3D 12 (12_0 feature level) support (Windows)
RAM	2 GB
Operating system	Windows 10, macOS 10.15, Linux distribution released after 2018

4.2 Installation

The application does not require installation on any supported platform. It is launched directly from the appropriate executable.

The application was tested only on the Windows operating system. As such, the instructions for other operating systems may contain errors.

4.2.1 Windows

For the Windows operating system, the application is provided as a single executable file. To start the application, the user should run the **GeoApp.exe** executable.

Depending on system security settings, Windows may display a warning about an unknown publisher. In this case, the user can confirm the warning to start the application. The executable is compiled for the x86-64 architecture.

4.2.2 Linux

For the Linux operating system, the application is provided as a single executable file. The user should ensure that the file has execution permissions. This can be done through the file manager or by executing the following command in the terminal:

```
chmod +x GeoApp.x86_64
```

When the file has execution permissions, the application can be started by running the **GeoApp.x86_64** executable. The executable is compiled for the x86-64 architecture.

4.2.3 macOS

On the macOS operating system, the application is provided in a `.zip` archive containing an application bundle. First, the user should extract the provided **GeoApp.zip** file. The user may then optionally move the extracted **GeoApp.app** file to the **Applications** folder. To start the application, the user should run the **GeoApp.app** application.

Since the application is not digitally signed, a security warning may appear when the application is started for the first time. When this happens, the user can start the application by right-clicking the **GeoApp.app** file, selecting **Open**, and confirming the dialog. Alternatively, the application can be allowed in the system settings under **Privacy & Security**.

4.3 User manual

The application consists of several modules, namely *task exploration*, *task solving*, and the *playground*. Each module provides different ways for the user to learn stereometry.

4.3.1 Main menu

Upon starting the application, the user sees the main menu. The main menu is visible in figure 4.1. The main menu is a central hub from which the rest of the application can be accessed. The main menu contains several button used to navigate the application:

- the **Explore tasks** button presents the user with the task exploration module,
- the **Solve tasks** button presents the user with the task solving module,
- the **Playground** button presents the user with the playground module,
- the **Settings** button presents the user with the settings screen,
- the **Quit** button closes the application.

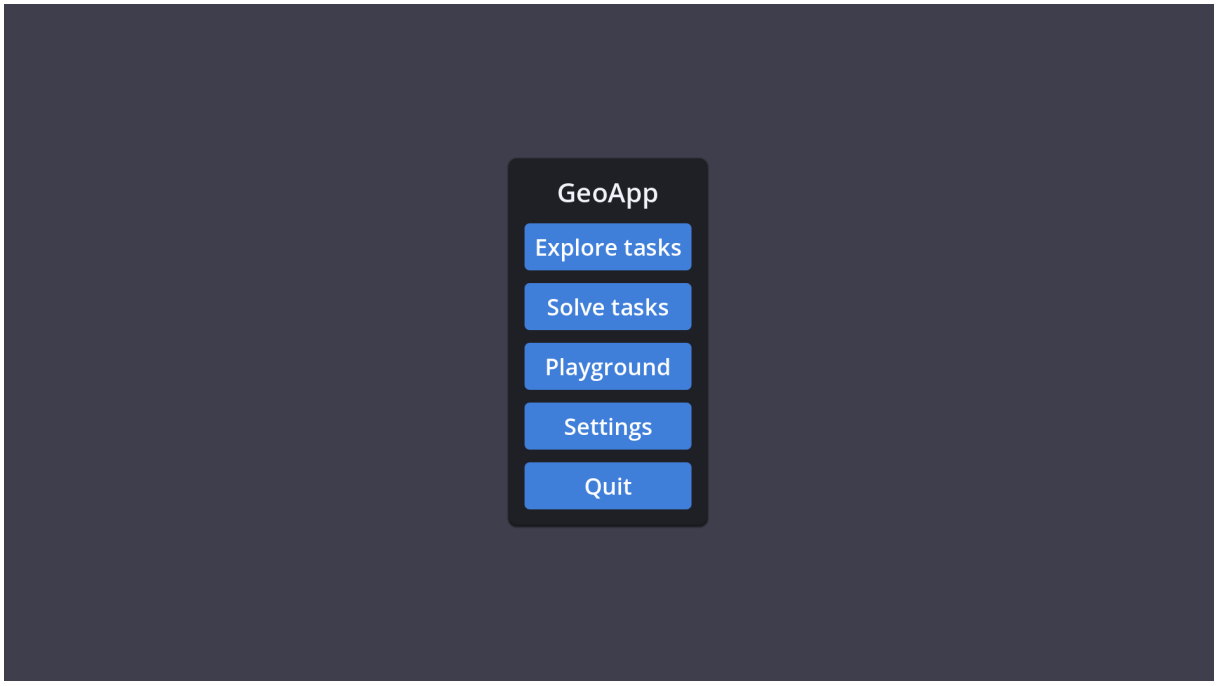


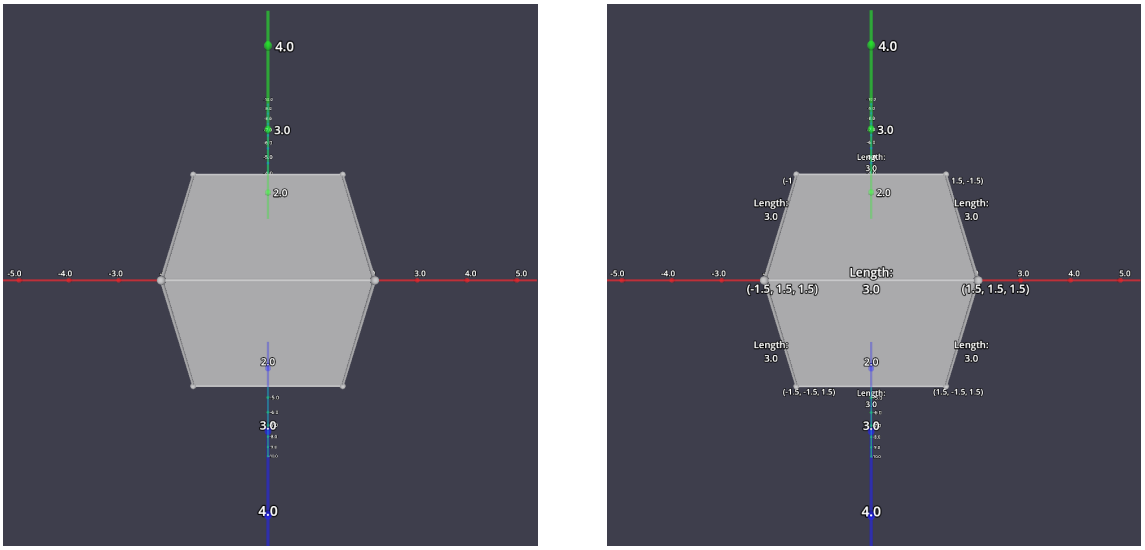
Figure 4.1: The main menu of the application.

4.3.2 The 3D view

The 3D view is a part of the task exploration, task solving, and playground modules. When the 3D view is visible, the user can interact with it.

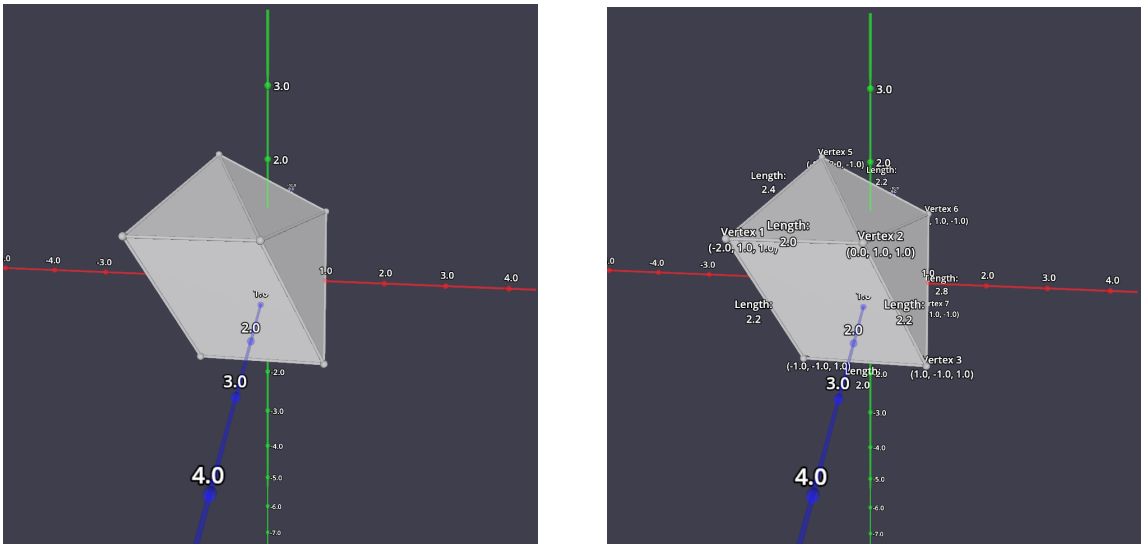
The camera of the 3D view can be moved. The camera orbits the center of the coordinate system and always looks looks in its direction. The camera can be rotated up, down, left, and right with the **W**, **S**, **A**, and **D** keys respectively. The camera can be zoomed in with the **E** key and zoomed out with the **Q** key.

When a solid is loaded in the 3D view, the user can inspect some of its properties. When an object (the solid, one of its vertices or one of its edges) is hovered overed by the mouse cursor, it becomes red. When a hovered vertex or edge is clicked using the left mouse button, an informational label appears (see figures 4.2 and 4.3). The informational label for a vertex contains its position. In the playground module, the label also contains the name of the vertex. The informational label for an edge contains its length. When an object with a visible informational label is clicked using the left mouse button, the informational label disappears.



(a) Invisible informational labels. (b) Visible informational labels.

Figure 4.2: The 3D view as seen in the task exploration and task solving modules.



(a) Invisible informational labels. (b) Visible informational labels.

Figure 4.3: The 3D view as seen in the playground module.

4.3.3 The navigation bar

Each screen, except for the main menu, features a navigation bar. The navigation bar appears above the main content of the current screen. The navigation bar contains five buttons:

- the **Main menu** button presents the user with the main menu,
- the **Explore tasks** button presents the user with the task exploration screen,
- the **Solve tasks** button presents the user with the task selection screen,
- the **Playground** button presents the user with the playground screen,
- the button with the gear icon presents the user with the settings screen.

The button corresponding to the current screen is disabled. Figures 4.4 and 4.5 show the navigation bar.

Figure 4.4: The navigation bar as seen in the task exploration screen.

Figure 4.5: The navigation bar as seen in the settings screen.

4.3.4 Task exploration

The purpose of the task exploration module is to teach the user about the tasks available in the application. Ordered from left to right, the user interface features:

- A list of all task, grouped by difficulty. The tasks can also be filtered by their name.
- All data describing the task. Each parameter can be changed. All UI (user interface) elements will update dynamically when a parameter is changed.
- The description of the task.
- A list of all of the steps required to solve the tasks. For each step, there is:
  - The description of what should be calculated for the step.
  - A help button. The button shows a hint for its respective step when pressed.

– A spin box[1] which displays the correct answer for its respective step.

- A view of the selected task.

The default, empty state of the task exploration module is visible in figure 4.6.

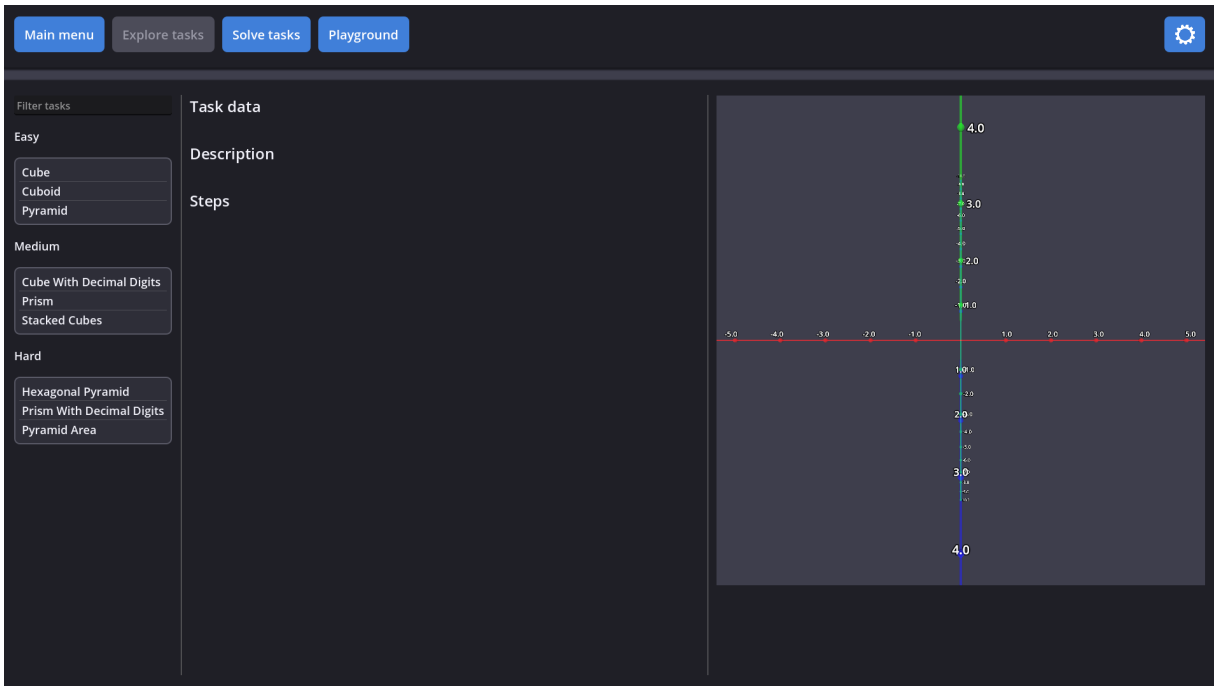


Figure 4.6: Empty task exploration module.

The tasks can be filtered by name. Figure 4.7 shows the task list being filtered. When the "cub" string is used as the filter, only the following tasks are shown: *Cube*, *Cuboid*, *Cube With Decimal Digits*, *Stacked Cubes*.

Figure 4.8 shows the task exploration module with a task selected. The task features a cube. The task is divided into three steps. The first steps requires the user to calculate the area of a single face of the cube. The second step requires the user to calculate the cube's total area. The third step requires the user to calculate the cube's volume.

[1]A spin box or spinner is a user interface element which allows the user to adjust a value using a text box or up and down arrows.

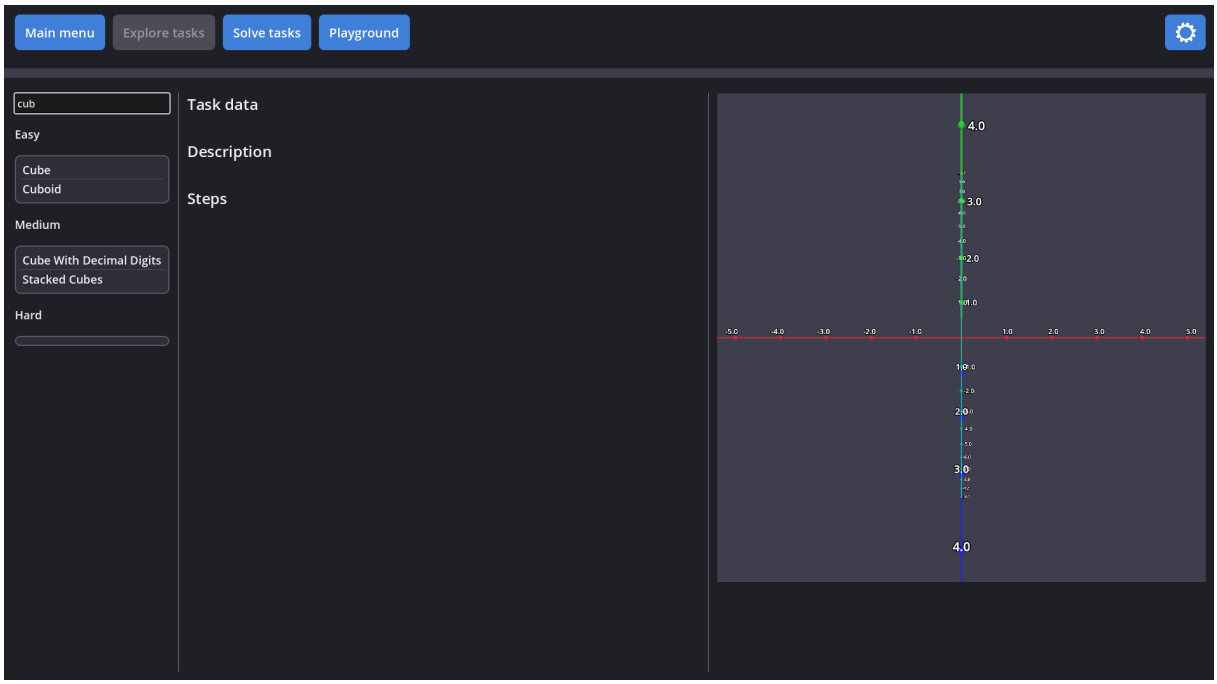


Figure 4.7: Empty task exploration module. The "cub" task name filter is applied.

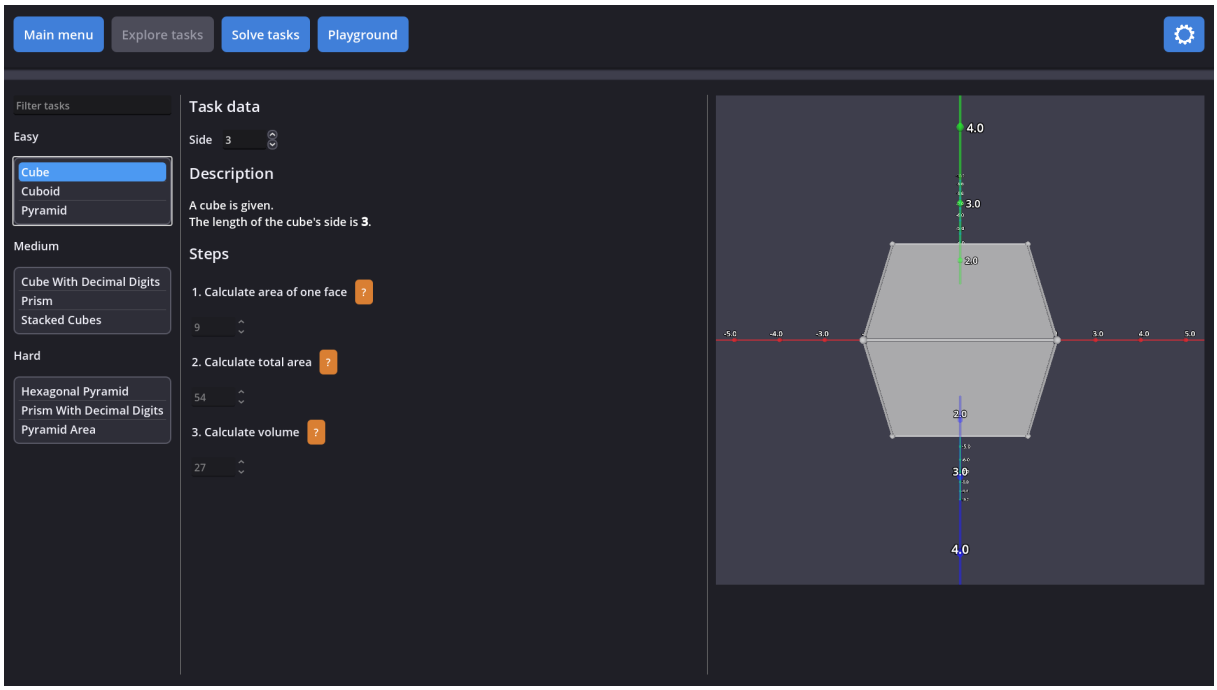


Figure 4.8: Task exploration module. The *Cube* task is selected.

Figure 4.9 shows the task exploration module with a task selected. The selected task is the same as the one in figure 4.8. The *Side* task parameter has a different value as compared to the one in figure 4.8. As a result, the cube in the 3D view appears bigger and correct answers for each step are different.

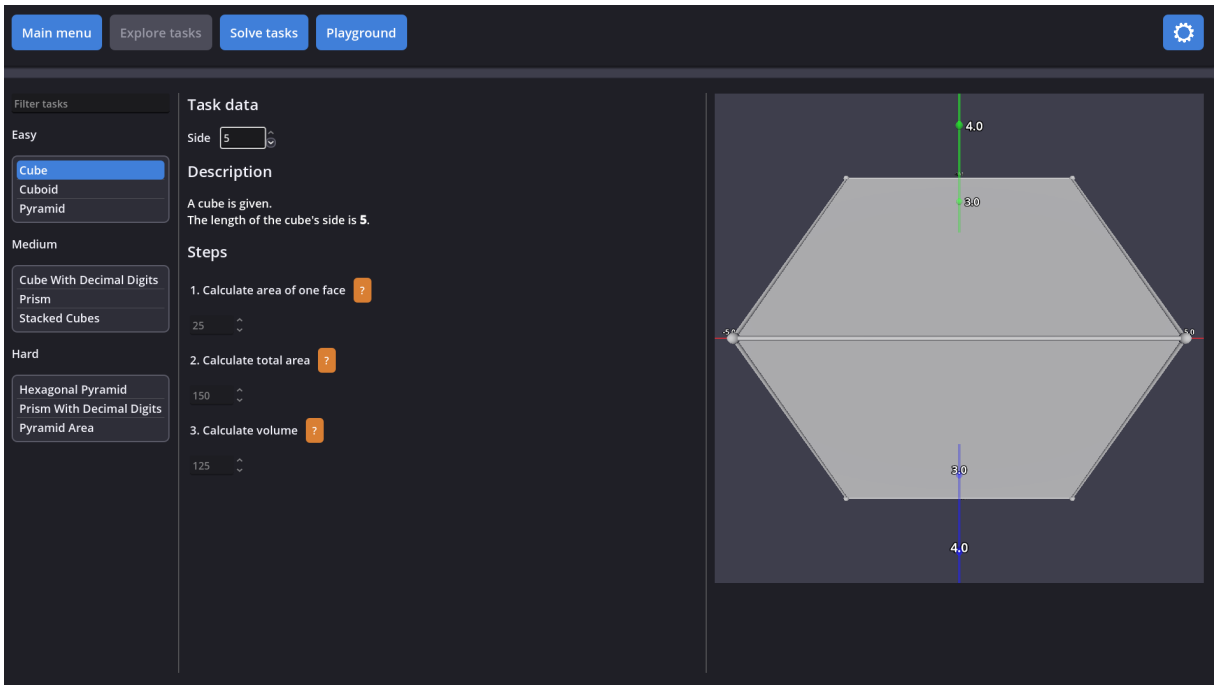


Figure 4.9: Task exploration module. The *Cube* task is selected. The task has a different parameter value as compared to the one visible in figure 4.8

Figure 4.10 shows the task exploration module with a task selected. The task features a pyramid with a hexagonal base. The task is divided into three steps. The first steps requires the user to calculate the area of a single triangle of the hexagonal base. The second step requires the user to calculate the area of the pyramid's base. The third step requires the user to calculate the pyramid's volume.

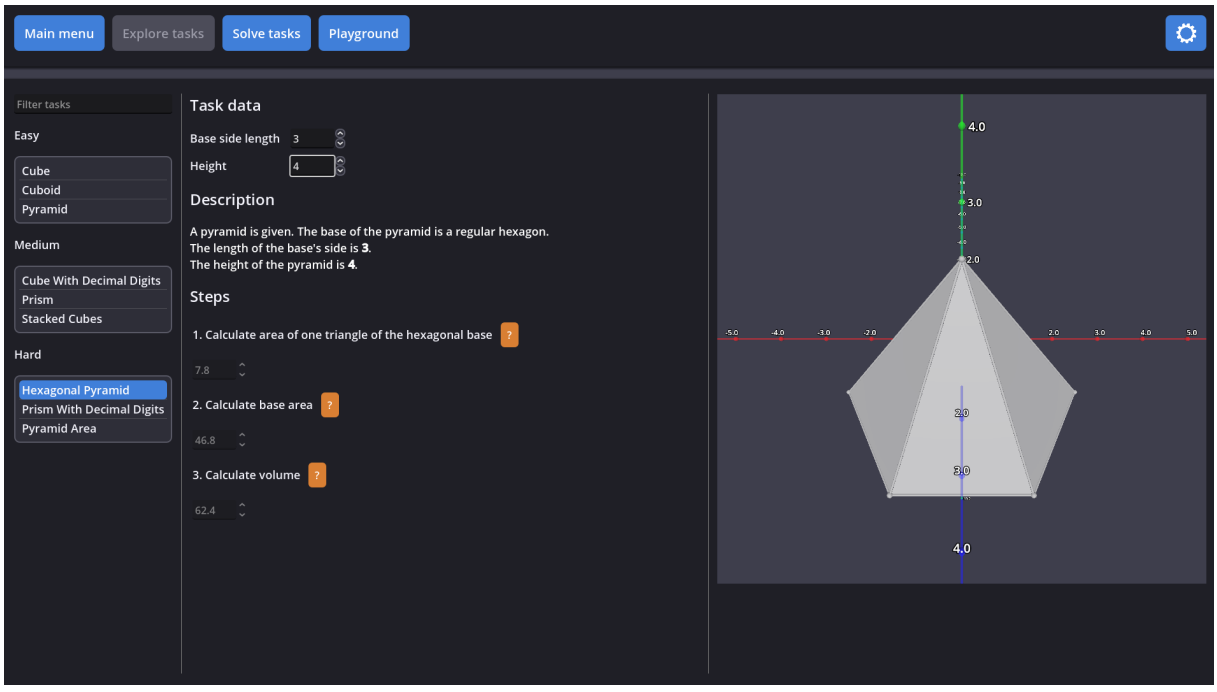


Figure 4.10: Task exploration module. The *Hexagonal Pyramid* task is selected.

Figure 4.11 shows the task exploration module whose selected task features a prism. In figure 4.12, the selected task features two cubes, with one stacked on top of the other. For both tasks, the list of steps is too long to fit on the screen. In this case, a scroll bar appears to make it possible to access the steps that are currently not visible. Figure 4.13 shows the same task as figure 4.12 with the step list scroll bar in the lowest position.

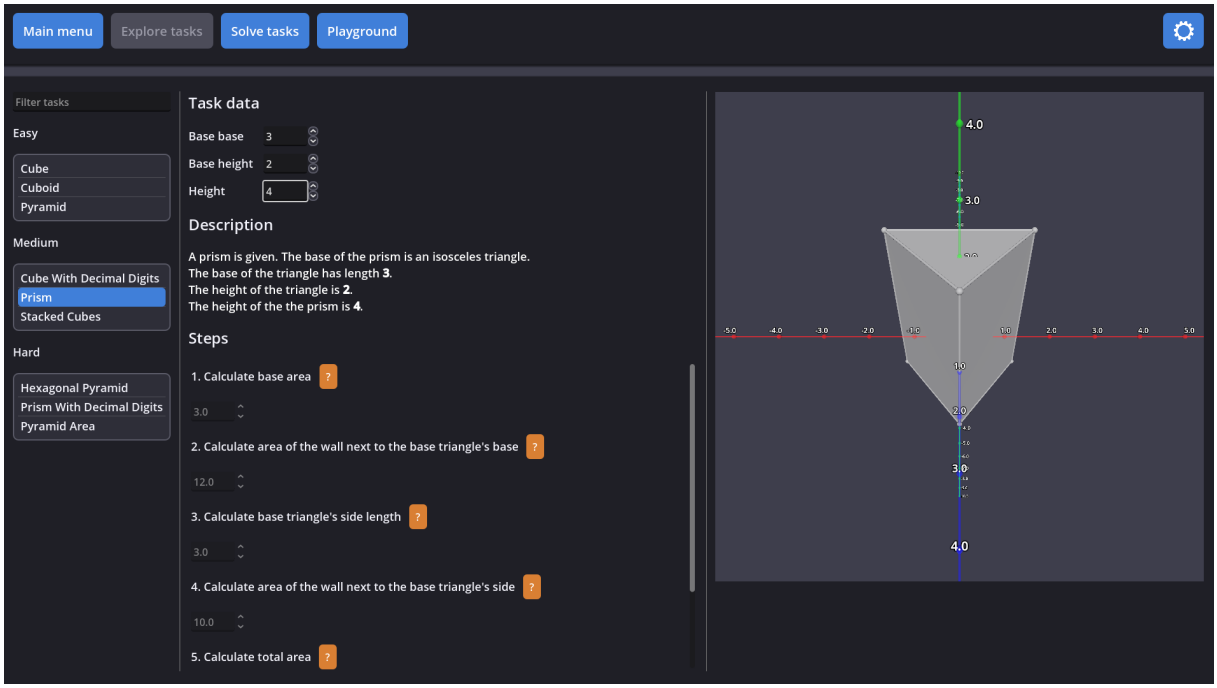


Figure 4.11: Task exploration module. The *Prism* task is selected.

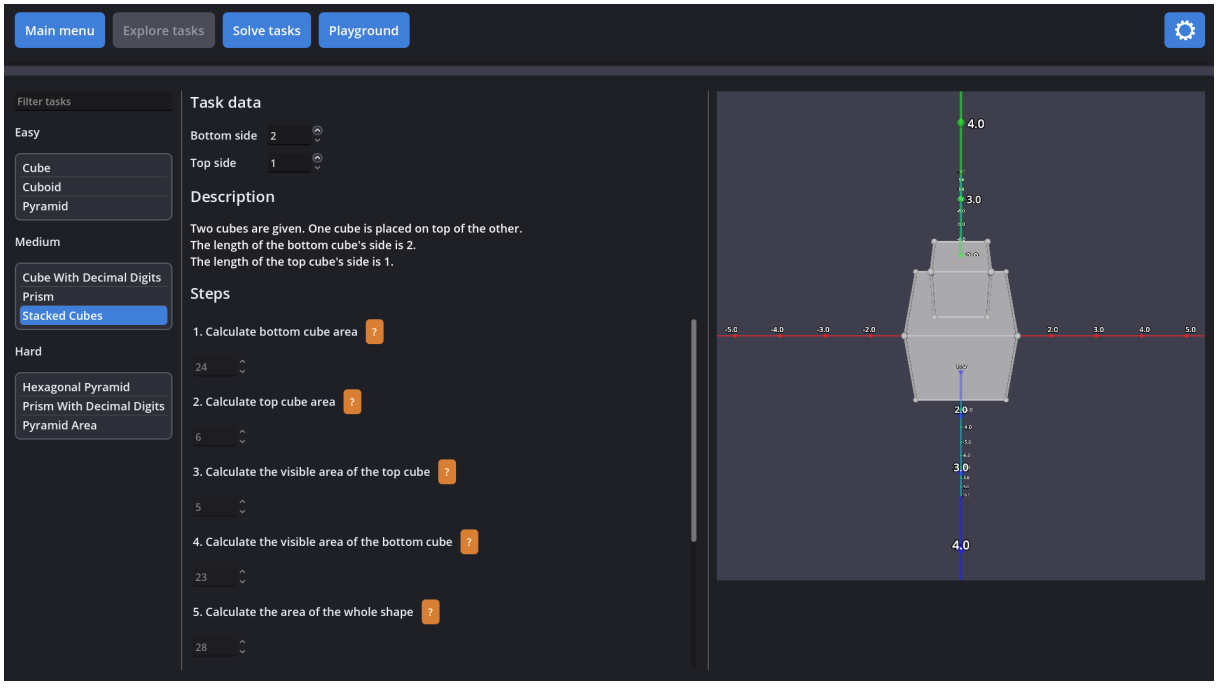


Figure 4.12: Task exploration module. The *Stacked Cubes* task is selected.

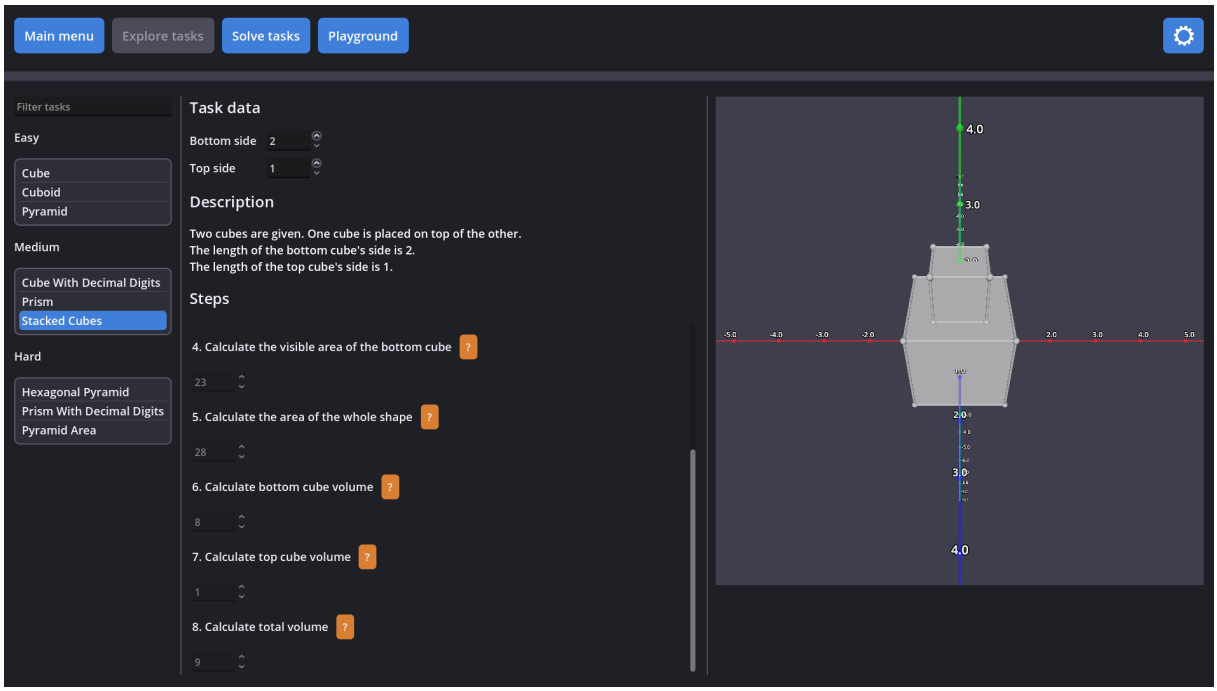


Figure 4.13: Task exploration module. The *Stacked Cubes* task is selected. The step list is scrolled to the bottom.

4.3.5 Task solving

The purpose of the task solving module is to test the user's knowledge and skills gained in the task exploration module. The task solving module is split into two parts.

4.3.5.1 Task selection

The first part is task selection. Here, the user selects the task they wish to solve. Figures 4.14 and 4.15 show the state of the task selection screen before the user selects a task. Figure 4.16 shows the state of the task selection screen when a task is selected.

To select a task, the user must first select a difficulty. When a difficulty is selected, all available tasks with this difficulty are shown. Then, the user should select their desired task and press the **Start** button. The **Start** button can be pressed only when a task is selected. After pressing the **Start** button, the user is presented with the main part of the task solving module.

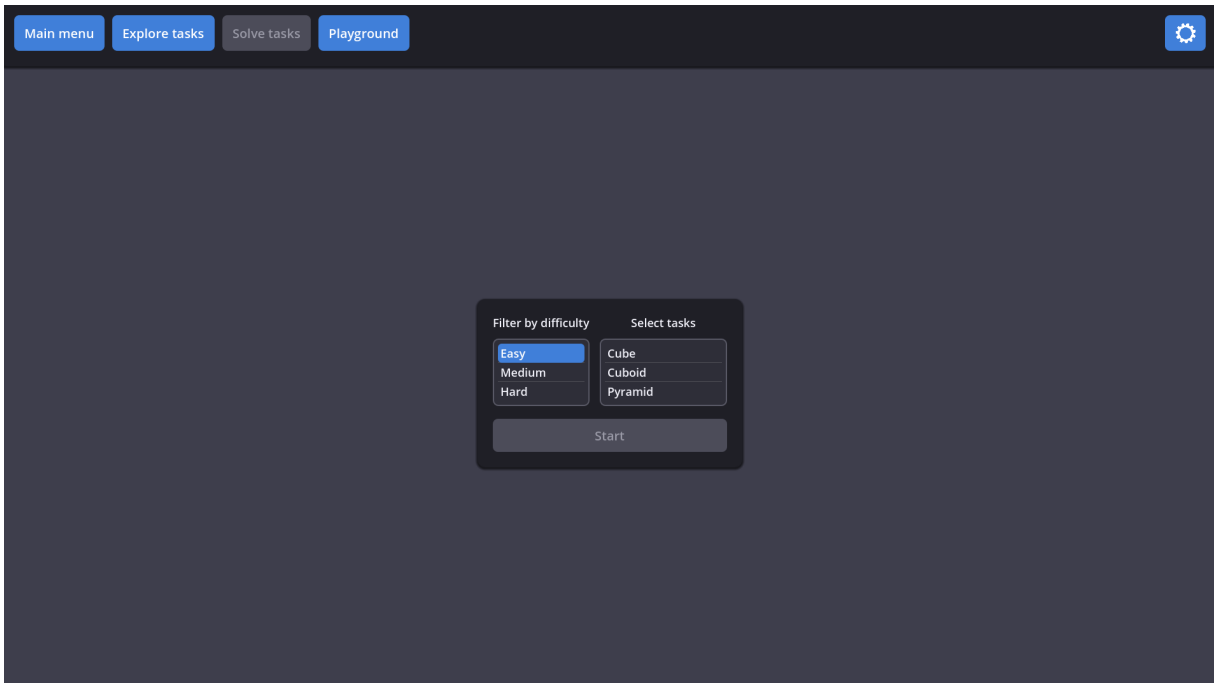


Figure 4.14: The task selection screen. The easy difficulty is selected. No task is selected.

4.3.5.2 The main part of the task solving module

Figure 4.17 shows the main part of the task solving module. The *Cube* task is loaded.

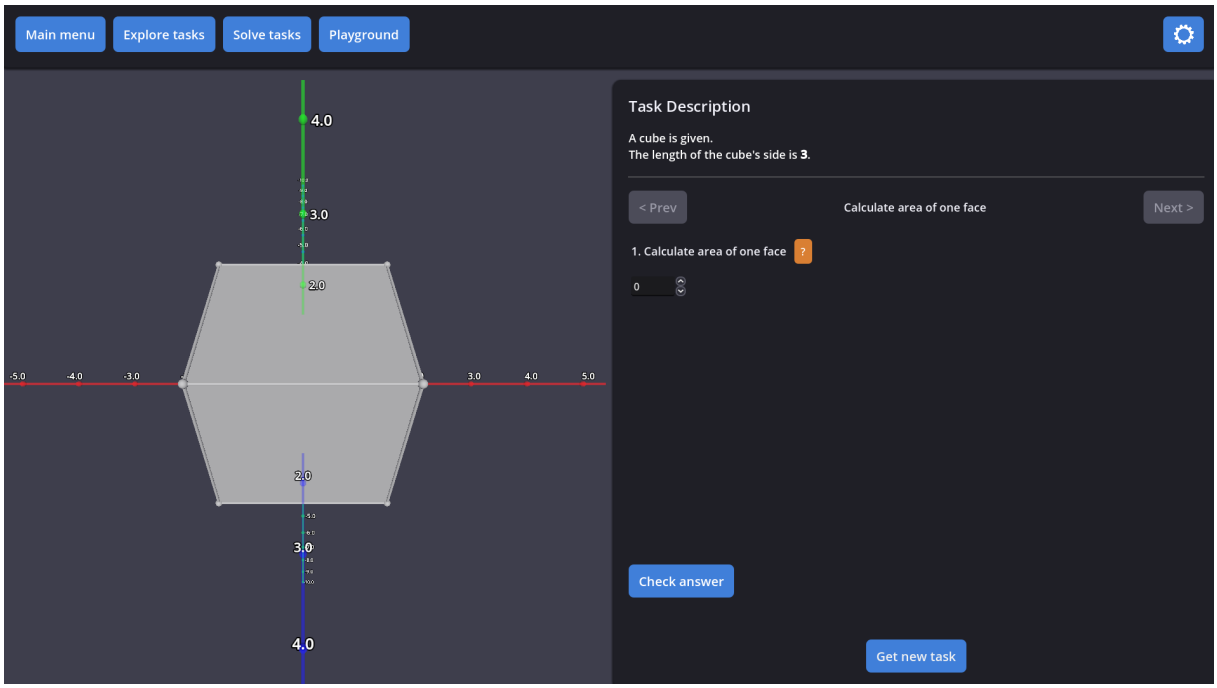


Figure 4.17: The task solving screen. The *Cube* task is loaded.

Inside the main part of the task solving module, the user can find:

- the 3D view showing the current task,

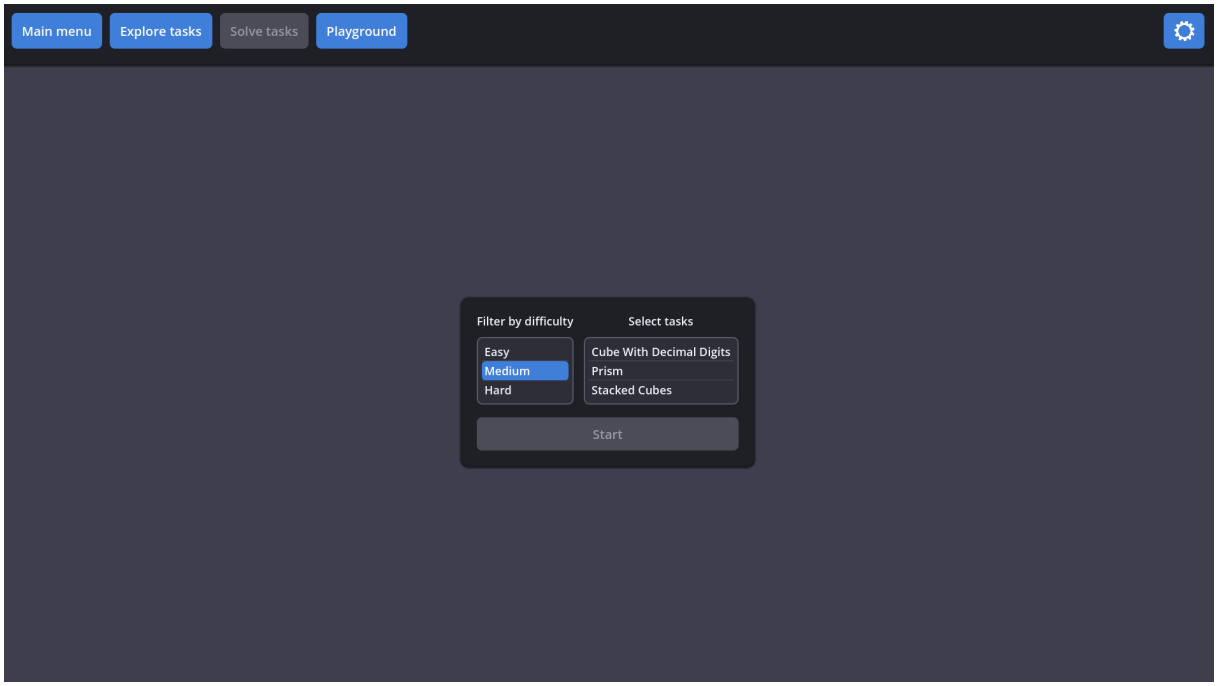


Figure 4.15: The task selection screen. The medium difficulty is selected. No task is selected.

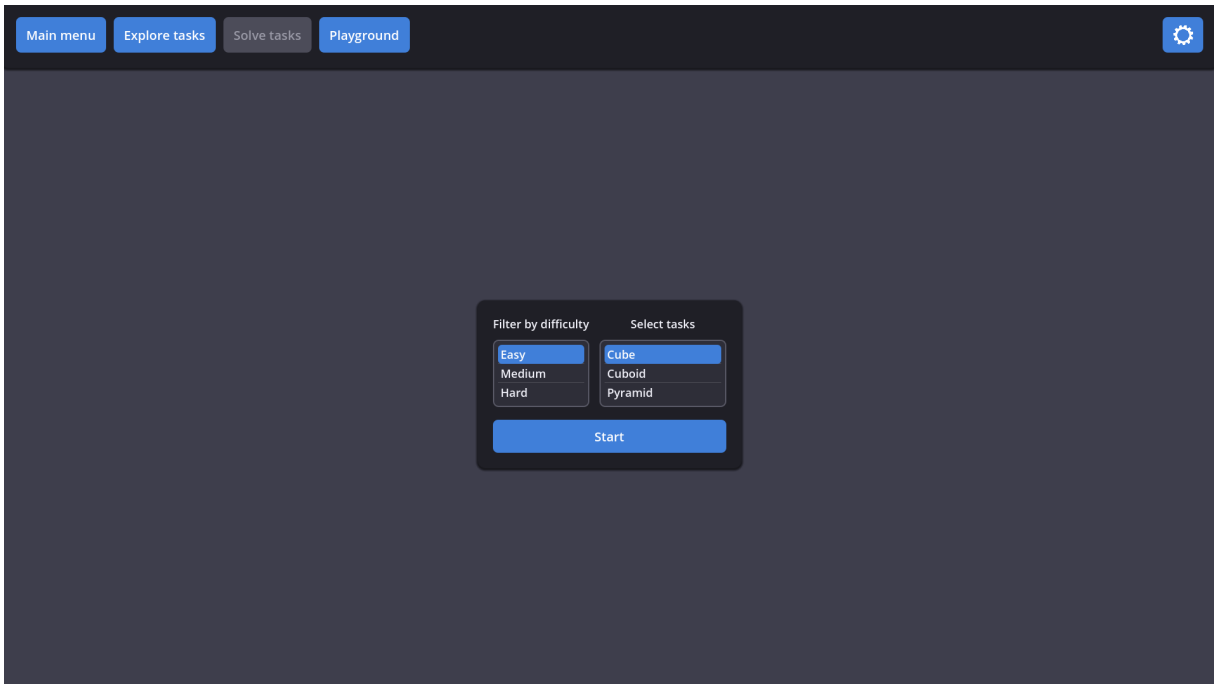


Figure 4.16: The task selection screen. The *Cube* task is selected.

- the description of the task,
- the step navigation buttons,
- the *step list*,
- the **Check answer** button,
- the **Get new task** button.

The *step list* is the part of the UI responsible for displaying the task's steps. The step list (marked in red) and its parts are marked in figure 4.18. Each element of the step list consists of three parts:

- The step description (marked in green). It states what should be calculated for the step.
- The help button (marked in blue). The button shows a hint for its respective step when pressed.
- The answer spin box (marked in yellow). The user places their answer for the respective step here.

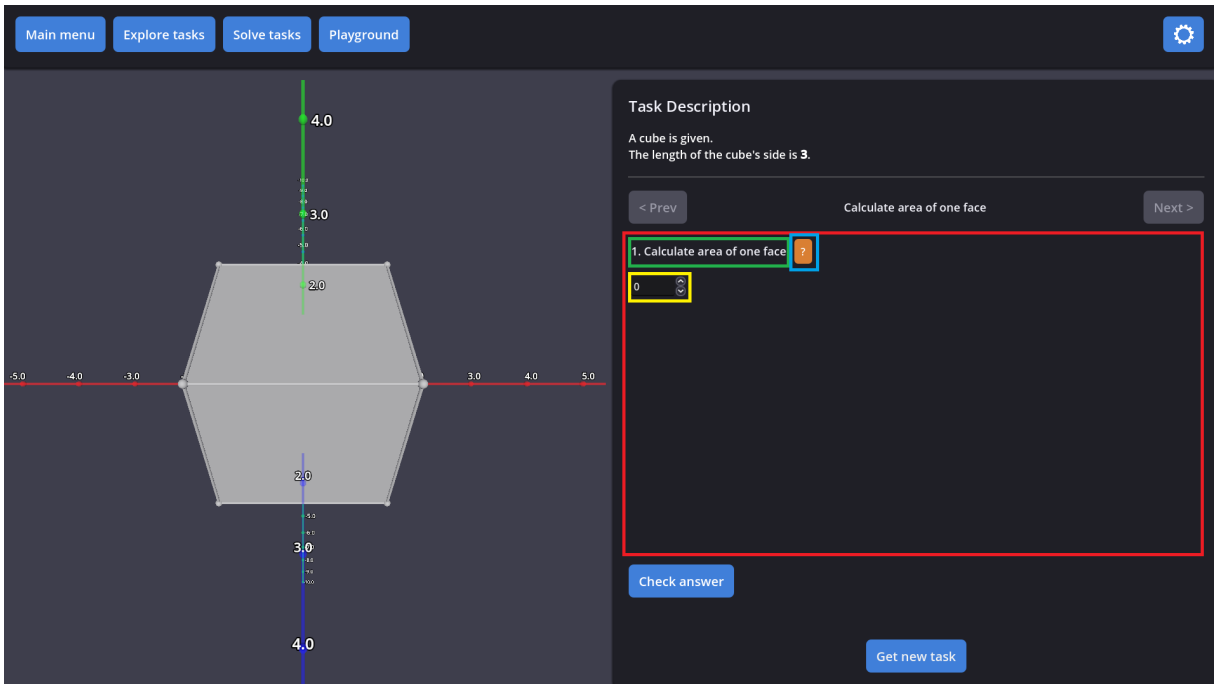


Figure 4.18: The task solving screen. The step list is marked with a red rectangle.

If the user enters the correct value and presses the **Check answer** button, a green checkmark symbol appears to the right side of the spin box (see figure 4.19). If the user enters an incorrect value, a red X symbol appears instead (see figure 4.20).

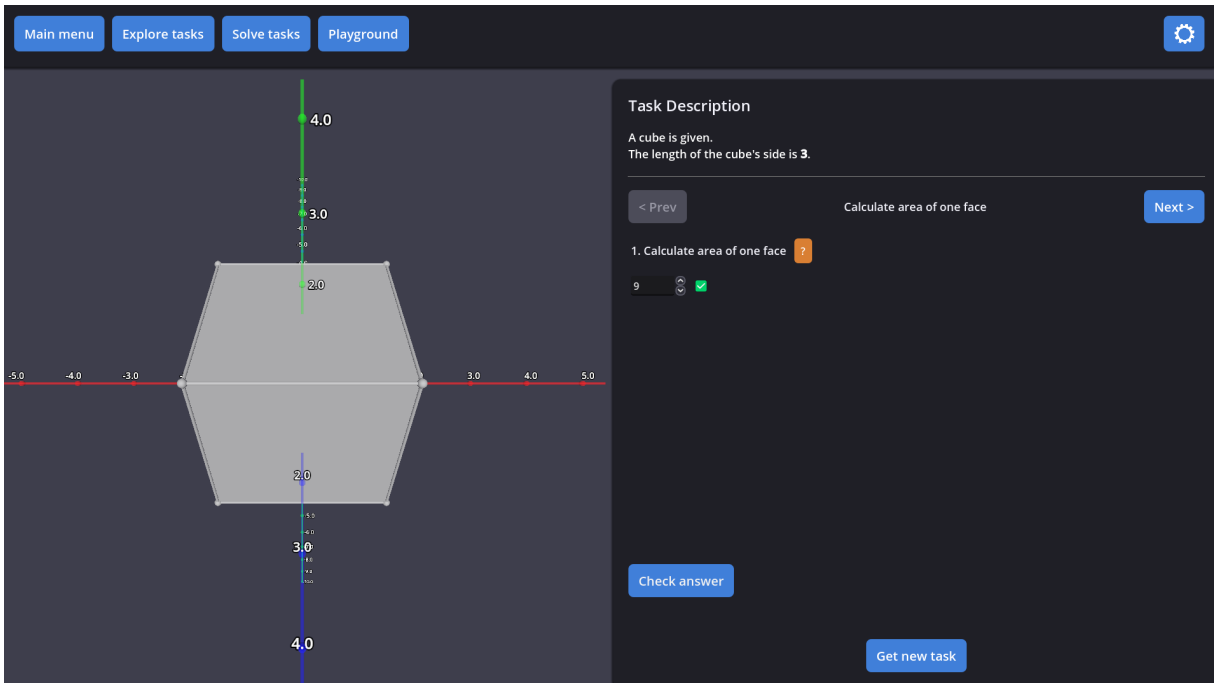


Figure 4.19: A green checkmark appears next to the spin box when the correct answer is provided.

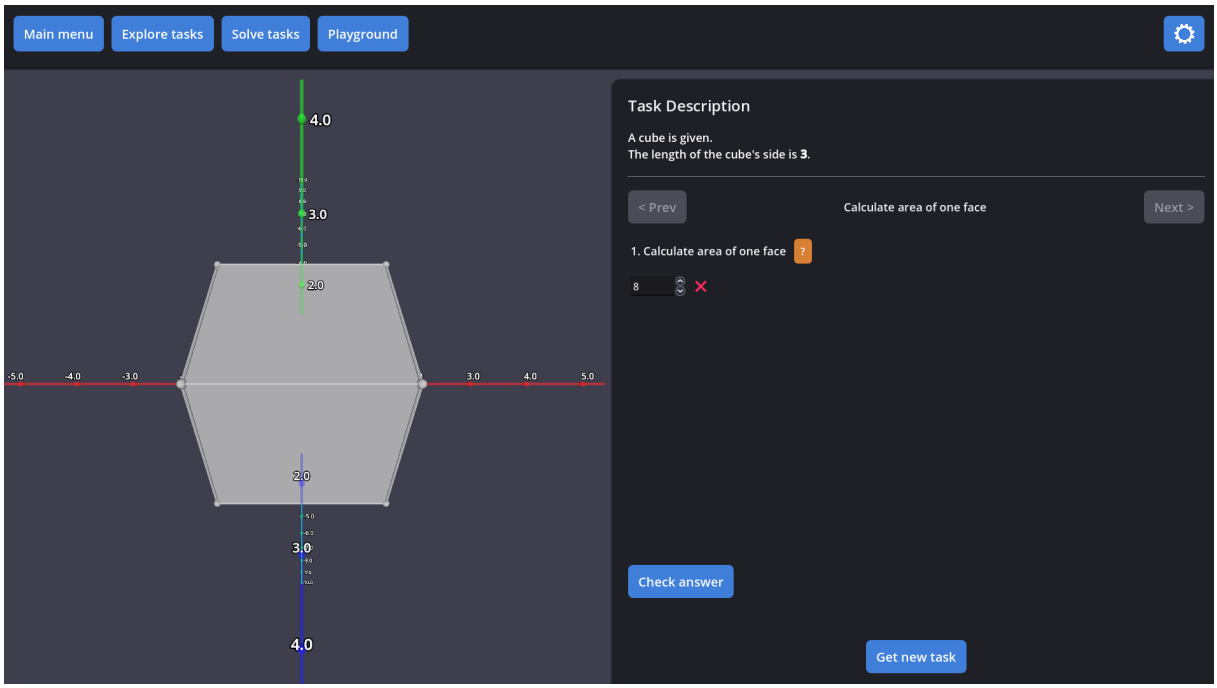


Figure 4.20: A red X symbol appears next to the spin box when an incorrect answer is provided.

After providing the correct answer for the current step, the user can move to the next step using the **Next >** button (see figure 4.21).

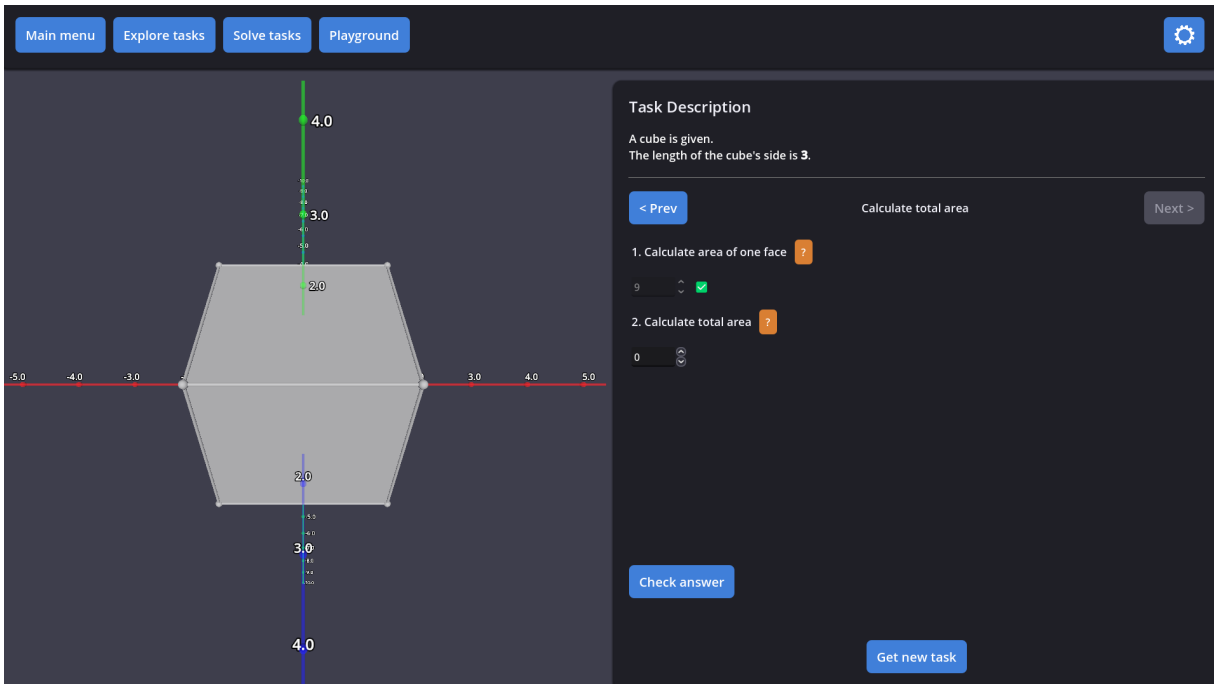


Figure 4.21: The state of the task solving screen after moving to the second step.

The user should repeat this process for each step of the task to complete the task. When the user enters the correct answer for the last step of the task, the task is completed. When a task is completed, a label with the text "Task complete!" appears under the **Check answer** button. The state of the task solving screen after the task is completed is visible in figure 4.22.

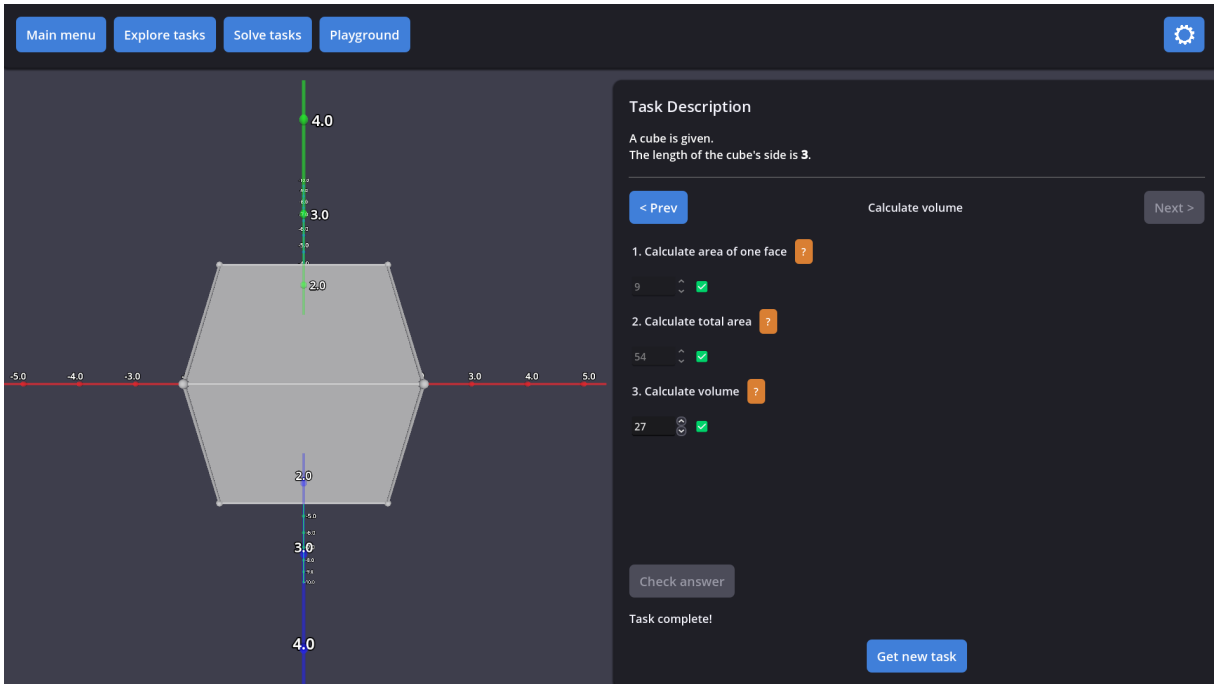


Figure 4.22: The state of the task solving screen after the task is completed.

Pressing the **Get new task** button provides the user with a new task of the same

type with randomized parameter values. The state of the task solving screen after pressing the **Get new task** button is visible in figure 4.23. This state is the same as the one seen in figure 4.17 with the exception of tasks parameter values.

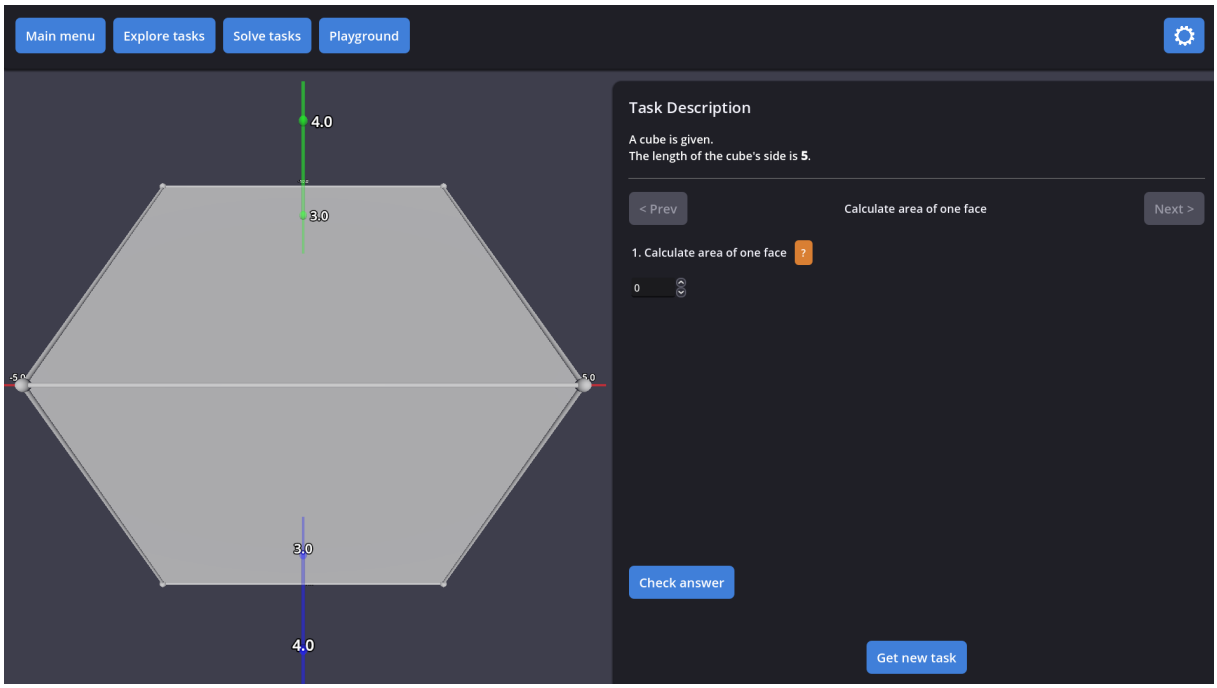


Figure 4.23: The state of the task solving screen after pressing the **Get new task** button.

4.3.6 Playground

The playground allows the user to create an arbitrary polyhedron and inspect its properties. The playground starts empty by default and the user can add vertices from which the polyhedron is created. Figure 4.24 shows the default state of the playground module.

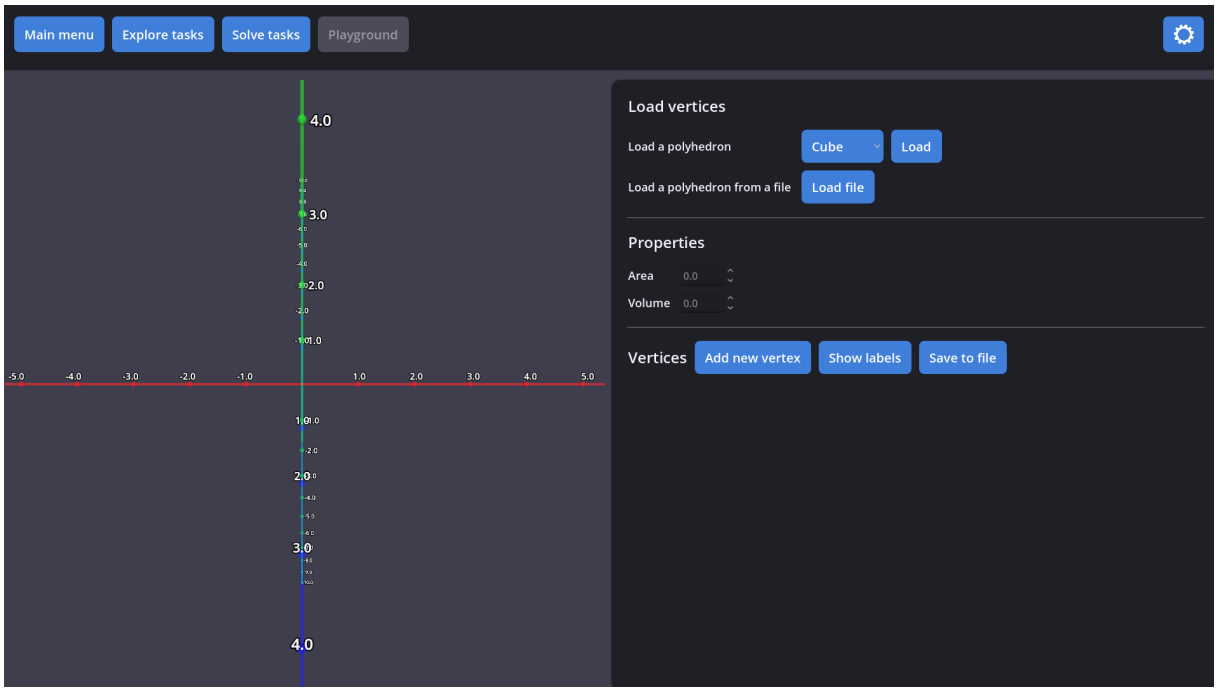


Figure 4.24: The default state of the playground module.

4.3.6.1 Adding vertices

The vertices can be added in several ways. The easiest way to add vertices is to load a predefined polyhedron. To do this, the user should select one of the predefined polyhedrons next to the "Load a polyhedron" label. Next, they should press the **Load** button to the right. This loads a predefined set of vertices. Figure 4.25 shows the state of the playground with the predefined prism vertices loaded.

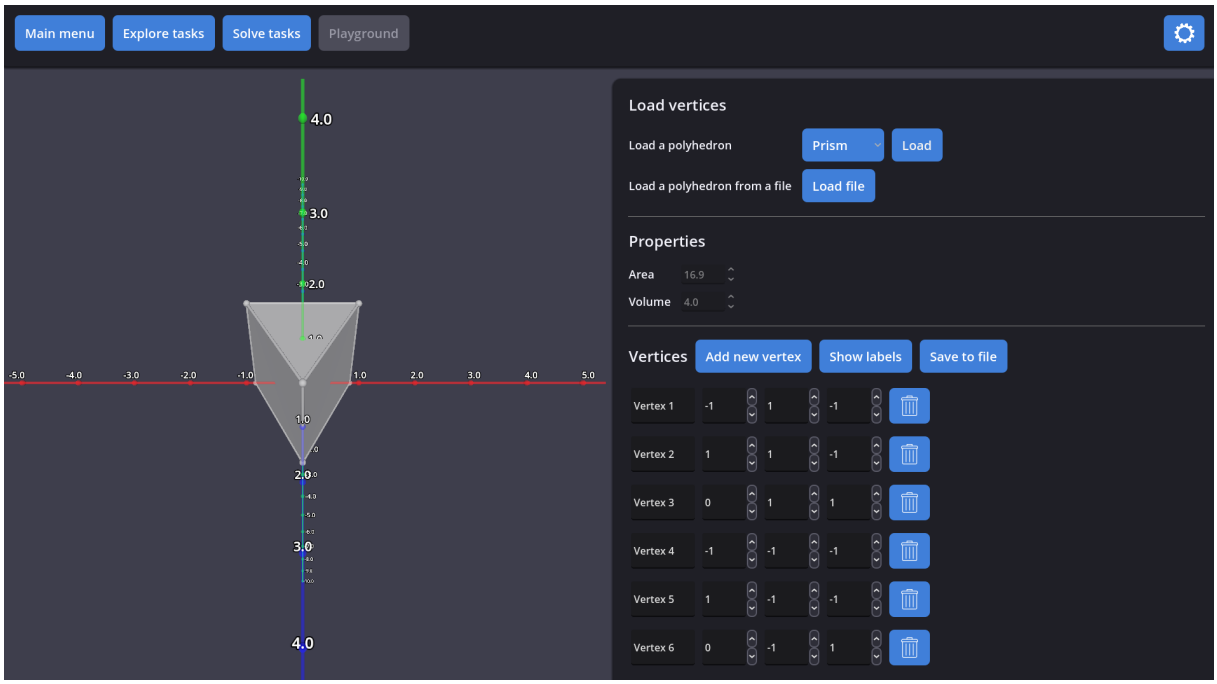


Figure 4.25: The predefined prism vertices loaded in the playground.

Another way to load a set of vertices is to load them from a file. To do this, the user should press the **Load file** button next to the "Load a polyhedron from a file" label. A system file dialog should appear, expecting the user to select a file with the `.poly` extension. No example `.poly` files are provided by default. A detailed explanation of `.poly` files is provided in section 4.3.6.4.

The last way to add a vertex is to use the **Add new vertex** button next to the "Vertices" label. This adds a new vertex in the origin of the coordinate system. When adding vertices this way, the polyhedron may become invalid. Polyhedron validity is explained in section 4.3.6.3. Figure 4.26 shows what happens after adding a new vertex to predefined prism vertices (as seen in figure 4.25).

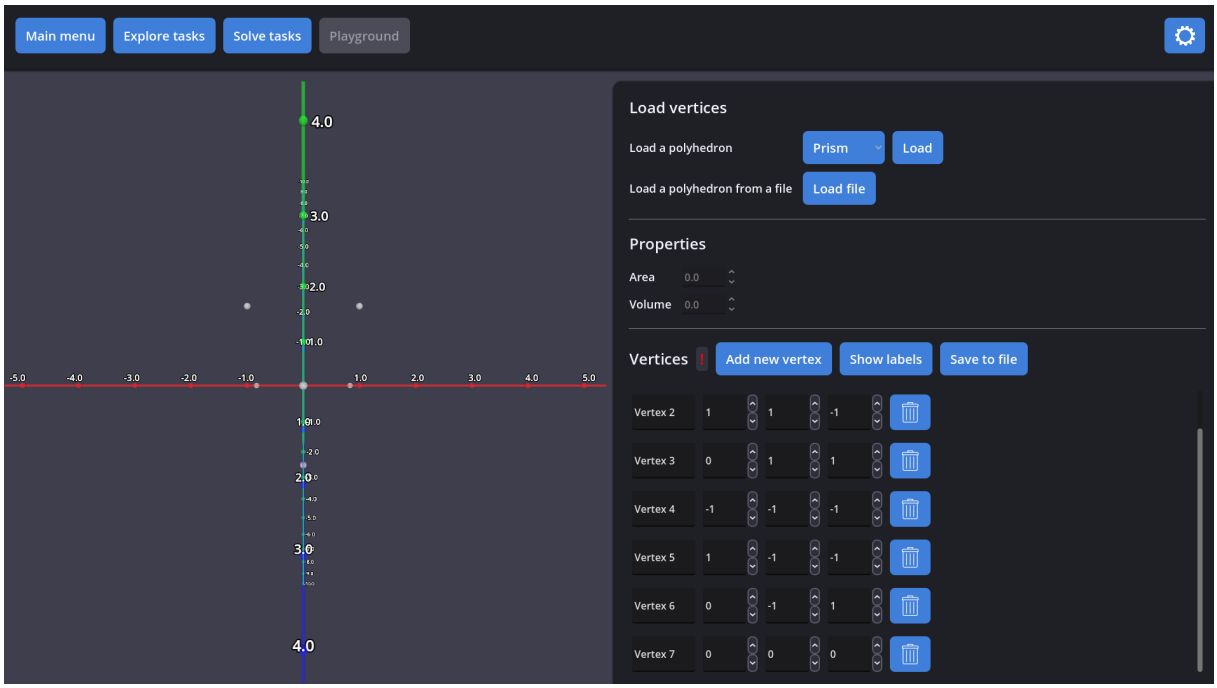


Figure 4.26: A new vertex is added to predefined prism vertices. The polyhedron becomes invalid.

One way to solve this issue is to move the newly added vertex (called "Vertex 7" in this case) such that a valid polyhedron can be created. The position (0, 2, 0) allows for the creation of a polyhedron. Figure 4.27 shows the polyhedron created by moving the new vertex.

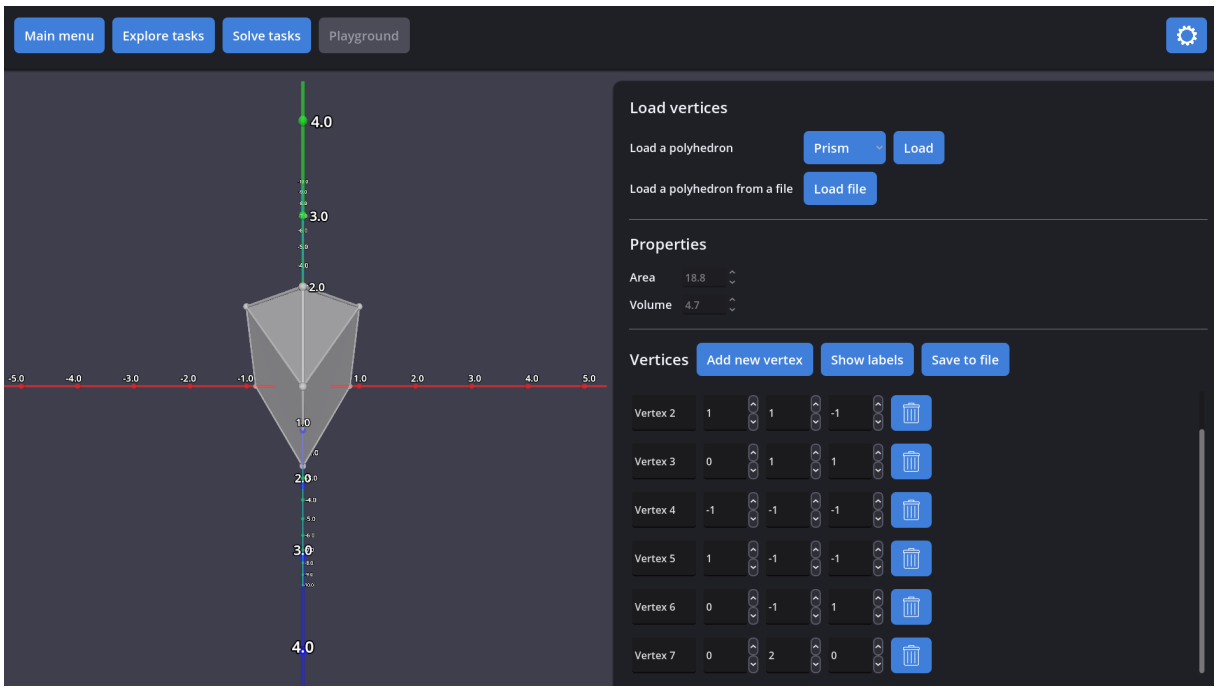


Figure 4.27: The position of the new vertex is set to (0, 2, 0). The polyhedron becomes valid.

4.3.6.2 The vertex list

The vertex list is the part of the UI where all of the added vertices are shown. Each polyhedron vertex corresponds to one entry in the vertex list. Each entry consists of:

- The editable name of the vertex. It is used for display only. It appears in the vertex informational label alongside the coordinates.
- Three spin boxes corresponding to the coordinates of the vertex. The first, second, and third spin box correspond to the x, y, and z coordinate respectively.
- The remove button. The vertex is removed when this button is pressed.
- A red "Duplicate vertex" warning. This warning appears only when there are multiple vertices with the same coordinates.

Changes in the vertex list are immediately reflected in the 3D view.

4.3.6.3 Polyhedron validity

When modifying vertices, the polyhedron may become invalid. The polyhedron is invalid when it cannot be created from the provided vertices. A polyhedron which contains concave faces is considered invalid. Otherwise, it is valid. When the polyhedron is valid:

- the polyhedron in the 3D view is filled, its vertices and edges are shown,
- the properties UI displays the values of the polyhedron's area and volume,
- no warning is shown.

When the polyhedron is invalid:

- only the vertices are shown in the 3D view,
- the properties UI displays 0 for both area and volume,
- a warning in the form of a red exclamation mark is shown next to the "Vertices" label.

It may happen that the user adds two or more vertices with equal coordinates. All such vertices except the first one are marked as duplicate. Duplicate vertices show a red "Duplicate vertex" warning in the vertex list. When there are any duplicate vertices, the polyhedron is considered invalid.

4.3.6.4 Polyhedron files

A polyhedron file is a file that stores all of the data of one polyhedron. Polyhedron files have the `.poly` extension. They can be created in the playground module. A polyhedron file is created with the use of the **Save to file** button near the "Vertices" label. When the **Save to file** button is pressed, a system file dialog appears. The user may save the polyhedron to a file even if the polyhedron is invalid. In the dialog, the user should specify the name of the file and where to save it. After the file is saved, it is ready to be loaded. The instructions on how to load a polyhedron file are provided in section 4.3.6.1.

4.3.7 Settings

The settings screen allows the user to change certain behaviors of the application. Figure 4.28 shows the settings screen.

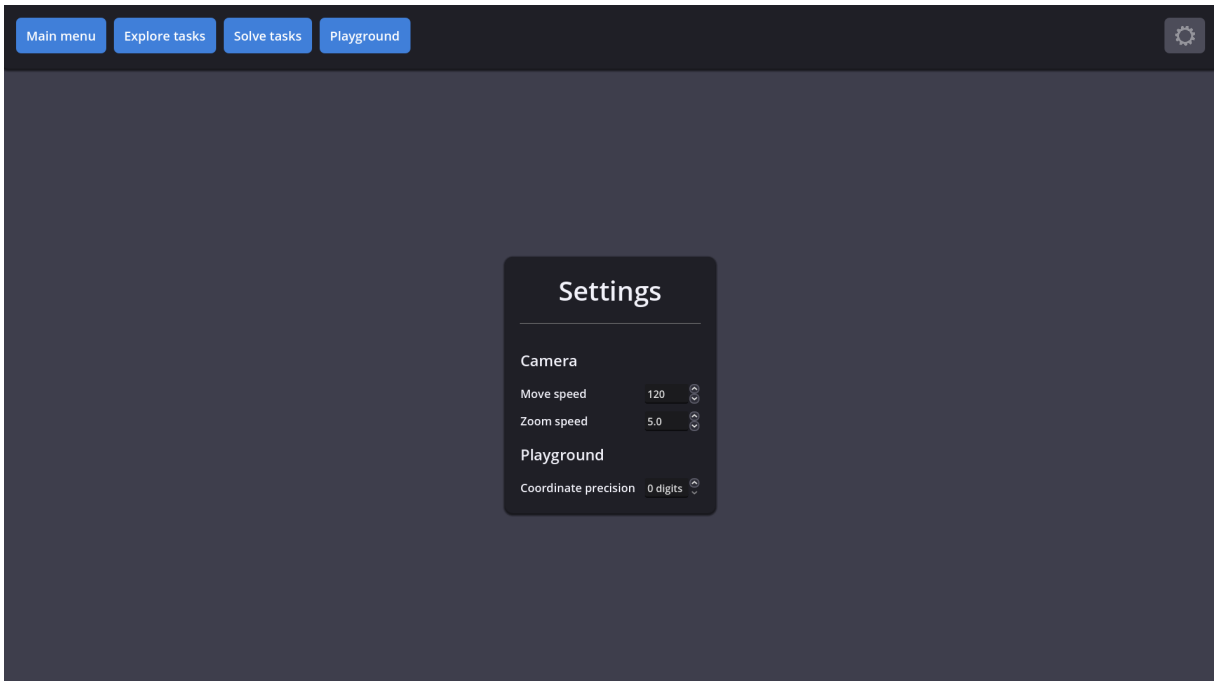


Figure 4.28: The settings screen.

Settings are split into two sections. The *Camera* section contains two settings related to the camera of the 3D view:

- The *Move speed* setting determines the rotation speed. It is expressed in degrees per second.
- The *Zoom speed* setting determines the zoom speed. It is expressed in the 3D view world units per second.

The *Playground* section contains one setting related to the playground module. The *Coordinate precision* setting determines how precisely the vertices can be placed. It is expressed in the number of decimal digits.

The settings are stored in a **settings.cfg** file. The settings are saved to this file each time the user leaves the settings screen provided at least one setting was changed from its previous value. Listing 4.1 shows the content of this file with all settings set to their default values. The location of the **settings.cfg** file is platform-dependent:

Platform	Location
Windows	`%APPDATA%\GeoApp`
Linux	`~/.local/share/GeoApp`
macOS	`~/Library/Application Support/GeoApp`

The user should not modify this file manually. If the file is modified in a way that the application cannot read it correctly, the settings are set to the default values. The file is then saved in the correct format. The same happens when the file is deleted.

If the user manually enters a value outside of the range defined by the setting, the value is constrained to the defined range. The file is not changed and contains the value entered manually.

Listing 4.1: The default content of the **settings.cfg** file.

```
1 [settings]
2
3 "Move speed"=120.0
4 "Zoom speed"=5.0
5 "Coordinate precision"=0
```

Chapter 5

Internal specification

5.1 Nodes and scenes

To understand the structure of a Godot project, one must understand the concept of *nodes* and *scenes*. A node represents a single logical part of the application. For example, a node of the `MeshInstance3D` type (class) represents the mesh of a 3D object. A node may have a script attached to extend its functionality. To be attached to a node, the script must extend the right class. A script attached to a `MeshInstance3D` node should extend the `MeshInstance3D` class or any class that is more general. For `MeshInstance3D`, this could be the general `Node3D` class. A script may be given a class name to create a user-defined class. A node can have multiple children, creating a tree structure.

A scene is a tree of nodes which can be instantiated multiple times. For instance, a simple scene representing a 3D object may be composed of an `Area3D` as the root node with a `CollisionShape3D` and a `MeshInstance3D` as its children. Such a scene scene would represent a 3D object which has a mesh and a collider.

5.2 GeoApp structure

As with any Godot project, the `root` node is the top-level node which contains the entirety of the application. In this project, the root node contains two logical parts. The first and most important part is the `Main` scene. The other part are the *autoloads*. Figure 5.1 shows the complete structure of the application.

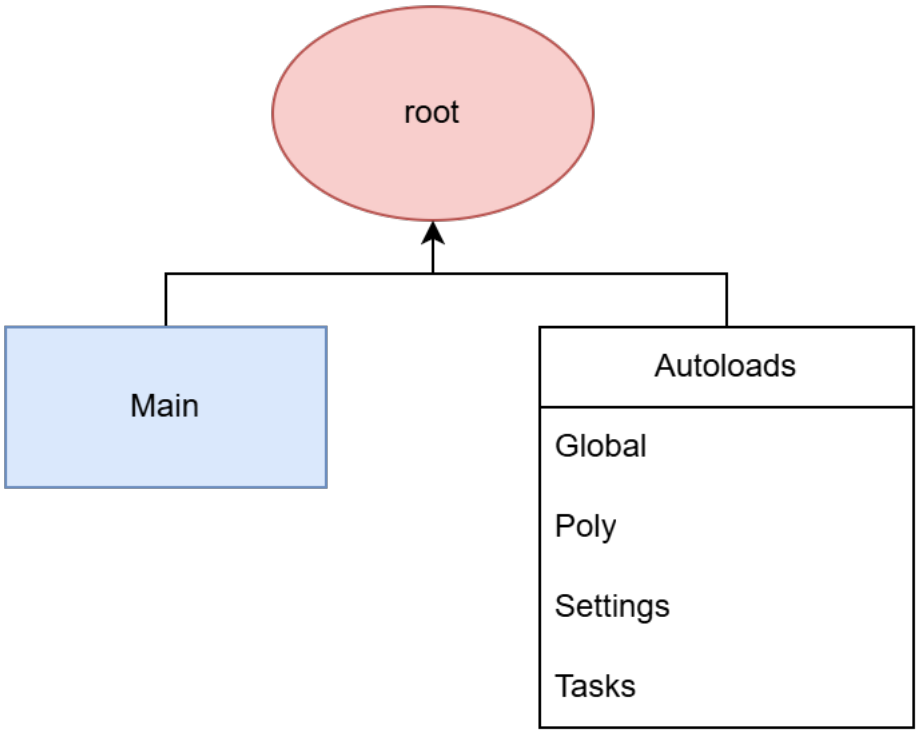


Figure 5.1: The complete application structure starting from the `root` node.

5.3 The Main scene

The `Main` scene represents the whole UI of the application. It is composed of all of the UI screens. One of its children is the background of the application. Figure 5.2 shows the structure of the `Main` scene.

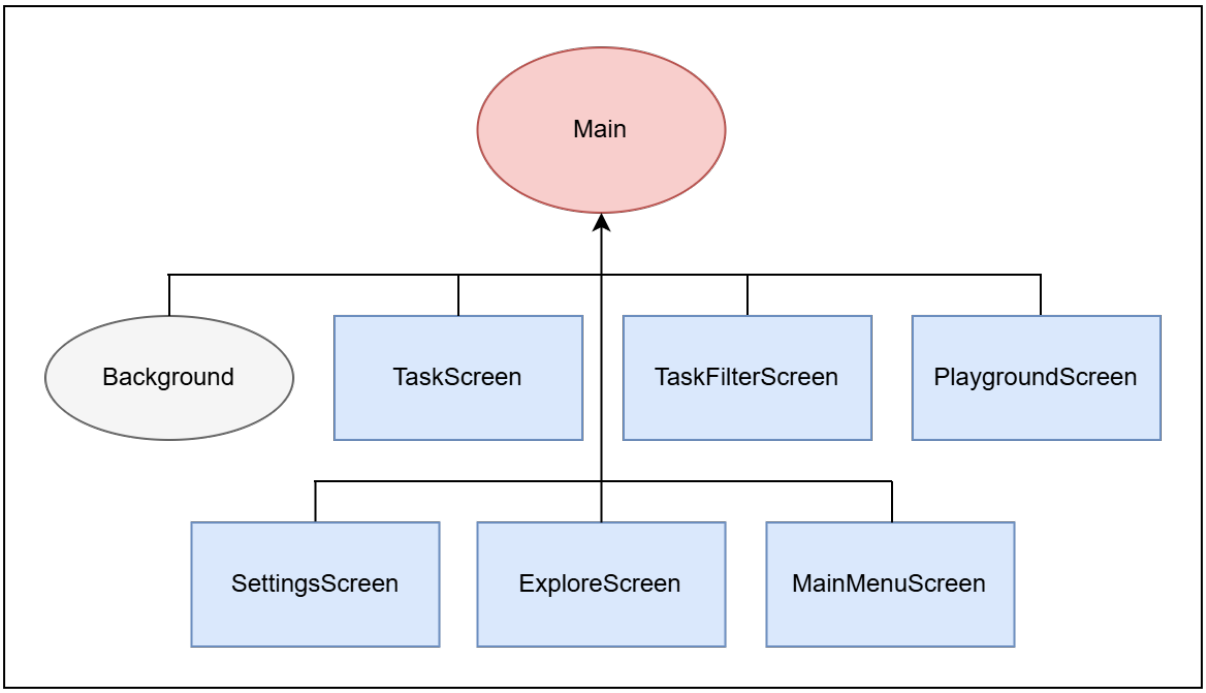


Figure 5.2: The Main scene.

The ellipses represent nodes, while rectangles represent scenes. The `Main` node has a script attached. The script controls transitions between screens. Each screen scene has a script attached. A screen's script is responsible for the internal logic of the screen.

5.4 Autoloads

Autoloads are Godot's implementation of the Singleton design pattern [5]. The Singleton pattern ensures that a class has exactly one instance and provides a global point of access to it. In the application, autoloaded scripts are used to store shared state and functionality that must be accessible from multiple scenes. The autoloads used in GeoApp are:

- `Global` - contains utility functions related to rounding of floating-point values and clearing of UI grids.
- `Poly` - contains the predefined vertices of certain polyhedrons and functions related to polyhedrons.
- `Settings` - contains all of the application's settings and functions for storing and loading them.
- `Tasks` - is responsible for loading all of the tasks.

5.5 Signals

Signals are Godot's implementation of the observer design pattern [4]. The Observer pattern defines a one-to-many dependency between objects, where changes in one object automatically notify and update dependent objects. Signals are used in the application to decouple user interface elements from application logic.

Each node defines a set of signals. These signals can be connected to a script's method. The method is called when the signal is emitted. A signal is emitted when the action associated with it happens. For example, a `Button` node's `pressed` signal is emitted when the button is pressed. The signal may be connected to a `_on_button_pressed` method which will be called whenever the button is pressed. The script containing the `_on_button_pressed` method may be attached to any node, including the button itself.

Signals allow for loose coupling between nodes. A node does not have to know where it is in the scene tree when communication with it is handled using signals.

5.6 Representation of solids

Each solid in the application is represented by a scene. The structure of the scene is similiar to the one in figure 5.3.

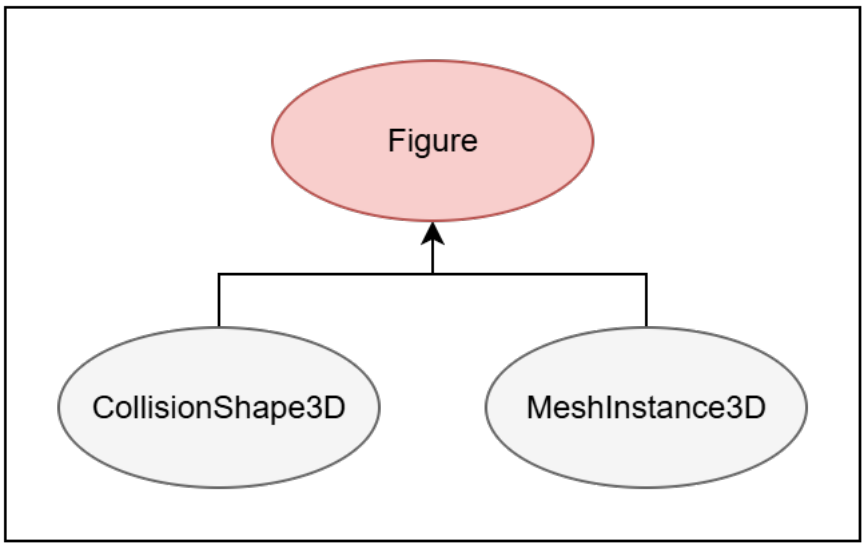


Figure 5.3: A simple 3D object scene. The red node is the root node of the scene.

The difference is that the type of the root node is a class which inherits from the custom `Figure` class. For example, the root node of the cube scene is of the `Cube` type.

5.6.1 Figure

The `Figure` class is the base class of each solid. It:

- stores the mesh and the collision shape,
- handles behavior that is triggered when the solid is hovered,
- contains the `area` and `volume` methods,
- emits signals when the properties of the solid change.

5.6.2 Polyhedron

The `Polyhedron` class extends the `Figure` class. It represents a convex polyhedron. The class handles the additional vertex and edge objects. The class handles the additional objects that are spawned alogside the main body of the polyhedron. Those objects are the spheres on the vertices and cylinders on the edges. The class implements `Figure`'s `area` and `volume` methods to calculate these properties for any polyhedron.

5.6.2.1 Area

The total area of a polyhedron can be calculated as the sum of the areas of all faces. A face of a valid polyhedron[1] is a convex polygon. A convex polygon can be decomposed into a set of triangles. If the polygon is defined a list of vertices lying on a plane, the face can be decomposed by fixing one vertex and creating triangles from pairs of adjacent vertices. The area of each triangle is calculated using the magnitude of the cross product of two edge vectors. For a triangle with vertices v_1, v_2, and v_3, its area A is calculated as:

$$A = \frac{1}{2} \|(v_2 - v_1) \times (v_3 - v_1)\| . \tag{5.1}$$

The total face area is the sum of the areas of all triangles. Algorithm 1 shows the algorithm used to calculate the area of one polyhedron face.

Algorithm 1 Face area computation

$area \leftarrow 0$
for $i \leftarrow 1$ to $n - 2$ **do**
  $v_1 \leftarrow vertices[0]$
  $v_2 \leftarrow vertices[i]$
  $v_3 \leftarrow vertices[i + 1]$
  $area \leftarrow area + \frac{1}{2} \|(v_2 - v_1) \times (v_3 - v_1)\|$
end for
return $area$

[1] This refers to valid polyhedron as defined in section 4.3.6.3

5.6.2.2 Volume

The volume of a polyhedron can be calculated by dividing it into pyramids [3]. For each face of the polyhedron, the algorithm computes the volume of the pyramid associated with the face. Let q be any of the face's vertices, n the surface normal of the face, and A the surface area of the face. The volume V_P of the pyramid is given by the formula:

$$V_P = \frac{1}{3}(q \cdot n) \cdot A. \tag{5.2}$$

The total volume of the polyhedron is the sum of all of the pyramid volumes. This approach assumes that the polyhedron is closed and that all face normals point outward. Algorithm 2 shows the algorithm used to calculate the volume of a polyhedron.

Algorithm 2 Polyhedron volume computation

$V \leftarrow 0$
for each face f in faces **do**
  Select three vertices v_1, v_2, v_3 from f
  $n \leftarrow \text{normalize}((v_3 - v_1) \times (v_2 - v_1))$
  $A \leftarrow \text{area}(f)$
  $q \leftarrow v_1$
  $V \leftarrow V + \frac{1}{3}(q \cdot n) \cdot A$
end for
return V

5.7 Tasks

Each task is a scene whose root node is of a type derived from the `Task` class. The `Task` class defines methods for the task difficulty, description, parameters, and steps. Each derived task class then overrides those methods to create a unique task. The only children of the task scene's root node are the the solids that make up the task. Figure 5.4 shows the diagram of the `Task` class, classes derived from it, and other related classes.

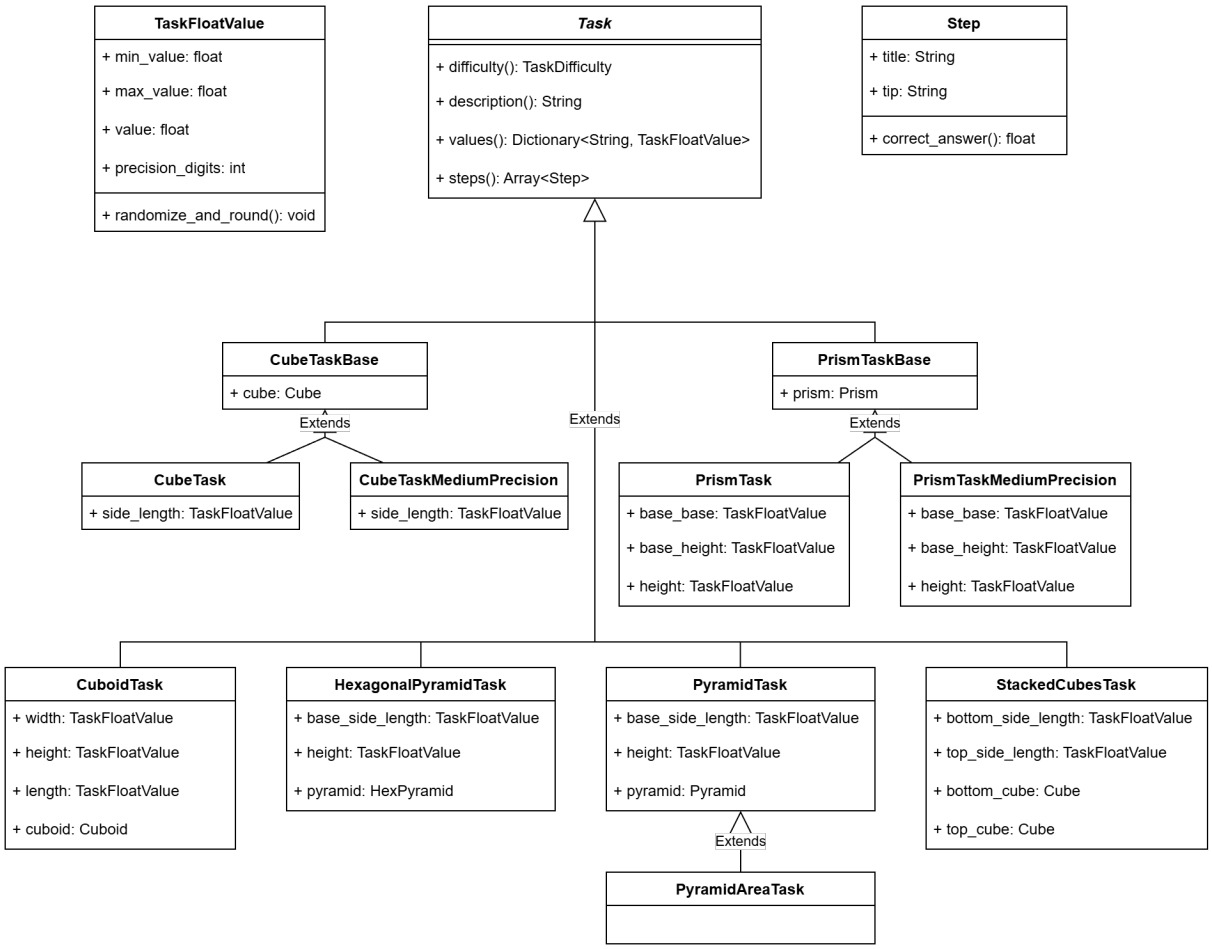


Figure 5.4: Task class diagram. Less important class members were omitted for conciseness.

- *TaskFloatValue* represents a task parameter. Each instance specifies the current, minimum, and maximum value as well the precision with which the value is randomly picked.
- *Step* represents a single task step.
- *Task* is the base class for all tasks.
- *CubeTaskBase* is the base class of *CubeTask* and *CubeTaskMediumPrecision*.
- *CubeTask* corresponds to the *Cube* task.
- *CubeTaskMediumPrecision* corresponds to the *Cube With Decimal Digits* task.
- *PrismTaskBase* is the base class of *PrismTask* and *PrismTaskMediumPrecision*.
- *PrismTask* corresponds to the *Prism* task.
- *PrismTaskMediumPrecision* corresponds to the *Prism With Decimal Digits* task.
- *CuboidTask* corresponds to the *Cuboid* task.

- *HexagonalPyramidTask* corresponds to the *Hexagonal Pyramid* task.
- *PyramidTask* corresponds to the *Pyramid* task.
- *PyramidAreaTask* corresponds to the *Pyramid Area* task. It features the same parameters but overrides the `steps` method.
- *StackedCubesTask* corresponds to the *Stacked Cubes* task.

5.8 User interface

User interface in Godot is created with the use of `Control` nodes. Contol nodes used in GeoApp may be split into three categories: containers, interactive elements, and labels. Containers contain other UI elements and are responsible for their layout. They are nodes that derive from the `Container` class. There are many types of containers. Each type of container lays out its children in a different way. For instance, the `HBoxContainer` lays out its children horizontally (see figure 5.5). The `VBoxContainer` container lays out its children vertically (see figure 5.6). Other types of containers include the `GridContainer` and `ScrollContainer`.

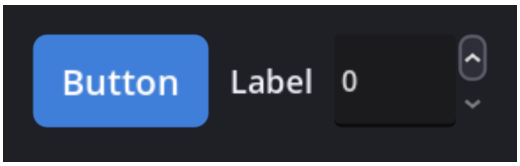


Figure 5.5: A `Button`, `Label`, and `SpinBox` in a `HBoxContainer`.

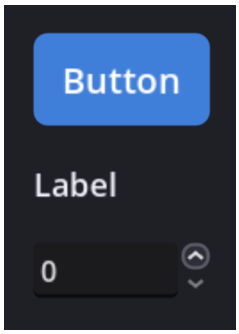


Figure 5.6: The same nodes in a `VBoxContainer`.

Interactive elements allow the user to interact with the application. They define signals which are emitted when the user iteracts with them. The two main interactive elements in GeoApp are the `Button` and `SpinBox` nodes. The `Button` node exposes the `pressed` signal which is emitted when the button is pressed. The `SpinBox` node exposes the `value_changed` signal which is emitted when the value inside of the spin box changes.

Labels are non-interactive elements which display text. The text can be changed dynamically. Labels are represented by two nodes: `Label` and `RichTextLabel`. `Label` is the simpler of the two. It allows for displaying text without additional formatting. `RichTextLabel` allow the text to be formatted with the BBCode (Bulletin Board Code) markup language [1]. BBCode allows sections of the text to be bolded, italicized or underlined. It also allows other format specifiers.

5.9 Polyhedron files

Polyhedron files store polyhedrons used in the playground module. They are introduced in section 4.3.6.4. This section explains their internal structure.

Polyhedron files store their content in the JSON (JavaScript Object Notation) format [8]. The file stores a JSON array of vertex objects. A vertex object is a JSON object with name-value pairs defined in table 5.1. A vertex object can contain other name-value pairs. Name-value pairs other than the expected ones will be ignored. Listing 5.1 contains the content of a polyhedron file which stores two vertices.

Table 5.1: Name-value pairs of a valid vertex JSON object.

Name	Value	Note
name	string	The name of the vertex
x	number	The x coordinate of the vertex
y	number	The y coordinate of the vertex
z	number	The z coordinate of the vertex

Listing 5.1: The content of a polyhedron file containing two vertices.

```
[
  {
    "name": "Vertex 1",
    "x": -1.0,
    "y": 0.0,
    "z": 0.0
  },
  {
    "name": "Vertex 2",
    "x": 1.0,
    "y": 0.0,
    "z": 0.0
  }
]
```

5.10 Project file structure

Godot does not enforce a strict file structure. Any file can be placed anywhere in the project directory or its subdirectories. Despite that, files were organized into subdirectories to group them logically and reduce the size of the main directory. The project directory is divided into three subdirectories: `assets`, `scenes`, and `scripts`. Visually, the project file structure looks like:

```
GeoApp
├── assets
├── scenes
└── scripts
```

- `assets` contains non-code resources used by the application, such as icons, and the application's theme.
- `scenes` contains Godot scene files, which define the structure and composition of the user interface and 3D objects.
- `scripts` contains script files implementing the application logic, including input hanlding and geometric calculations.

This structure improves clarity and simplifies project navigation.

Chapter 6

Verification and validation

Testing is an important part of the software development process. Software testing is a process whose goal is to make sure that the tested software behaves as intended [10]. Testing helps ensure that the system behaves as expected under both normal and exceptional conditions.

Software testing can be divided into several categories, such as unit testing, integration testing, and system testing [13]. In addition, user-oriented testing focuses on validating usability and overall behavior of the application.

Due to the scope and nature of the project, testing mainly focused on system testing and manual testing of the application functionality. Testing was performed iteratively throughout the development. Individual features were tested after being implemented to verify that they function correctly and do not break existing functionality.

The application was tested only by the author of the thesis. Due to limited access to other devices, the application was only tested on a laptop device with the following software and hardware configuration:

CPU	AMD Ryzen 7 5800H
GPU	NVIDIA GeForce RTX 3050
RAM	16 GB
Operating system	Windows 11

6.1 Identified bugs

During testing, several bugs were identified and resolved. This section describes selected bugs that were discovered, their causes, and the applied fixes.

6.1.1 Unintentional camera zooming

While testing the camera movement, a bug was found when opposite movement keys were pressed simultaneously. In this situation, the camera zoomed in toward the origin at

a slow rate.

The issue was caused by an incorrect implementation of camera movement logic. For each movement direction, a difference (delta) from the original camera position was calculated. All deltas were then accumulated into the final position. When opposite movement directions were processed in the same update cycle, their deltas did not cancel out. This caused the camera position vector to be slightly shorter after the camera movement calculations.

The problem was resolved by modifying the update logic. In the fixed version, each camera rotation is calculated sequentially. Changes from previous directions' calculations are included in the subsequent ones. This approach guarantees that the effects fully cancel out when the camera is moved in opposite directions at the same time. Listings 6.1 and 6.2 show the camera movement logic before and after the fix.

Listing 6.1: Original camera movement implementation.

```
var next_camera_position := position
if move_up and camera_angle - rotate_amount > -PI / 2.0:
    var left := -Vector3.UP.cross(position).normalized()
    next_camera_position += position.rotated(left,
        rotate_amount) - position
if move_down and camera_angle + rotate_amount < PI / 2.0:
    var right := Vector3.UP.cross(position).normalized()
    next_camera_position += position.rotated(right,
        rotate_amount) - position
if move_left:
    next_camera_position += position.rotated(Vector3.DOWN,
        rotate_amount) - position
if move_right:
    next_camera_position += position.rotated(Vector3.UP,
        rotate_amount) - position

position = next_camera_position
```

Listing 6.2: Fixed camera movement implementation.

```
if move_up and camera_angle - rotate_amount > -PI / 2.0:
    var left := Vector3.DOWN.cross(position).normalized()
    position = position.rotated(left, rotate_amount)
if move_down and camera_angle + rotate_amount < PI / 2.0:
    var right := Vector3.UP.cross(position).normalized()
    position = position.rotated(right, rotate_amount)
```

```
if move_left:
    position = position.rotated(Vector3.DOWN, rotate_amount)
if move_right:
    position = position.rotated(Vector3.UP, rotate_amount)
```

6.1.2 Issues related to edge cylinder transform

A bug was discovered when testing a polyhedron whose center was sufficiently far away from the origin. The bug caused the informational labels to be in incorrect positions. Vertical edge cylinders were visibly thinner compared to the rest.

The issue was caused by an incorrect implementation of the edge cylinder rotation logic. The rotation of an edge cylinder is set using a 3x3 basis matrix. Originally, the basis matrix was calculated with the assumption that the center of the polyhedron was in the origin of the global coordinate system. The result appeared correct for solids whose center was close to the origin of the global coordinate system. As the solid got further away from the origin, the error became more apparent.

The bug was fixed by correctly calculating the center of the polyhedron. In the fixed version, the center of the polyhedron is calculated as the average position of all of its vertices. Figures 6.1 and 6.2 show the *Stacked Cubes* task before and after the fix.

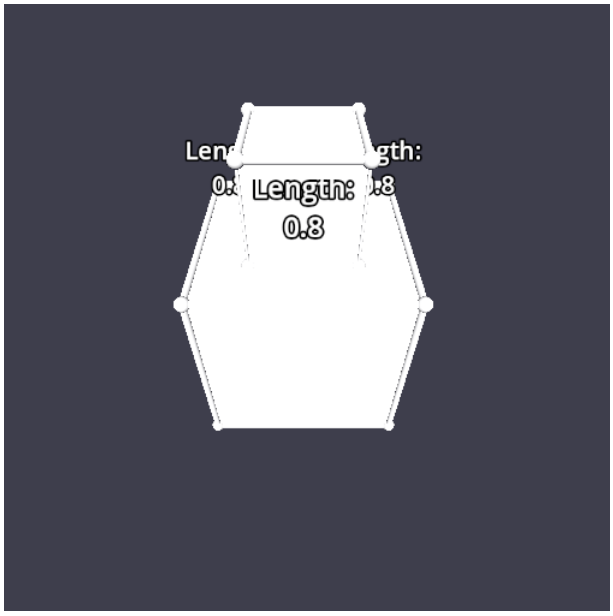


Figure 6.1: Before the fix. Labels appear in incorrect positions. Vertical edges of the top cube are slightly thinner than the rest.

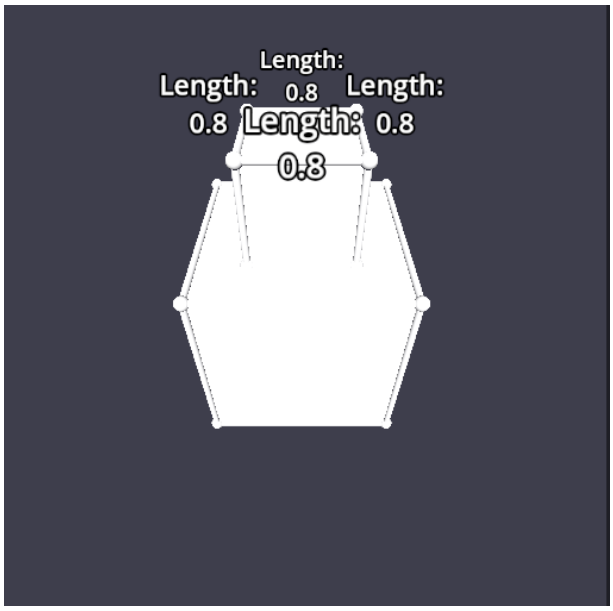


Figure 6.2: After the fix.

Chapter 7

Conclusions

7.1 Obtained result

The goal of the project was to create an educational application which made learning stereometry easier. The finished result achieves this goal with the use of three modules, namely task exploration, task solving, and the playground.

The task exploration module teaches the user how to solve various tasks from the field of stereometry. It prepares the user for solving the tasks themself in the task solving module. The ability to change task parameter values helps the user visualize how those changes influence how the solids featured in the task look.

The task solving module improves the user's skills related to calculation of various characteristics of solids. By dividing each task into steps, the user learns how to divide a bigger problem into smaller parts. Multiple difficulties allow the user to select the difficulty appropriate for their skill level. With time, the user may progress to more difficult tasks.

The playground module allows the user to freely explore 3D geometry by playing with polyhedrons. The user can create a polyhedron from a list of vertices and see the polyhedron in a 3D view. The properties of the displayed polyhedron can be inspected to help the user better understand the relationship between the specified vertices and the created polyhedron.

7.2 Evaluation of used tools and technologies

The chosen tools played an important role in the successful completion of the project. The Godot engine provided scene management, user interface creation, and 3D rendering within a single environment. This made it a suitable platform for developing an educational application with 3D rendering capabilites.

The use of the Neovim text editor improved the development of the application's source code. Its extensibility and keyboard-oriented controls allowed for fast editing of scripts.

Neovim's integration with version control tools made version control simpler and more efficient.

Version control was managed using Git and the repository was hosted on GitHub. This enabled systematic tracking of changes, safe experimentation during development, and easy recovery from errors. Publishing the source code of the application supports future extension of the application.

7.3 Further development

The application meets the specified functional and non-functional requirements. Despite that, there is room for improving the existing features and adding new ones.

The keyboard controls for the 3D view currently cannot be modified. This could be improved by allowing the user to change the controls in the settings.

Another improvement for the playground is related to the polyhedron's vertices. Currently, the only way to move a vertex is through the vertex list. Another way of moving a vertex could be accomplished through the 3D view. Upon clicking a vertex, a control would appear. The control would allow the user to easily move the vertex on each axis.

The set of tasks is small and could thus be expanded. New tasks could feature new solids, including non-polyhedrons.

7.4 Encountered difficulties

Even though the used tools were well suited for the project, some difficulties were encountered during the development. An issue that appeared multiple times was screen state management. Some elements, like the polyhedron or task in the 3D view, were not being reset correctly when entering or leaving the screen they were in. Care had to be taken to ensure that the state of each screen would be correct after leaving and entering it. Another issue was the class structure. Classes were refactored several times to accomodate for changing requirements. This was especially prevalent in classes related to solids.

7.4.1 Acquired knowledge and experience

The development of the application provided valuable experience in the design and implementation of an interactive three-dimensional application with a graphical user interface. As this was the first project of this type for the author, particular emphasis was placed on understanding 3D scene management and spatial transformations.

The project also required integrating 3D visualization with user interface elements. This led to a deeper understanding of event-driven programming and communication between subsystems of the application.

The Godot engine was used in combination with the Neovim text editor, which required configuring and connecting Neovim to Godot's language server. This contributed to improved understanding of development tools and communication between them.

Overall, the project significantly expanded the author's practical experience in software development, particularly in the areas of 3D graphics, application architecture, and toolchain configuration.

Bibliography

[1] *BBCode.org, BBCode users guide and tricks on web2 and web3*. URL: https://www.bbcode.org (visited on 29/01/2026).

[2] S. Chacon and B. Straub. *Pro Git*. Apress Berkeley, CA, 2014. ISBN: 9781484200773.

[3] Wm. Randolph Franklin. *Volume of a Polyhedron*. URL: https://wrfranklin.org/Research/Short_Notes/volume.html (visited on 03/02/2026).

[4] Eric Freeman and Elisabeth Robson. *Head first design patterns*. O'Reilly Media, 2020.

[5] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Pearson Education, 1994. ISBN: 9780321700698. URL: https://books.google.pl/books?id=6oHuKQe3TjQC.

[6] S. Gottwald. *The VNR Concise Encyclopedia of Mathematics*. Springer Netherlands, 2012, p. 193. ISBN: 9789401169844. URL: https://books.google.pl/books?id=bCLfnQEACAAJ.

[7] N. Gunasinghe and N. Marcus. *Language Server Protocol and Implementation: Supporting Language-Smart Editing and Programming Tools*. Apress, 2021. ISBN: 9781484277911. URL: https://books.google.pl/books?id=zeuezgEACAAJ.

[8] *JSON*. URL: https://www.json.org/json-en.html (visited on 29/01/2026).

[9] Serge Lang and Gene Murrow. *Geometry: A High School Course*. Springer New York, NY, 1988.

[10] Glenford J. Myers, Corey Sandler and Tom Badgett. *The Art of Software Testing*. Wiley, 2011.

[11] G. Polya. *Mathematics and Plausible Reasoning: Induction and analogy in mathematics*. Induction and Analogy in Mathematics. Princeton University Press, 1990, p. 138. ISBN: 9780691025094. URL: https://books.google.pl/books?id=-TWTcSa19jkC.

[12] *System requirements*. URL: https://docs.godotengine.org/en/stable/about/system_requirements.html (visited on 18/01/2026).

[13] *Types of Software Testing - GeeksforGeeks*. URL: https://www.geeksforgeeks.org/software-testing/types-software-testing/ (visited on 03/02/2026).

Appendices

Index of abbreviations and symbols

2D	two-dimensional
3D	three-dimensional
BBCode	Bulletin Board Code
FOSS	free and open-source software
GUI	graphical user interface
JSON	JavaScript Object Notation
LSP	Language Server Protocol
PC	personal computer
UI	user interface
A	area of a plane figure
V	volume of a solid

List of additional files in electronic submission

Additional files uploaded to the system include:

- **GeoAppSource.zip** - zipped project files of the application,
- **GeoAppVideo.mp4** - video file showing how the application works.

List of Figures

2.1 A right prism with a rectangular base. 6
2.2 A right pyramid with a hexagonal base. 7
2.3 GeoGebra 3D Calculator. 9
2.4 Cabri 3D. 10
2.5 Cabri Express. 11
2.6 Shapes 3D Geometry Learning. 12
2.7 Shapes 3D Geometry Drawing. An error occurs when trying to run the demo. 13

4.1 The main menu of the application. 21
4.2 The 3D view as seen in the task exploration and task solving modules. . . 22
4.3 The 3D view as seen in the playground module. 22
4.4 The navigation bar as seen in the task exploration screen. 23
4.5 The navigation bar as seen in the settings screen. 23
4.6 Empty task exploration module. 24
4.7 Empty task exploration module. The "cub" task name filter is applied. . . 25
4.8 Task exploration module. The *Cube* task is selected. 25
4.9 Task exploration module. The *Cube* task is selected. The task has a different parameter value as compared to the one visible in figure 4.8 26
4.10 Task exploration module. The *Hexagonal Pyramid* task is selected. 26
4.11 Task exploration module. The *Prism* task is selected. 27
4.12 Task exploration module. The *Stacked Cubes* task is selected. 27
4.13 Task exploration module. The *Stacked Cubes* task is selected. The step list is scrolled to the bottom. 28
4.14 The task selection screen. The easy difficulty is selected. No task is selected. 29
4.17 The task solving screen. The *Cube* task is loaded. 29
4.15 The task selection screen. The medium difficulty is selected. No task is selected. 30
4.16 The task selection screen. The *Cube* task is selected. 30
4.18 The task solving screen. The step list is marked with a red rectangle. . . . 31
4.19 A green checkmark appears next to the spin box when the correct answer is provided. 32

4.20 A red X symbol appears next to the spin box when an incorrect answer is provided. 32
4.21 The state of the task solving screen after moving to the second step. 33
4.22 The state of the task solving screen after the task is completed. 33
4.23 The state of the task solving screen after pressing the **Get new task** button. 34
4.24 The default state of the playground module. 35
4.25 The predefined prism vertices loaded in the playground. 36
4.26 A new vertex is added to predefined prism vertices. The polyhedron becomes invalid. 37
4.27 The position of the new vertex is set to (0, 2, 0). The polyhedron becomes valid. 37
4.28 The settings screen. 39

5.1 The complete application structure starting from the `root` node. 42
5.2 The Main scene. 43
5.3 A simple 3D object scene. The red node is the root node of the scene. . . . 44
5.4 Task class diagram. Less important class members were omitted for conciseness. 47
5.5 A `Button`, `Label`, and `SpinBox` in a `HBoxContainer`. 48
5.6 The same nodes in a `VBoxContainer`. 48

6.1 Before the fix. Labels appear in incorrect positions. Vertical edges of the top cube are slightly thinner than the rest. 53
6.2 After the fix. 54

List of Tables

5.1 Name-value pairs of a valid vertex JSON object. 49