Development of the Student Simulator Game: From Concept to Code

Vasyl P. Oleksiuk[1,2], Denys Y. Dzhuha[1], Petro P. Melnyk[1] and Dmytro V. Verbovetskyi[2]

[1]*Ternopil Volodymyr Hnatiuk National Pedagogical University, 2 Maxyma Kryvonosa str., Ternopil, 46027, Ukraine*
[2]*Institute for Digitalisation of Education of the NAES of Ukraine, 9 M. Berlynskoho Str., Kyiv, 04060, Ukraine*

Abstract

This paper describes and analyses the phases of development of a Student Simulator game application. The study analyses various game types employed in computer science education. Based on a SWOT analysis, the authors justify using simulators and combined gaming applications. Several basic requirements for a Student Simulator game are identified, such as a 3D interface, multiple game locations, manipulation of object models (computers, operating systems, programming languages), and the registration and rating of players. A matrix of game elements that meets the requirements for the Student Simulator has been created.

Following a comparative analysis, Godot, Blender, and Firebase were selected as development tools. The authors describe their experience in developing the Student Simulator game. The design process of this game emphasises the incorporation of game elements with both testing and code-writing tasks involving the manipulation of computer hardware. The paper includes some fragments of the game workspace and application code. Further improvements to the game are indicated.

Keywords

educational games, game design, game development, Godot, Blender

1. Introduction

In today's world, where new technologies are created, and old ones are improved, the demand for IT specialists is constantly growing. That is why educational games are used increasingly in the educational process. Teachers of various disciplines use games in both desktop and computer formats. The design and development of such games are becoming increasingly relevant due to the possibility of engaging students in active learning. Educational games provide an interactive and engaging experience that can capture learners' attention and keep them interested. The integration of game elements such as challenges, rewards and feedback increases learners' motivation to learn and retain learning content compared to traditional methods. Games encourage active participation and develop critical thinking, problem-solving and decision-making skills. Unlike passive learning, educational games engage learners in activities that require application and analysis, which is crucial for deep learning. In addition, modern learning games can be designed to adapt to learners' pace, style and needs. This personalised approach helps to accommodate different learning abilities, ensuring that every learner can develop effectively. Learning games can help to develop 21st Century Skills such as collaboration, communication, creativity and technological literacy, which are essential in today's digital world. By simulating real-life scenarios, games prepare learners for future challenges innovatively and practically. The use of educational games contributes to the development of digital literacy and technological competencies, which are an integral part of modern education and are essential for navigating the digital age. With the rapid growth of the gaming industry, users are offered more and more game applications for learning theoretical knowledge and skills, especially in computer science. Today, there are many types of games, such as

CS&SE@SW 2024: 7th Workshop for Young Scientists in Computer Science & Software Engineering, December 27, 2024, Kryvyi Rih, Ukraine

oleksyuk@fizmat.tnpu.edu.ua (V. P. Oleksiuk); dzhuga_dy@fizmat.tnpu.edu.ua (D. Y. Dzhuha); melnyk_pp@fizmat.tnpu.edu.ua (P. P. Melnyk); verbovetskyj.dv@gmail.com (D. V. Verbovetskyi)
https://tnpu.edu.ua/faculty/fizmat/oleksyuk-vasil-petrovich.php (V. P. Oleksiuk)
0000-0003-2206-8447 (V. P. Oleksiuk); 0000-0001-6261-8429 (D. Y. Dzhuha); 0009-0006-4867-6386 (P. P. Melnyk); 0000-0002-4716-9968 (D. V. Verbovetskyi)

simulation games, puzzle-based learning games, language learning games, strategy and planning games, and role-playing educational games. Many authors have highlighted the advantages and disadvantages of these games for computer science education [1], [2], [3]. We systematised these factors according to the SWOT methodology (see Table 1)

Table 1: SWOT analysis of some types of educational game applications

SWOT/ Games	**Strengths**	**Weaknesses**	**Opportunities**	**Threats**
Simulation	1. Realistic simulation of software development environments	1. High development costs for realistic CS simulations	1. Integration with actual development tools and IDEs	1. Rapid technological changes making simulations obsolete
	2. Hands-on practice with system architecture and networking concepts	2. May oversimplify real-world technical complexities	2. Growing demand for DevOps and cloud computing training	2. Competition from real-world programming environments
	3. Safe environment for experimenting with security concepts	3. Requires significant computational resources	3. Potential for incorporating real-world codebases	3. Risk of teaching outdated practices
	4. Direct application of theoretical knowledge	4. Complex to maintain and update with new technologies	4. Expansion into emerging technologies (AI, blockchain)	4. High expectations from tech-savvy students
	5. Effective for teaching complex systems and algorithms	5. Challenging to simulate all possible edge cases	5. Virtual lab environments for remote learning	5. Security risks in networked simulations
Puzzle-Based	1. Excellent for teaching algorithmic thinking	1. May seem too simplified for advanced concepts	1. Integration with popular programming languages	1. Market saturation of coding puzzle games
	2. Clear progression path for learning concepts	2. Limited applicability to large-scale programming	2. Gamification of algorithm complexity concepts	2. Risk of students only learning "puzzle solutions"
	3. Low barrier to entry for beginners	3. Can focus too much on solutions rather than process	3. Development of multiplayer coding challenges	3. Disconnect from professional programming practices
	4. Immediate feedback on solution efficiency	4. Often lacks real-world programming context	4. Expansion into mobile learning platforms	4. Competition from coding challenge websites
	5. Easy to integrate with existing curricula	5. May not teach good documentation practices	5. Creation of custom puzzle sets by educators	5. Potential focus on entertainment over education

Strategy	1. Teachers project management and resource allocation	1. May oversimplify software project complexities	1. Integration with real project management tools	1. Gap between game strategies and real-world practices
	2. Develops system architectural thinking	2. Teaching Agile and Scrum methodologies	2. Time-intensive gameplay	2. Risk of teaching outdated management methods
	3. Suitable for learning optimization strategies	3. Difficult to assess individual learning	3. Simulation of large-scale system design	3. Competition from professional PM tools
	4. Encourages long-term planning skills	4. Less focus on actual coding skills	4. Incorporation of real-world case studies	4. Difficulty in keeping content current
	5. Helps understand trade-offs in system design	5. Complex learning path	5. Development of team-based scenarios	5. Potential lack of technical depth
Role-Playing	1. Excellent for teaching software development roles	1. Limited focus on technical skills	1. Integration with real code review processes	1. Risk of stereotyping tech roles
	2. Develops collaboration and communication skills	2. High production costs	2. Teaching soft skills for tech roles	2. Difficulty in maintaining relevance
	3. Good for learning debugging scenarios	3. Difficult to create realistic scenarios	3. Simulation of client interactions	3. Competition from real-world experiences
	4. Teaches problem-solving in context	4. May oversimplify workplace dynamics	4. Development of ethical decision-making scenarios	4. Challenge in assessment metrics
	5. Helps understand user perspectives	5. Challenge in balancing fun and learning	5. Creation of team-based learning experiences	5. Potential lack of technical credibility
Language Learning	1. Progressive learning of syntax and semantics	1. May oversimplify language complexities	1. Integration with multiple programming languages	1. Rapid evolution of programming languages
	2. Regular practice and reinforcement	2. Focus on syntax over problem-solving	2. Adaptive learning paths	2. Competition from traditional tutorials
	3. Immediate feedback on code correctness	3. Limited coverage of advanced concepts	3. Mobile-first learning approaches	3. Risk of surface-level learning
	4. Gamified repetition of core concepts	4. May not teach best practices	4. Social coding elements	4. Challenge in teaching practical application
	5. Effective for vocabulary and syntax learning	5. Risk of mechanical learning	5. Personalized learning progression	5. Market saturation

As seen from Table 1, each type of educational game has advantages and disadvantages. They should be applied according to the field of computer science. The authors of the studies believe that simulations are the most suitable genre for teaching computer science. The main factors supporting this thesis are engagement [4], [5] motivation [6], [7] impact on cognitive activity, and declarative and procedural knowledge of students [8]. Another approach to using educational games in computer science teaching involves combining different types of games, each of which meets specific learning objectives [1], [9]. Success depends on careful alignment with the needs of the curriculum [4], the skill levels of the students [10], and the resources available [11]. We had experience with a similar learning game called PythonLearner, which can be classified as both a Language Learning Game and a Simulation Game [12]. Now, we decided to develop another Student Simulator game. Creating a Student Simulator game is very relevant nowadays for several reasons.

Firstly, the use of games in the educational process can motivate students to engage with the subject matter and facilitate a deeper understanding of the discipline. Additionally, the Student Simulator enables users to immerse themselves in the experience of student life, which can serve as an effective tool for showcasing educational institutions to prospective applicants and illustrating the educational process. This platform allows educators to conduct interactive lessons, where students can not only learn theoretical concepts but also apply them through specially designed mini-games.

The teacher's ability to add and modify tasks ensures a wide range of engaging and unique challenges. The game's immediate feedback mechanism will alert students to incorrect actions and, when necessary, provide guidance on how to correct their mistakes. The inherent feature of progressively increasing difficulty in most games will promote incremental learning, starting with basic tasks and advancing to more complex projects. This approach to education guarantees a smooth and thorough transition from student to competent professional.

Although there are many off-the-shelf solutions on the Internet, the problem of creating new gaming applications remains relevant. **Our article aims** to create a simulation game to study computer science disciplines. To do this, we need to solve the following tasks:

1. Review the existing analogues of the projected game.
2. Select tools for development.
3. Design a model of the game application.
4. Describe and analyse the main stages of development.

2. Game design

2.1. A brief overview of existing gaming simulators

Before developing the Student Simulator game, we analysed similar projects that have gained an audience among simulators and gaming platforms for learning. The analogues include MHRD, TIS-100, Shenzhen I/O, Minecraft Education Edition, CyberStart, and Classcraft, which offer game simulations of various areas of computer science.

MHRD is a hardware design game where players go from building basic logic gates to creating a fully functional processor using the Hardware Description Language (HDL) [13]. Despite its exciting idea and advanced features, commercial software is available via Steam. TIS-100 is a complex game for learning assembly language programming. It simulates parallel computing. Players must optimise code and solve puzzles using a fictional assembly language, teaching low-level programming concepts [14]. Like MHRD, it is available on Steam and is not freeware.

Shenzhen I/O is a similar simulator to TIS-100. However, it focuses on hardware engineering and embedded systems programming. Players design circuits and write code for various electronic devices.

Classcraft is not an accurate simulation game. However, it is a modern platform that turns learning into a role-playing game. Students complete tasks, improve their characters, earn points for achievements, and receive virtual penalties for mistakes [15]. The main goal of Classcraft is to increase student motivation by introducing gamification elements into the educational environment [16]. The game

integrates with some educational platforms, allowing teachers to track student progress and encourage student engagement. The main feature of Classcraft is team dynamics, which is when students are united in groups to achieve common goals.

Minecraft Educational Edition is an educational version of Minecraft's popular video game, designed specifically for educational purposes. Using this tool, it is possible to create educational environments and tasks for students of different ages and levels of learning. Educational Minecraft aims to promote collaboration and cooperation between players by creating shared virtual environments where they can work together on projects and tasks. The game provides many opportunities for creative expression and imagination. Users can create their worlds, objects and structures using a variety of blocks and tools. Within the game, you can create educational tasks and scenarios that help players learn various subjects, such as mathematics, computer science, history and others [17].

Given that combining different types of games when teaching computer science is advisable, quiz elements can be incorporated into the designed simulation game. Currently, one of the most famous platforms for creating quizzes is Kahoot! It focuses on conducting quizzes, polls, and other forms of interactive interaction. The main advantage of Kahoot! is its ease of use and the ability to engage many participants simultaneously. Students answer questions in real-time, competing for the highest score. The game-like structure of Kahoot! Stimulates interest in learning and encourages healthy competition between participants. Although Kahoot! does not have as advanced role-playing elements as Classcraft, it is well suited for integration into the classroom as an additional tool for testing knowledge.

2.2. Analysis of the main approaches and stages of game simulation development

When designing a training simulator, we understand that teams of many specialists, such as storytellers, modellers, designers, programmers, QA engineers, and others, are currently working on creating such games. Moreover, they have been working for a long time. Nevertheless, we organised a project to design and develop the Student Simulator game. The work was carried out within the framework of the joint research laboratory 'Digital Educational Innovations', which was created in cooperation with Ternopil Volodymyr Hnatiuk National Pedagogical University and Institute for Digitalisation of Education of the National Academy of Pedagogical Sciences of Ukraine. The project methodology was used to implement the tasks. The article's authors and students studying under the Game Project Engineering programme at Ternopil Volodymyr Hnatiuk National Pedagogical University led and executed the project. Designing a simulation game involves reducing this complex reality to a simpler model. [18]. Three principles play a role in building this simpler model:

- Reduction of elements. Not all of them will be represented in the simulator.
- Abstraction. The elements included in the new model are represented in less detail than in real life.
- The symbolism of elements. Some real-life objects are presented in the new model in a new form.

We tried to organise the game development workflow following the study by Vincent Peters and Samenspraak Advies [18] (see (figure 1)). However, due to the scale of the project and time constraints, not all defined phases were strictly followed.

The first phase involved drafting the terms of reference. The game's goal was to engage the user in learning the basics of a computer system by simulating a student's actions moving through several classrooms that can be thought of as online laboratories [19], [20]. At this stage, the developers consulted with teachers who had experience teaching courses such as Operating Systems, Computer Architecture, and Programming. The specification of the game as a simulator was finally defined, and what it should be at the end of the project was determined. The following requirements for the end application were specified:

- It should be a desktop game in the simulation genre.
- Several locations (rooms) of the game should be implemented, including a quest room, a room for learning the PandaOS, and a room for manipulating a prototype computer system unit.

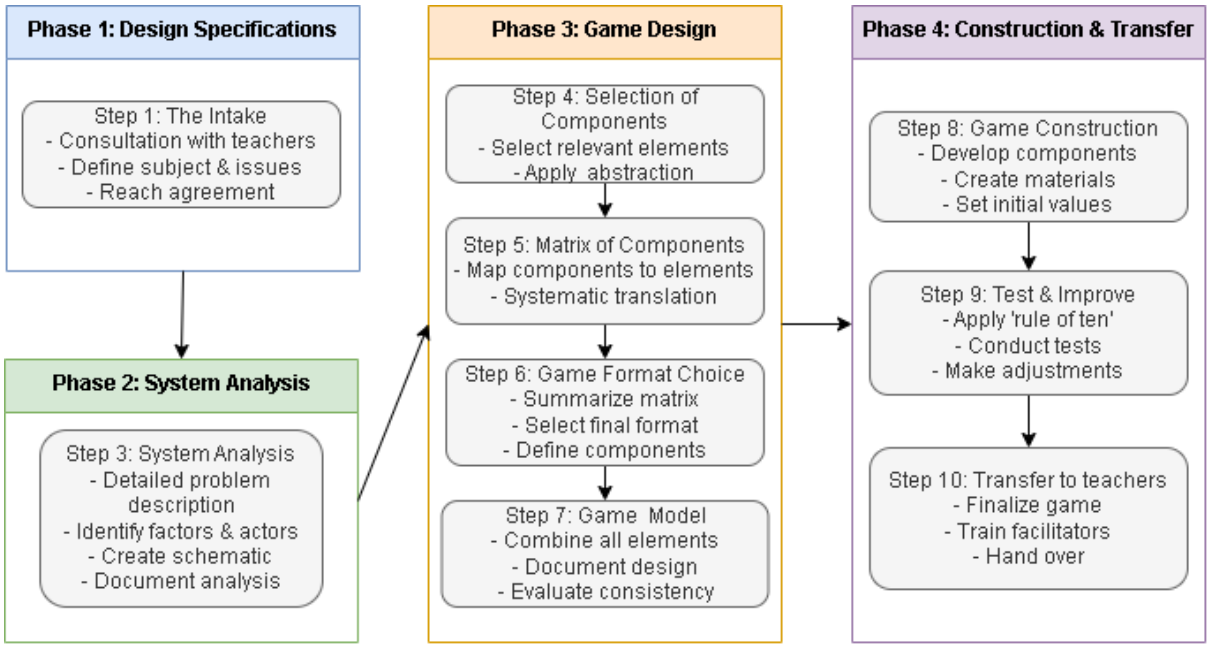

Figure 1: Stages of creating a game simulator

- The application should have a system for registering gamers.
- The safe and efficient storage of user data and their progress and achievements must be implemented.

During the system analysis phase, the actor (student), its relationships with the surrounding objects and the factors influencing the player's actions were identified. The surrounding objects are walls, doors between rooms, windows, computers, tablets, etc. The game design phase involves the transformation of the system analysis into gameplay. It consists of several steps, such as clarifying the game components through abstraction, creating a game element matrix, and choosing a game format. The game element matrix provides a structured basis for the student simulator by mapping the main system components to game elements (see Table 2).

Table 2: Matrix of game elements

Elements Systems	**Scenarios**	**Events**	**Roles**	**Rules**	**Mechanics**	**Rewards**
Academic	Class attendance sessions. ~~Exam preparation periods~~ ~~Group project deadlines.~~	Pop quizzes ~~Guest lecturer visits~~ Academic competitions	Student Teacher	Mandatory authorisation ~~Assignment deadlines~~ ~~Academic integrity policies~~	Knowledge retention mechanics Test-taking mini-quizzes ~~Study effectiveness system~~	Grade conducting ~~Academic honors~~ ~~Special certifications~~

Computer Lab	Hardware assembly challenges ~~OS troubleshooting missions~~ Programming assignments	Launch applications System settings ~~Hardware malfunctions~~ ~~Software updates~~	User ~~Lab Assistant~~ ~~IT Support~~ ~~System Administrator~~	Equipment usage rules ~~Lab safety protocols~~ ~~Software licensing~~	Hardware assembly simulation ~~Coding challenge system~~ Debugging mechanics	~~Technical badges~~ Hardware upgrades
Social	~~Social networking~~ Site of project	Faculty events Promotion among applicants ~~Advertising in newspapers and magazines~~	Events Leader(s) Study Group Member	Social interaction limits ~~Club participation rules~~	Social network system ~~Reputation mechanics~~	Social connections ~~Network benefits~~ ~~Club leadership roles~~
Progress	Games room selection events ~~Semester progression~~ ~~Graduation requirements~~	Grade announcements ~~Scholarship opportunities~~ ~~Academic warnings~~	Student ~~Research Assistant~~ ~~Student Representative~~	Grade requirements ~~Credit completion rules~~ ~~Graduation criteria~~	Experience point system ~~Achievement tracking~~ ~~Skill tree progression~~	Academic achievements ~~Unlocked opportunities~~ ~~Special titles~~

The next step was to summarise the data from the matrix, including removing (crossing out) those elements that were not feasible in this project. As a result, it becomes clear what information should be included in the game scenario and what roles and actions should be highlighted. After that, a game model was developed (see Figure 3), which combined all the game locations and identified possible player actions. The next phase was the development of the simulation game. The concepts and ideas were implemented in the programme code. The game was tested, which led to bug fixes. Teachers were also tested to determine how the functionality corresponded to the game's defined characteristics.

2.3. Choosing development tools

Before creating a Student Simulator, the necessary tools for design and development have been identified. To solve this problem, we analysed scientific publications [21], [22], [23], [24] explored vendors' official sites, and compared some software types. The main selection criteria were the following characteristics.

- Affordability. The cost of purchasing a license or subscription for the tool.
- System requirements. Demand for hardware power for comfortable development.
- Experience of the development team members in using the tool.
- Opportunities. Availability of tools to create the functionality we need.

The main game engines we considered were Godot, Unity, and Unreal Engine. *Unreal Engine* is a powerful game engine that creates AAA games and projects with high graphical requirements. Its unique Blueprints system makes it possible to create complex logic without programming, making it easier for beginners to get started. In addition, the engine supports C++ to solve the most complicated

technical problems. The Unreal Engine is known for its photorealistic graphics capabilities, which makes it ideal for cinematic-quality projects. However, significant technical resources are required to work effectively with the engine, including modern, powerful computers and a high team expertise. It is free of charge if the annual income is less than $1,000,000. This engine is great for teams with enough experience, time, and resources to implement large projects [24].

Unity is a versatile and popular engine among developers of all levels. It supports both 2D and 3D graphics. Its licensing system allows you to use the engine for free if your annual income is less than $200,000. Thanks to an extensive database of training materials, Unity provides an easy start for beginners and rapid further development. A large marketplace of resources greatly simplifies game work, allowing developers to use ready-made models, textures, shaders, and even mechanics. However, when working on massive projects, there can be problems with scene organisation and performance optimisation, which require additional efforts from developers [24].

Godot has proven to be a convenient and powerful tool for indie developers and small studios. It's completely free, which minimizes the cost of licenses or royalties, and the open-source code allows you to customize the engine to meet the specific needs of your project. Godot provides developers with tools for creating 2D and 3D games like Unity. Due to its low system resource requirements, it suits teams with low-power systems. However, it should be borne in mind that Godot may be limited in implementing complex graphical effects or large 3D projects. Nevertheless, its intuitive interface and native GDScript programming language (similar to Python) make it ideal for beginners or those who want to get started quickly [23], [22].

Therefore, at the fifth step of the project, a matrix for selecting a game engine was created based on the previously defined criteria (see Table 3).

Table 3
Game engine selection matrix

Criteria/Engine	Unreal Engine	Unity	Godot
Affordability	+	+	+
System requirements	–	+	+
Experience	–	–	+
Opportunities	–	+	+

Based on our analysis of the advantages and disadvantages of game engines, we concluded that Godot is the most suitable for our project, as it is entirely free, does not require significant system resources, is easy to use, and provides a wide range of tools for playing various simulations.

Other tools needed for the project are programs for creating and editing 3D graphics. Among them, were compared Blender and Autodesk 3ds MAX [25], [26].

Autodesk 3ds MAX is a full-featured professional 3D graphics editor. It is a paid software with a fairly high price, namely a $235 per month subscription cost for one user. The requirements for computer resources are high, especially when working on complex scenes. Our team has minimal skills with this tool, but not enough for our project. This creates the need for additional time to learn the interface and functionality. 3ds Max is a powerful editor and can provide tools for all our 3D graphics needs, such as modelling, texturing, and animation.

Blender is a software package that works with three-dimensional graphics. Blender is completely free and open source, which makes it more attractive to small teams and indie developers. It demonstrates high performance even on average computers. Although it requires powerful hardware to work with large scenes, optimization allows you to work comfortably with the correct settings. Our team's experience with Blender was insufficient, just like with Autodesk 3ds MAX. Blender offers various capabilities: modelling, animation, sculpting, material creation, and rendering. An active user community also provides numerous add-ons and plug-ins to extend the program's functionality. A matrix was also built to select a tool for working with 3D graphics (see Table 4)

At the end of the analysis, we concluded that Blender, due to its accessibility, low system requirements

Table 4
3D graphics software selection matrix

Criteria/3D graphics software	Autodesk 3ds MAX	Blender
Affordability	–	+
System requirements	–	+
Experience	–	–
Opportunities	+	+

compared to Autodesk 3ds Max, and comprehensive functionality that met our needs, was the best choice for developing graphics for our project.

Before starting development, choosing a cloud service to store user data was essential [27]. We analysed and compared such services as Firebase, Supabase, Backendless [28], [29]. Also, when choosing, it is worth considering the interaction between the network tool and the Godot engine.

Backendless is focused on creating a backend for applications and games with minimal code writing. Its key features include.

- Codeless logic creation is suitable for quick implementation of simple projects;
- Real-time support, notification system, and data synchronisation;
- Visual database editor makes it easy to work with data;
- Cloud or local deployment, flexibility in choosing the environment.

For Godot, Backendless offers convenient REST APIs for integration. Its advantages include easy setup and powerful tools for sending push notifications.

Supabase is an open-source platform that provides developers convenient tools for creating applications with a PostgreSQL backend. The main advantages of gaming projects are:

- PostgreSQL as a database that supports complex SQL queries and stores data in a structured form.
- Real-time via WebSocket for data synchronisation between clients.
- Authentication with support for OAuth, Magic Links, and others.
- Extensibility through features and the ability to deploy your own server.

Supabase integrates well with Godot via the REST API or WebSocket, which provides a convenient data exchange setup. Its main advantages are low cost, open source, and the possibility of self-hosting.

Firebase is an application development platform from Google that provides a wide range of tools for backend development. The main features that attract the attention of game developers include:

- Realtime Database, which allows you to update data instantly.
- Firestore is a more modern, flexible NoSQL database for scalable applications.
- Authentication with support for social networks, email, and passwords.
- Cloud functions for writing server logic.
- Analytics to track user activity.

Firebase is ideal for projects that require scalability, real-time, and integration with other Google services. However, its disadvantages are high costs when the project grows, the limitations of the free plan, and dependence on Google's infrastructure.

Based on this analysis, we concluded that Firebase is the best choice for our project. Its advantages in working with data, ease of integration with Godot via HTTP requests or custom add-ons, and scalability make it easy to adapt the project to future changes. However, Supabase is also a promising alternative, and Backendless can be a choice for simple games with minimal backend requirements.

2.4. Designing the game structure

In the 3rd and 4th phases of development, our team paid considerable attention to the study of scientific literature and technical documentation. The primary sources were scientific articles on developing educational games and technical manuals of Godot, Firebase, and other tools used. Additionally, we turned to developer forums such as Stack Overflow to get practical advice and find solutions to problems we had difficulty with. Conducting such research helped us to clearly define the tasks we were facing and choose the most appropriate tools to solve them. This, in turn, enabled us to understand better existing methodologies and approaches to developing such game projects.

Designing the game structure included using UML diagrams to visualise the main components and their interaction (see figure 2). We also created flowcharts to detail the program logic. Use cases were described in detail for each module, which helped us to better think through the functionality and ensure convenience for end users.

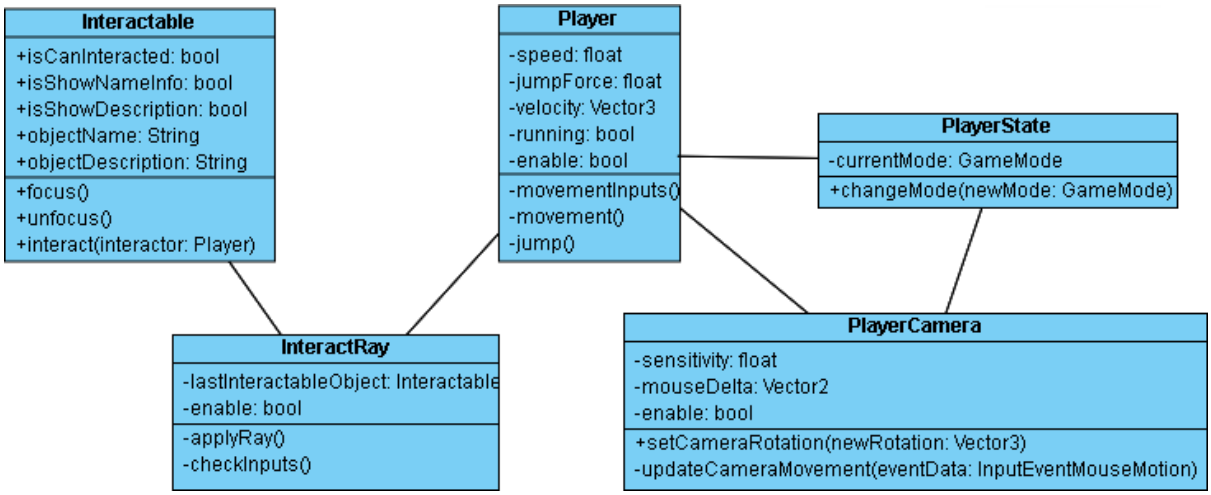

Figure 2: UML class diagram

To write the code for our project, we chose the GDScript programming language, which is part of the Godot game engine. This choice was made due to its high performance and ease of use when developing gaming applications. Using GDScript also ensured fast prototyping and efficient work with various aspects of the game.

The tests were performed manually, as manual testing is faster and less resource-intensive than developing and setting up automated tests. This allowed us to quickly identify and fix bugs, ensuring the game's high quality. In addition, we also conducted tests on real users, which allowed us to get feedback on the immersiveness of the game project.

3. Game development

To develop a game product to be released, one of the most important elements is planning resources and product design. To keep the development going at the right pace, it is necessary to distribute tasks among developers and establish constant communication between them. Therefore, game developers used the GitHub platform. It allowed us to store the project in the cloud and synchronize the work between developers. Also, thanks to the associated GitHub Projects resource, we distributed tasks for implementing the game's functionality and creating assets across the team. We also tracked the effectiveness of their implementation with further retrospectives of the implemented tasks to improve cooperation with each other.

The design and development phases of the Student Simulator game can be divided into several main steps such as

- Pre-production (steps 4-7). At this stage, the design and idea of the game were developed, the design document and matrices were written, resources were estimated, a work schedule was drawn up, and the first game prototypes were created.
- Production (step 8). This stage characterizes the primary and least active development process when all design aspects are agreed upon. The team works on the main parts of the game, such as programming the main game elements, creating content such as 3D models, music, and levels, testing the finished parts of the game, and integrating them all into a complete product.
- Beta testing (step 9). At this stage, most of the game was ready, and the main focus was on finding and fixing various bugs and performance issues, as well as polishing the visual appearance of the 3D graphics and user interface (UI).
- Release (step 10). This is the stage at which the game is fully ready for publication with access through the official website of the game.

A generalized model of the game was developed (see Figure 3).

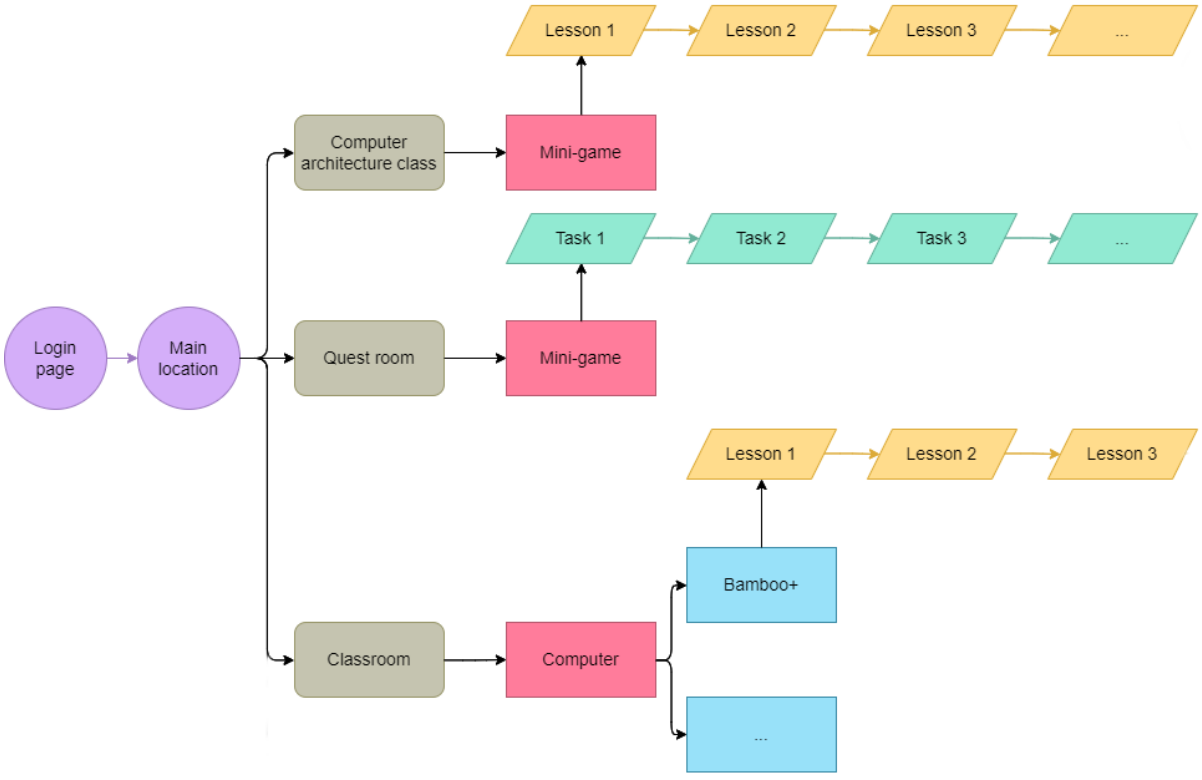

Figure 3: The model of Student Simulator game

These modules are implemented as follows in the published version of the game.

- The main player. It allows you to move around locations in the first person in the role of a student;
- Interaction system. It provides the player with interaction with the environment using the camera. Location system. It dynamically loads locations and changes them for the player.
- Virtual operating system. It is an in-game operating system that allows you to run applications and games inside the program and view information about the player.
- Visual programming language. It was created specifically for the Student Simulator project and can be considered as a model of natural programming languages.
- Authorization system. It allows you to create user accounts and store user data.
- Rating system. The module stores and displays users' rating points.
- The task system. It allows you to perform interactive tests at different levels to get points.

To implement the possibility of movement, a particular physical object was created whose movement needs to be configured using code. With the help of keyboard input, we get the direction of movement and then call a particular built-in function that makes this object move, taking into account physics and the environment. This is how the main player was implemented in the listing 1.

Listing 1: Player movement processing function

```
func _movement_inputs() -> void:
    _running = Input.is_action_pressed("run")
        _velocity = Vector3.ZERO
    _velocity.x = -Input.get_action_strength("walk_left") +
    Input.get_action_strength("walk_right")
    _velocity.z = -Input.get_action_strength("walk_front") +
    Input.get_action_strength("walk_back")
    _velocity = _velocity.normalized() * _speed *
    (RUN_SPEED_MODIFIER if _running else 1)
    _velocity = _velocity.rotated(Vector3.UP,
  deg_to_rad(main_camera.global_rotation_degrees.y))
  if Input.is_action_just_pressed("jump") and
  _grounded_ray.is_colliding():
          _jump()
```

To see the world through the eyes of the player, a virtual camera was attached to him, which followed him everywhere 4. To change the rotation of this camera when the mouse is moved, we wrote a function that receives the difference in the cursor position of this and the previous frame and assigns these values to the camera rotation.

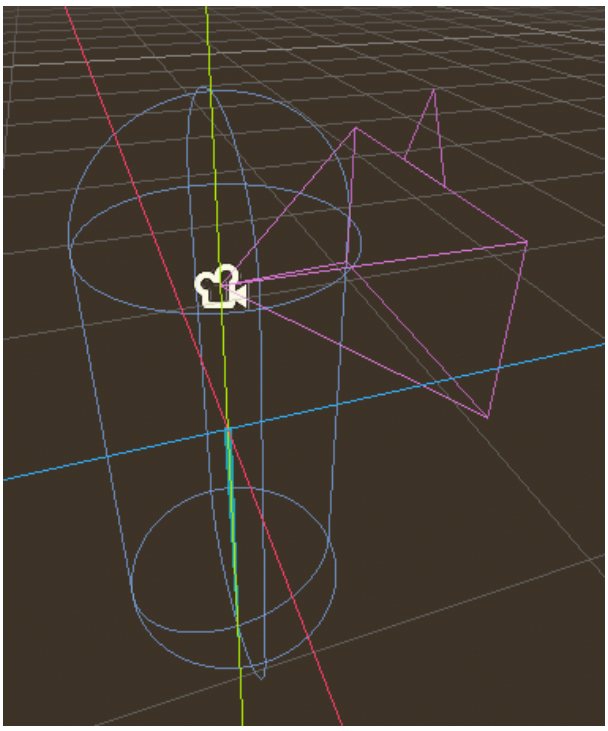

Figure 4: Visualisation of the player's body with the camera

To make the game more immersive and increase the player's connection with the environment, we created a system that allows him to interact with objects he points the camera at by pressing a unique key. To implement this mechanic, we added a beam to the camera that receives the first object it crashes into and checks whether interaction with it is possible (see Figure 5). If the player is looking at such

an object and presses the interaction key, the interactive object will call a function and respond to the interaction (see Figure 6).

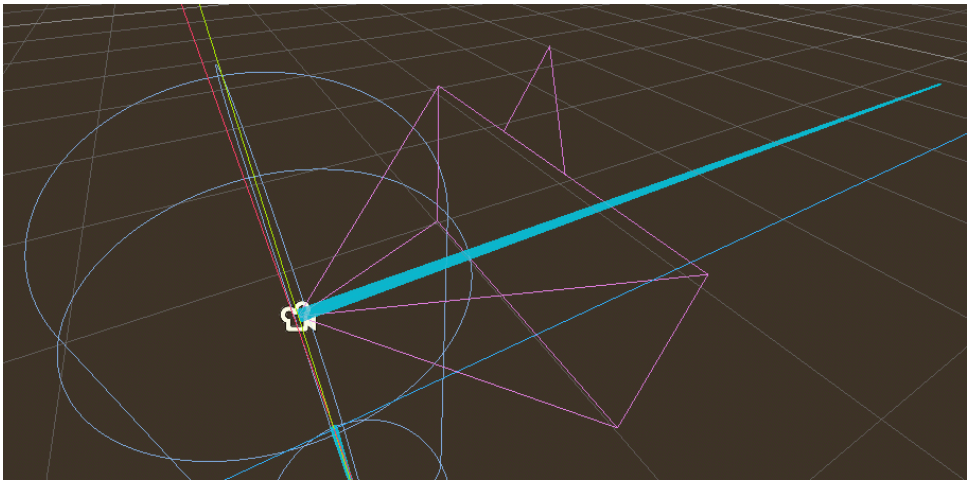

Figure 5: Visualisation of the player's body with the camera

Figure 6: The example of interaction

The game's locations, which take place in university classrooms, were divided into separate scenes. This was done to optimise the use of computing resources and increase the modularity of the project. To move between them, we developed the function that dynamically loads and unloads rooms from memory without changing the player's state (see the Listing 2).

Listing 2: Function to change the current room

```
func change_room(scene_path : String) -> void:
        _player.hide()
        _current_room.hide()
        _current_room.queue_free()
        _current_room = load(scene_path).instantiate()
        add_child(_current_room)
        _current_room.position = Vector3.ZERO
        _player.global_position =
```

```
    _current_room.get_player_start_global_position()
        var player_camera : Camera3D =
            _player.main_camera
        player_camera.set_camera_rotation
            (_current_room.get_player_start_global_rotation())
        _player.show()
```

When creating the authorisation system, we used Firebase tools such as authentication to implement the authorisation system and Firestore to store user data. We used the Godot Firebase addon to simplify the interaction between Godot and Firebase. Authentication in Student Simulator works as follows: the user enters the required data, email and password. Since these are the primary data for authorisation, they are stored in the Authentication tool. To store other data, a file with an individual and unique name for each player is created in the player_stats collection in the Firestore (see Figure 7).

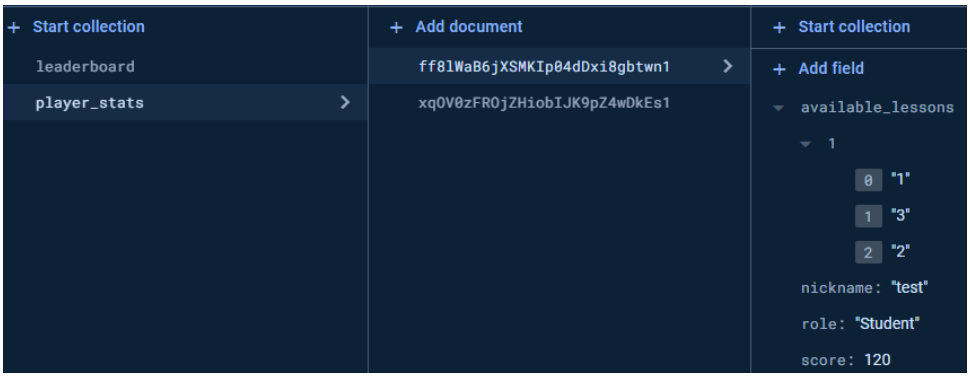

Figure 7: User data stored on the server

User registration requires filling out a form, including entering a nickname, choosing a role (teacher or student), and entering an email address and password (see Figure 8). After registration/login on a particular device, user data is stored locally and used automatically the next time the game is launched.

Figure 8: User sign-up screen

An example of a method for sending user data to the server is the _send_account_data() function (see Listing 3), which sends data during registration. The principle of operation is as follows: first, the user is verified, then the data from the completed form is collected and arranged, and finally sent to the server.

Listing 3: The function of sending data to the server (Firestore)

```
func _send_account_data():
        var auth = Firebase.Auth.auth
        if auth.localid:
                var collection: FirestoreCollection =
        Firebase.Firestore.collection(COLLECTION_ID)
                var nickname = %NicknameLineEdit.text
                var role_id = %RoleSelectionEdit.selected
                var role
                match role_id:
                        0:
                                role = "Student"
                        1:
                                role = "Teacher"
                var data: Dictionary = {
                        "nickname": nickname,
                        "role": role,
                        "available_lessons": {"1": ["1"]}
                }
                var task: FirestoreTask =
                    collection.update(auth.localid, data)
```

The virtual operating system implemented in the project is called PandaOS. It is a virtual operating system designed to provide an interactive user experience. It is based on a design of visual 'Control' nodes. The PandaOS interface displays visual elements similar to those in the Windows operating system (see Figure 9 and Figure 10). This includes windows, buttons, icons and other controls, allowing users without additional knowledge to use it using familiar interfaces.

Figure 9: Basic PandaOS interface

The critical function of PandaOS is to act as a bridge between the 3D game world and mini-games implemented as separate applications or games for this operating system. This allows seamless mini-game integration into the main gameplay, providing users additional features and entertainment. In addition to the basic features, PandaOS offers several additional options for customising the system. Users can change the background and colour scheme of the system to suit their preferences. The system also has a clock that displays real-time and date, which adds even more realism and ease of use. In

Figure 10: Changed PandaOS interface

Student Simulator, one of the mini-games offered is the Bamboo+ application, where you can easily acquire basic programming skills using the Bamboo+ visual programming language of the same name.

The program can be divided into 5 main blocks such as (see Figure 11 and 12)

- Top bar for programme management;
- Right menu of lessons for selection of available lessons;
- Left 'theoretical' panel contains theory for the current lesson (article with images);
- Area for building an algorithm;
- Console is the area of data output using the print function block.

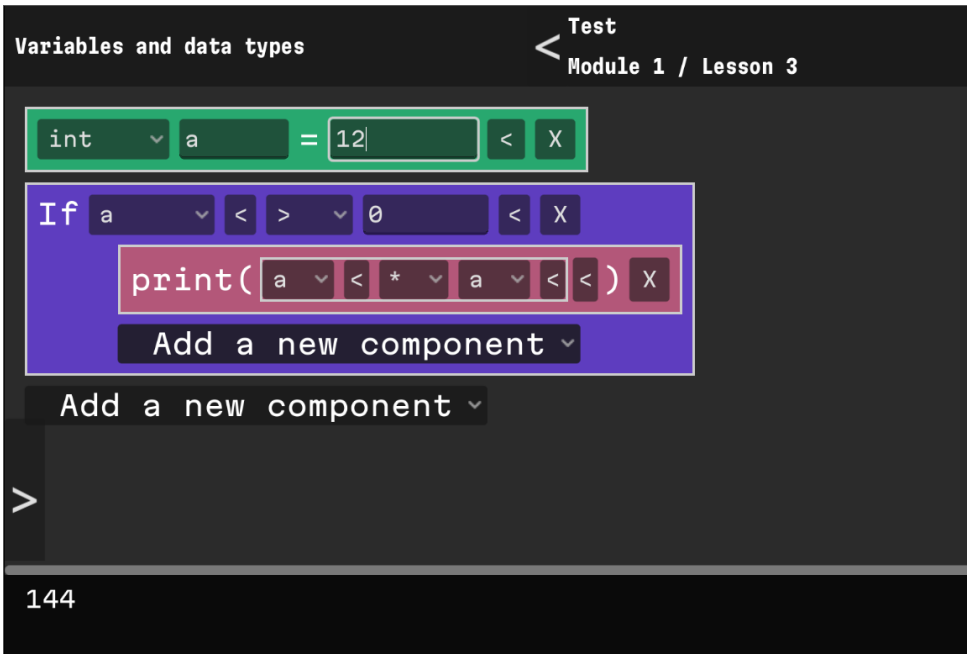

Figure 11: Bamboo+ interface with codeblocks and console

The lesson theory, available units for work and correct answers for checking are loaded into the lesson from the local JSON database. Data on which lessons are available to the user and the lesson's progress are stored in the Firestore database.

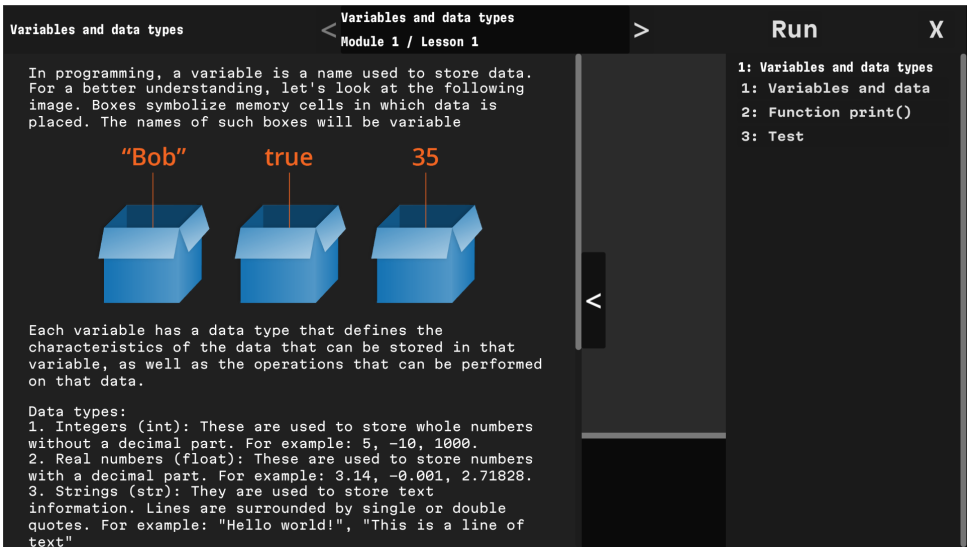

Figure 12: Bamboo+ interface with theoretical matherial

Each block of the Bamboo+ programming language is a separate scene with its script and functionality. When you click on the 'Run' button, the program execution starts. The code is read gradually from top to bottom. Each block contains a run_block function executed when the queue reaches its block (see Listing 4).

Listing 4: An example of calling the run_block function (initialising a variable)

```
func run_block():
        var auth = Firebase.Auth.auth
        if auth.localid:
                var collection: FirestoreCollection =
        Firebase.Firestore.collection(COLLECTION_ID)
                var nickname = %NicknameLineEdit.text
                var role_id = %RoleSelectionEdit.selected
                var role
                match role_id:
                        0:
                                role = "Student"
                        1:
                                role = "Teacher"
                var data: Dictionary = {
                        "nickname": nickname,
                        "role": role,
                        "available_lessons": {"1": ["1"]}
                }
                var task: FirestoreTask =
                    collection.update(auth.localid, data)
```

The following blocks are implemented for programming in Bamboo+:

- Variable initialisation. Bamboo+ is strictly typed and supports such data types as integers, floats, strings, and booleans.
- Reassignment variable.
- Function initialisation.
- Call function.

- Construction 'if' and 'if-else' for branching.
- 'While' is a loop until the specified logical expression is true.
- 'For' is a loop with a local variable whose value starts from 0 and goes up to the specified value. The loop stops when the local variable becomes equal to the specified values.
- 'Print' is a function that outputs data to the console.

Another interactive mini-game was a test system using 3D models of tablets. For this purpose, we developed a scene with a tablet and an interaction system 13. The objects themselves are hidden in different locations in visible and not-so-visible places. When the player finds one, he/she can pick it up and then a tablet window will appear with a question he/she has to solve.

Figure 13: The tablet with some test

Players are awarded rating points for completing various tasks and passing mini-games. This will create healthy competition and encourage users to learn more to gain the top spot. Standard Godot control nodes were used to build scenes displaying the best players (see Figure 14). The Firebase Firestore tool was used for saving the data. The data is stored in the rating file as 'key': 'value'.

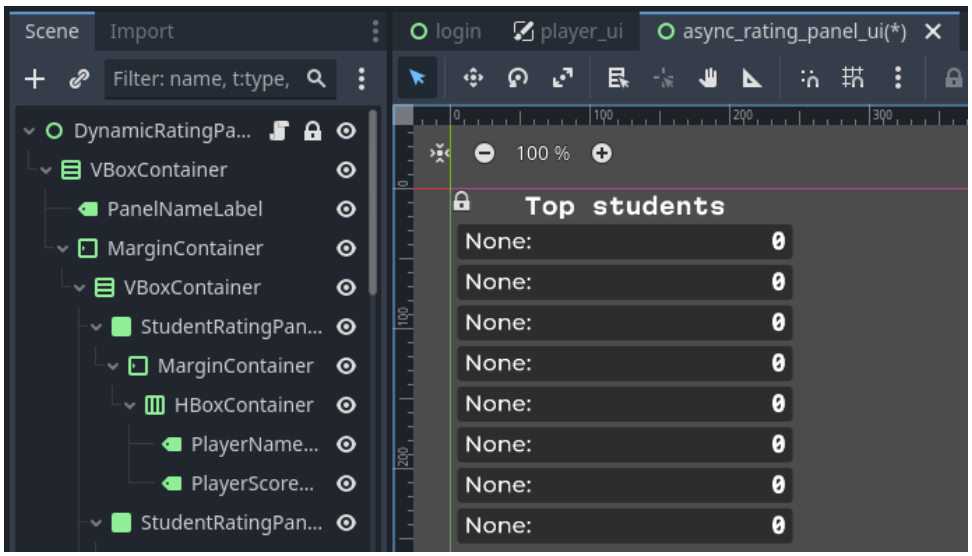

Figure 14: Scene of the rating system

4. Prospects and possible improvements to the Student Simulator game

The ability to improve the Student Simulator game opens up broad prospects for further development. One of the critical areas could be the introduction of artificial intelligence to adapt the learning process to the individual needs of each student. AI can analyse user performance, strengths, and weaknesses and automatically select tasks of the appropriate difficulty level. This will provide a personalised approach to learning and allow for more efficient achievement of goals. Implementing multiplayer and collaboration modes can make the Student Simulator even more attractive to users, allowing students to collaborate on projects and solve problems in a virtual environment. This approach will help develop team skills and increase motivation through social interaction and competition.

The integration of the game with popular learning management systems, such as MOODLE or Blackboard, is in demand among teachers. This will provide easy access to educational materials and tasks and allow teachers to track student progress and make the necessary adjustments more effectively.

An essential task of implementing the simulator is determining its effectiveness criteria and indicators. To accomplish this task, teachers should be involved as experts. A survey of students to assess the advantages and disadvantages of the developed game application is another promising area for further research.

5. Conclusions

The results of this study show that its primary objectives were met. We analysed. Modern approaches to designing, creating, and using simulator games were explored. On this basis, the sequence of game development was clarified, a matrix of its objects was made, and a model was designed. As a result of the comparative analysis, we have grounded and selected the main tools for development: Godot as a game engine, Blender as software for creating 3D graphics and Firebase as a cloud service for data storage. The chosen set of tools should consider the needs of graphic development, interaction and ease of use. The author's experience in creating the game was systematised, and its critical development points were described.

The development of 3D games, such as Student Simulator, requires the collaboration of various specialists. In our case, it was organised as a project whose participants acted as storytellers, designers, programmers, modellers, and testers. This approach encouraged teamwork and healthy competition among developers. The created Student Simulator has a modular structure at the level of locations and objects with which the player interacts. These objects are the PandaOS virtual operating system and the Bamboo+ visual programming language, which add new levels of interactivity and learning opportunities. The project participants created a website to promote the game. However, the project has prospects for further improvement, mainly through implementing AI and expanding multiplayer functions.

6. Acknowledgments

We want to express our sincere gratitude to all the students of the CS-46 (KN-46) group of Ternopil Volodymyr Hnatiuk National Pedagogical University, who are not the authors of this paper but were participants in the project and put a lot of effort into creating the Student Simulator game. Among them are Maksym Bazyvoliak, Honcharuk Maksym, Ivan Hrytsai, Vitaliy Melnychuk, Vladyslav Serpevskyi, Ivan Shovag, and Andriy Yasinskyi

7. Declaration on Generative AI

While preparing this work, authors used AI-Assisting Tools (ChatGPT, Copilot and Grammarly) to do grammar and spelling checks, reword text, and search for some primary sources. After using these tools,

the authors reviewed and edited the paper's content. They take full responsibility for the publication's content.

References

[1] M. Videnovik, T. Vold, L. Kiønig, A. Madevska Bogdanova, V. Trajkovik, Game-based learning in computer science education: a scoping literature review, International Journal of STEM Education 10 (2023) 54. doi:10.1186/s40594-023-00447-2.

[2] P.-Y. Chen, G.-J. Hwang, S.-Y. Yeh, Y.-T. Chen, T.-W. Chen, C.-H. Chien, Three decades of game-based learning in science and mathematics education: an integrated bibliometric analysis and systematic review, Journal of Computers in Education 9 (2022) 455–476. doi:10.1007/s40692-021-00210-y.

[3] Y. B. Kafai, Q. Burke, Constructionist gaming: Understanding the benefits of making games for learning, Educational Psychologist 50 (2015) 313–334. doi:10.1080/00461520.2015.1124022.

[4] S. Adipat, K. Laksana, K. Busayanon, A. Ausawasowan, B. Adipat, Engaging students in the learning process with game-based learning: The fundamental concepts, International Journal of Technology in Education 4 (2021) 542–552. doi:10.46328/ijte.169.

[5] S. O. Semerikov, M. V. Foki, D. S. Shepiliev, M. M. Mintii, I. S. Mintii, O. H. Kuzminska, Methodology for teaching development of web-based augmented reality with integrated machine learning models, in: Proceedings of the 11th Illia O. Teplytskyi Workshop on Computer Simulation in Education (CoSinE 2024), volume 3820, 2024, p. 118 – 145. URL: https://ceur-ws.org/Vol-3820/paper249.pdf.

[6] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, A. Anandkumar, Minedojo: Building open-ended embodied agents with internet-scale knowledge, 2022. doi:10.48550/ARXIV.2206.08853.

[7] T. Vakaliuk, V. Kontsedailo, D. Antoniuk, O. Korotun, S. Semerikov, I. Mintii, Using game dev tycoon to develop professional soft competencies for future engineers-programmers, in: Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer., volume 2732, 2020, p. 808 – 822. URL: https://ceur-ws.org/Vol-2732/20200808.pdf.

[8] M. B. Armstrong, R. N. Landers, An evaluation of gamified training: Using narrative to improve reactions and learning, Simulation & Gaming 48 (2017) 513–538. doi:10.1177/1046878117703749.

[9] K. N. Wilson, B. Ghansah, P. Ananga, S. O. Oppong, W. K. Essibu, E. K. Essibu, Exploring the efficacy of computer games as a pedagogical tool for teaching and learning programming: A systematic review, Education and Information Technologies (2024). doi:10.1007/s10639-024-13005-2.

[10] R. L. Lamb, L. Annetta, J. Firestone, E. Etopio, A meta-analysis with examination of moderators of student cognition, affect, and learning outcomes while using serious educational games, serious games, and simulations, Computers in Human Behavior 80 (2018) 158–167. doi:10.1016/j.chb.2017.10.040.

[11] A. M. Toda, A. C. T. Klock, W. Oliveira, P. T. Palomino, L. Rodrigues, L. Shi, I. Bittencourt, I. Gasparini, S. Isotani, A. I. Cristea, Analysing gamification elements in educational environments using an existing gamification taxonomy, Smart Learning Environments 6 (2019). doi:10.1186/s40561-019-0106-1.

[12] V. Oleksiuk, D. Verbovetskyi, I. Hrytsai, Design and development of a game application for learning python, in: Proceedings of the 6th Workshop for Young Scientists in Computer Science & Software Engineering (CS&SE@SW 2023), volume 3662, 2024, p. 111 – 124.

[13] S. Basu, S. Saha, S. Das, R. Guha, J. Mukherjee, M. Mahadevappa, Assessment of attention and working memory among young adults using computer games, Journal of Ambient Intelligence and Humanized Computing 14 (2022) 2413–2428. doi:10.1007/s12652-022-04494-5.

[14] S. Cass, Some assembly (language) required - three games that make low-level coding fun, IEEE Spectrum 54 (2017) 19–20. doi:10.1109/mspec.2017.7906890.

[15] S. Sipone, V. Abella, M. Rojo, J. L. Moura, Sustainable mobility learning: Technological acceptance model for gamified experience with classcraft in primary school, Education and Information Technologies 28 (2023) 16177–16200. doi:10.1007/s10639-023-11851-0.

[16] L. Parody, J. Santos, L. A. Trujillo-Cayado, M. Ceballos, Gamification in engineering education: The use of classcraft platform to improve motivation and academic performance, Applied Sciences 12 (2022) 11832. doi:10.3390/app122211832.

[17] A. Riabko, T. Vakaliuk, O. Zaika, R. Kukharchuk, I. Novitska, Gamification method using Minecraft for training future teachers of computer science, in: Proceedings of the 3rd Workshop on Digital Transformation of Education (DigiTransfEd 2024), volume 3771, 2024, p. 22 – 35. URL: https://ceur-ws.org/Vol-3771/paper26.pdf.

[18] V. Peters, M. V. D. Westelaken, Simulation games - a concise introduction to the design process, Samenspraak Advies, 2014. doi:10.13140/2.1.4259.1367.

[19] A. Striuk, O. Rybalchenko, S. Bilashenko, Development and using of a virtual laboratory to study the graph algorithms for bachelors of software engineering, in: Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer, volume 2732, 2020, p. 974 – 983.

[20] O. Holovnia, V. Oleksiuk, Selecting cloud computing software for a virtual online laboratory supporting the operating systems course, in: Proceedings of the 9th Workshop on Cloud Technologies in Education (CTE 2021), volume 3085, 2021, p. 216 – 227.

[21] G. C. Ullmann, C. Politowski, Y.-G. Guéhéneuc, F. Petrillo, Game Engine Comparative Anatomy, Springer International Publishing, 2022, pp. 103–111. doi:10.1007/978-3-031-20212-4_8.

[22] J. Holfeld, On the relevance of the godot engine in the indie game development industry (2024). doi:10.48550/ARXIV.2401.01909.

[23] T. Salmela, Game development using the open-source Godot Game Engine, Tampere University of Applied Sciences, 2022. URL: http://www.theseus.fi/handle/10024/746943.

[24] C. Perez, J. Veron, F. Perez, A. Moraga, C. Calero, C. Cetina, A comparative analysis of energy consumption between the widespread unreal and unity video game engines (2024). doi:10.48550/ARXIV.2402.06346.

[25] M. McPherson, Blender vs 3ds max | head-to-head comparison (2023), 2023. URL: https://www.designbuckle.com/blender-vs-3ds-max/.

[26] Y. Hendriyani, V. A. Amrizal, The comparison between 3d studio max and blender based on software qualities, Journal of Physics: Conference Series 1387 (2019) 012030. doi:10.1088/1742-6596/1387/1/012030.

[27] V. P. Oleksiuk, O. R. Oleksiuk, Methodology of teaching cloud technologies to future computer science teachers, in: Proceedings of the 7th Workshop on Cloud Technologies in Education (CTE 2019), volume 2643, 2020, p. 592 – 608. URL: https://ceur-ws.org/Vol-2643/paper35.pdf.

[28] A. Z. Ayezabu, Supabase vs Firebase: Evaluation of performance and development of Progressive Web Apps, Tampere University of Applied Sciences, 2022. URL: http://www.theseus.fi/handle/10024/771009.

[29] R. Jain, Firebase vs Supabase, 2024. URL: https://dev.to/codeparrot/firebase-vs-supabase-4770.