UNIVERSITY OF TARTU
Faculty of social sciences
Narva college
“Information Technology Systems Development” Curriculum

Robert Efros

Development of the interactive game for learning Typescript

Bachelor’s thesis (9 ECTS)

Supervisor(s):
Andre Sääsk, M.Sc.

Narva 2024

Table of Contents

Terms and Abbreviations

Code Typing - a process of assigning results of the code operations as predefined immutable data types.

Compilation - in programming, a process of translating a programming language into instructions that the computer can understand.

Sprint - in agile methodology, a predefined period of time, within which a set of tasks must be accomplished.

Sprint board - a type of organizational chart, that contains all of the tasks within the scope of a sprint, divided by their completion state.

Game design - a process of creating rules, mechanics and systems of a game

Gameplay loop - repetitive nature of the game

API - or Application Programming interface is a set of rules that allows two different applications to communicate with each other

REST - a popular architecture for APIs

Rendering - The process of generation of two or three dimensional image

UI - User Interface

BBCode - or Bulletin Board Code, a lightweight markup language

HTTP - or Hypertext Transfer Protocol, is a format of data transfer

Entity AI - A simple set of instructions defining behavior of entities, that are not controlled by the player

Sprites - Two dimensional image represented within a game environment

QoL - Quality of Life, a term used to describe parts of the system, created for user convenience

Introduction

In current times, rapid technological advancements have propelled the evolution of web development, generating a surge in the popularity of STEM fields. This surge has manifested notably in the significant increase of individuals eager to join the workforce as web developers, drawn by the prevalence and visual appeal of this field, as web development is a source for a majority of systems people interact with daily. However, this surge has also underscored a pressing need: the demand for accessible learning materials. As the number of aspiring web developers increases, there arises a necessity to develop and disseminate educational resources that cater to diverse learning needs and levels of expertise. Addressing this problem becomes important in preparation of a skilled workforce capable of leveraging technological advancements to create new systems used by untold billions daily.

One of the most popular web development languages JavaScript, used in the majority of web based systems, is rarely used in its pure form as of today, since in the base configuration javascript lacks features for code typing and debugging, as well as many more. Although there are a lot of languages that compile into JavaScript like CoffeeScript and Ruby, one certain superset of JavaScript seems to have acquired most market share and love of developers and corporations alike, TypeScript.

Problem

The problem is that with the rise of demand for learning resources, not enough entry level resources provide interactive, engaging and entertaining ways of education. For a person starting their journey as a JavaScript/TypeScript developer, it is very easy to get bored or overwhelmed with an amount of new information, while practicing and applying skills they accumulate isn’t alway as straightforward as many would like it to be. Often passing paid courses online, learning from free online tutorials or sometimes even getting a specialized education is not enough to conceptualize knowledge into an applicable skill, which can be done through practice alone. This results in many people abandoning the idea of becoming a software developer.

Second problem is that often in professional studies only basic JavaScript is being taught, which introduces difficulties of switching from basic JavaScript to the more popular and featured TypeScript superset.

Solution

The Author's solution to those problems is gamification of TypeScript learning, by creating a computer game which requires the player to write TypeScript code in order to achieve goals within it. By achieving goals, gradually increasing in difficulty, the user will seamlessly learn key principles of writing, reading, and improving TypeScript code. With gradual introduction of more difficult goals and immediate visual representation of users' code effects, the learning experience would be improved and made more entertaining and application of newly learned skills would be more apparent to the player. Additionally, by combining both educational and entertainment qualities, this solution could also be considered as a standalone product for a general puzzle game audience in the computer gaming market.

Goal

The Author’s goal is to develop a game system that will allow them to easily expand it with new content, and continually improve it with end-user feedback in mind. This is important, due to the educational nature of the game, during the free end-user testing phase, players will share inconsistencies in the learning process and propose changes to the learning materials

and structure in which those are presented, the system should be flexible enough, to allow for quick changes.

The initial system, outlined in this document, should include several working components and features:

- Dynamic level system - a system that will allow the Author to add more content to the game quickly, following a general template.
- Dynamic level guide system - each level should include a related way to communicate educational content to the player, and help them navigate through the challenges. This communication should be provided in text and image form.
- Visual rendering system - a system that will visualize player actions, providing live feedback based on player's inputs
- Simple user interface - a system that will allow the player to navigate the game, and interact with it
- TypeScript translation layer - a system that will interpret TypeScript code as game instructions, that will be handled by the game, and represented using a rendering system.
- Basic objects and entities - a small set of components, that will represent objects and entities manipulated by the player to achieve goals or manipulated by the game, in order to present harder challenges to the player.

Tasks

In order to achieve set goals, three main tasks have been outlined by the Author:

- Research - current state of market regarding code learning applications on the market, and technologies that could be used to achieve outlined goals should be researched, and based on the result of this research, some of the researched technologies should be picked and used.
- Learning - picked technologies that are unfamiliar to the Author should be learned and practiced with, before the implementation of this project.
- Development - after sufficient research and technology learning is done, the project should be implemented.

1 Research

1.1 Market Research:

Analyzation process of current gamified TypeScript education market.

1.1.1 Checkio

One of the accessible gamified solutions exclusive to TypeScript learning, is checkio (https://js.checkio.org/).

This solution represents a web platform, where users are invited to progress through a number of pre-defined or user-created sections, designed to teach typescript principles starting from most basic ones, by defining a problem, and allowing a user to solve it within a code editor at the bottom of the screen. Users are prompted to type code into the code editor field and test it by pressing a button. The result of users actions will be outputted as a message below the code editor, representing a simple console log. Although a good solution, Authors project goals include creating a more visualized and entertaining form of education, in a more common video game form, which is different in principle to the scope and philosophy of checkio.

1.1.2 CodinGame

CodinGame is a broader web based solution, that is providing users with a set of puzzles or miniature games, that are similar in concept to the author's project (https://www.codingame.com/home).

CodinGame also features a visual representation of player input, and does allow for usage of typescript code in most of their puzzles. CodinGame's solution has several differences to the one outlined in this document, notably, CodinGames Authors concentrate on teaching general concepts instead of specific languages, which allows them to include a big variety of programming languages to solve puzzles with. Additionally, CodinGames is more of a platform for hosting different kinds of puzzle games, instead of having a game with a single theme or story, which does work better for visual variety, but Author believes that it could make continued support more difficult, as completely new puzzles need to be made for new challanges. The presence of multiplayer modes for competition between players combined with a big focus on representing only key concepts with single player puzzles leads Author to believe that CodinGame's Authors first and foremost aimed at the competitive multiplayer option of their game.

1.1.3 Code Combat

CodeCombat is an educational product, represented by a web based application, that among other things, features a code learning game where the player is tasked with controlling a character in order to accomplish goals, to collect more abilities for said characters, to accomplish more difficult goals in a progressive loop. Although very similar to the Author's idea, it does not provide the ability to write TypeScript language.

1.2 Aim

The aim of the project is to create a game that would provide goals to a user, achievable through writing TypeScript code. The game should visualize users actions to provide feedback, containing positive or negative reinforcement. In order to achieve that, the game

should be separated into two main components, those being the game itself written using GoDot script within GoDot engine, containing all the means of interaction and visualization of user actions and a TypeScript based node service running in development mode, capable of accommodating live code changes, that will interpret user inputs as commands and return instructions for the game to visualize. This will allow the player to write actual typescript code in order to interact with a game.

1.3 Technology research

1.3.1 Existing technology

Unity Engine

Unity's real-time 3D development engine lets artists, designers, and developers collaborate to create amazing immersive and interactive experiences. You can work on Windows, Mac, and Linux (Unity, 2024)[1].

Hasn't been chosen due to the controversy regarding Unity's distribution policy.

Games made with Unity would be charged a fee for downloads after certain thresholds were passed (GameRant, Trey Griffeth, October 8, 2023)[2]

Author does not support such monetization practices and doesn't feel comfortable knowing that Unity's developers might change their monetization this way again.

Unreal Engine

Unreal Engine enables game developers and creators across industries to realize next-generation real-time 3D content and experiences with greater freedom, fidelity, and flexibility than ever before. (Unreal Engine, 2024)[3]

One of the most known graphical engines for video games and filmmaking. Although incredibly powerful and well-developed, it is centered around 3D graphics, and games made with this engine are generally more difficult to run, which would needlessly limit the amount of possible users. Additionally, it is much more difficult to learn, compared to engines like Godot and Rpg Maker.

Rpg Maker

RPG Maker™ XP gives you the power to create your own original RPG on Windows. Its popular and user-friendly interface has been carried over from RPG MAKER 2000, and its graphic capabilities, battle screen layout, and data packaging features are better than ever! By popular demand, this latest installment also contains a brand-new Scripting function. RPG MAKER XP is perfect for beginners and experts alike. (RpgMaker, 2024)[4]

A simple to use game engine, designed to make role-playing video games. Its major use-case is out of scope of the described project.

---

[1] https://unity.com/products/unity-engine
[2] https://gamerant.com/unity-ceo-john-riccitiello-stepping-down-pricing-controversy/
[3] https://www.unrealengine.com/en-US/unreal-engine-5
[4] https://www.rpgmakerweb.com/products/rpg-maker-xp

Adobe Photoshop

Start with Photoshop. Amazing will follow. With Photoshop and generative AI, you can create gorgeous photos, rich graphics, and incredible art. (Adobe, 2024)[5]

An industry standard application for graphics editing and creation. Hasn't been chosen due to high subscription cost.

1.3.2 Chosen technology

GIMP

GIMP is an acronym for GNU Image Manipulation Program. It is a freely distributed program for such tasks as photo retouching, image composition and image authoring. (GIMP, 2024)[6]

Was chosen due to being free, familiar to Author, and including all the necessary tools for the project

Pixilart

A safe social platform for everyone. Create beautiful pixel art, share, collaborate, shop and more! (Pixilart, 2024)[7]

Pixilart is a web based solution that, besides social functions, also contains a very convenient and simple to use pixel art based editor. Was chosen due to simplicity and an abundance of training materials.

Godot

Godot Engine is a feature-packed, cross-platform game engine to create 2D and 3D games from a unified interface. It provides a comprehensive set of common tools, so that users can focus on making games without having to reinvent the wheel. Games can be exported with one click to a number of platforms, including the major desktop platforms (Linux, macOS, Windows), mobile platforms (Android, iOS), as well as Web-based platforms and consoles (Godot Engine, 2024)[8]

Godot engine was chosen due to its ease of use, good documentation, open-source nature, ability to write code both in Godot Script and C# languages on demand, and ability to export projects into a big variety of platforms.

Godot Script

GDScript is a high level, dynamically typed programming language used to create content. It uses a syntax similar to Python (blocks are indent-based and many keywords are similar). Its goal is to be optimized for and tightly integrated with Godot Engine, allowing great flexibility for content creation and integration. (Godot Engine, 2024)[9]

Comes included with Godot engine and is well documented and convenient. Expected to be used for the majority of the logical elements within the project.

---

[5] https://www.adobe.com/products/photoshop.html
[6] https://www.gimp.org/about/
[7] https://www.pixilart.com/about
[8] https://docs.godotengine.org/en/stable/about/introduction.html
[9] https://docs.godotengine.org/en/3.0/getting_started/scripting/gdscript/gdscript_basics.html#doc-gdscript

C#

C# is a modern, innovative, open-source, cross-platform object-oriented programming language and one of the top 5 programming languages on GitHub.(Microsoft, 2024)[10]

In this project, Godot compatibility with C# is used to interact with a system on a low level, since C# includes a plethora of tools designed to interact with the system, like opening command line applications and overtaking their outputs.

JavaScript

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area. (Mozilla, 2024)[11]

JavaScript is the main scripting language used for web development, it is being expanded by TypeScript and still remains at its core, as all of the TypeScript code is being compiled into JavaScript before execution.

TypeScript

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale. (TypeScriptLang, 2024)[12]

TypeScript was chosen due to its popularity, and Authors familiarity with the superset.

Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment. (Nodejs, 2024)[13]

In order to run an API using JavaScript compiled from TypeScript, a runtime environment capable of interacting with the operating system is required. Nodejs has been chosen for this purpose as one of the most popular and documented runtime environments for Javascript.

Express.js

Express.js is a fast, unopinionated, minimalist web framework for Node.js (Expressjs, 2024)[14]

In order to run the Node service as an API capable of listening to HTTP requests, a web framework is required. Express.js was chosen for this purpose for its documentation, popularity, and Authors working experience with it.

[10] https://dotnet.microsoft.com/en-us/languages/csharp
[11] https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
[12] https://www.typescriptlang.org/
[13] https://nodejs.org/en
[14] https://expressjs.com/

2 Development

This section describes game development process

2.1 Non-Functional requirements

The TypeScript interactive code learning game should meet following non-functional requirements:

- Accessible on personal computers running Windows operating systems
- Contain simple easy to use graphic interface for user to navigate through it's systems
- Contain attractive graphic visualization of all actions performed by the user

2.2 Functional requirements

The TypeScript interactive code learning game should meet following functional requirements:

- Game could accept user input
- Game could react to user input
- Game could visualize user actions
- Game had interactable user interface
- Game allowed user to choose from a number of challenges
- Game included a node service based on TypeScript
- Game included goals, obstacles, and entities for the player to control, interact or reach
- Game included playing field to render goals, obstacles and entities in
- Game provided means of writing code into the typescript files inside node service
- Game interpreted player-written code as instructions to control the entities within the playing field
- Game provided written feedback on results of user-written code execution
- Game provided learning materials and hints for user
- Game initiated node service without additional user interaction
- Game communicated with node service to receive instructions based on user-written code

2.3 Planning and work organization

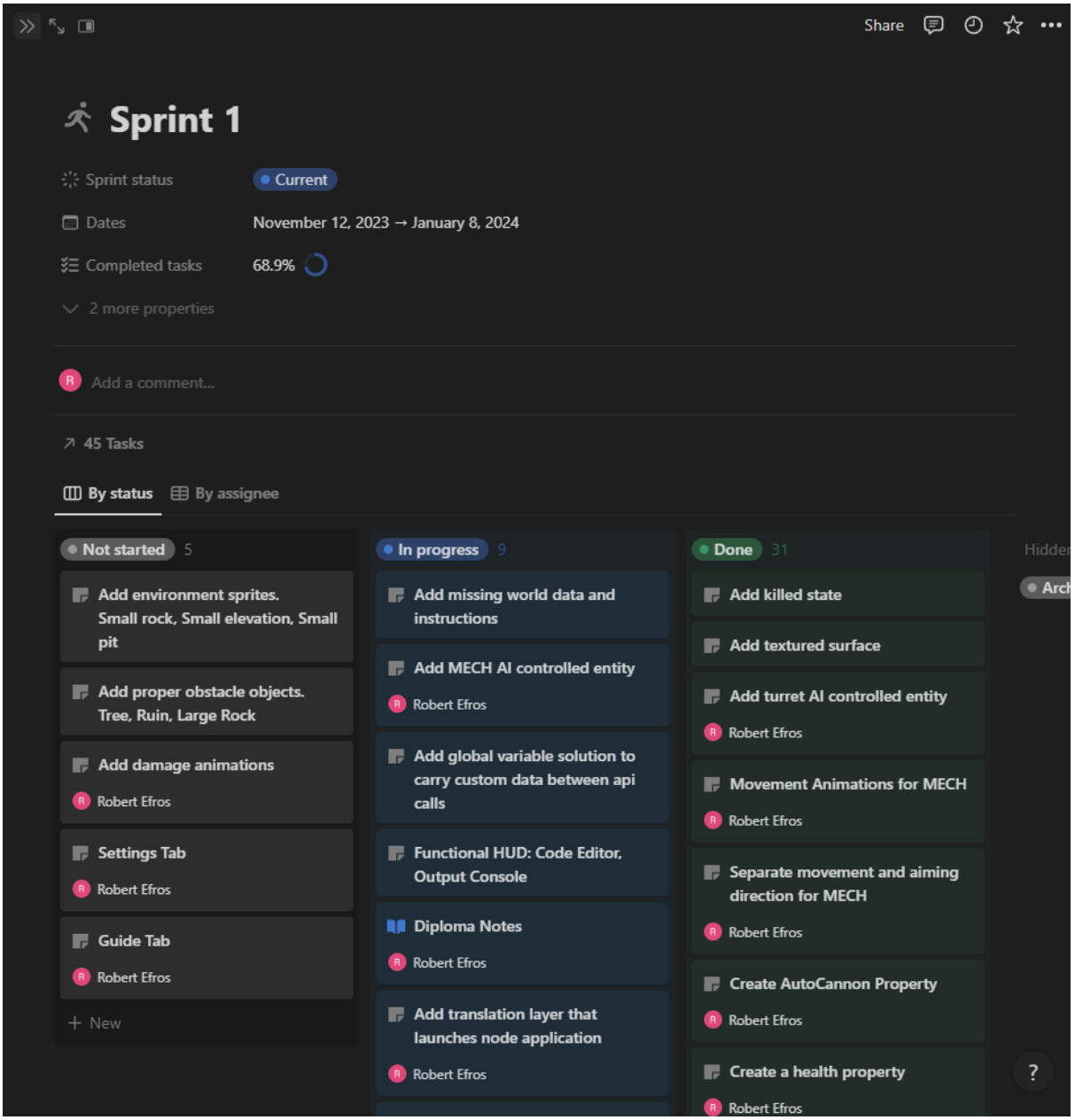

Figure 1. Author's first sprint in Notion snapshot. Taken by Author

With the size and scope of the project, keeping track of project scope, timeframes, system development scopes and groomed tasks is crucial. For those purposes web based application Notion was used in order to create sprints and sprint boards with system scopes containing tasks relative to sprints. All the work for the first demonstration version of the product outlined in this thesis, have been done within a first sprint lasting from November 12, 2023 to January 8th, 2024. **Figure 1**.

2.4 Use cases

The solution should accommodate following use cases:

1. Navigate to level
   1.1. Open the game
   1.2. Press “Play” button in the “Main Menu” view

1.3. “Scenario Picker” view should appear to the right
1.4. Press a button with any level name on the “Scenario Picker” view
1.5. “Game Renderer” view should appear containing level picked by a user

2. Open guide view
   2.1. Open the game
   2.2. Press “Guide” Button in the “Main Menu” view
3. Run written TypeScript code
   3.1. Open the game
   3.2. Press “Play” button in the “Main Menu” view
   3.3. “Scenario Picker” view should appear to the right
   3.4. Press a button with any level name on the “Scenario Picker” view
   3.5. Write code into the text editor in the bottom panel
   3.6. Press “Run” button represented by a “Play” icon on the right side of the bottom panel
4. Exit back to main menu from finishing screen
   4.1. Open the game
   4.2. Open game
   4.3. Press “Play” button in the “Main Menu” view
   4.4. “Scenario Picker” view should appear to the right
   4.5. Press a button with any level name on the “Scenario Picker” view
   4.6. “Game Renderer” view should appear containing level picked by a user
   4.7. Write code that will accomplish given goal into the text editor in the bottom panel
   4.8. Press “Run” button represented by a “Play” icon on the right side of the bottom panel
   4.9. Field with level above the bottom panel should cover with a “Level End View”
   4.10. Press “Back to menu” button on “Level End View”
5. Exit back to main menu from level
   5.1. Open game
   5.2. Press “Play” button in the “Main Menu” view
   5.3. “Scenario Picker” view should appear to the right
   5.4. Press a button with any level name on the “Scenario Picker” view
   5.5. “Game Renderer” view should appear containing level picked by a user
   5.6. Press “Back” button represented by a green arrow
6. Exit the game
   6.1. Open game
   6.2. Press “Exit” button in the “Main Menu” view

2.5 Game design

The solution is a game, and though technical requirements as “Game included goals, obstacles, and entities for the player to control, interact or reach” are simple, in reality it entails designing a gameplay loop that would keep the player entertained.

“A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome.” (Katie Salen, Eric Zimmerman, Rules of Play: Game Design Fundamentals, 2003) According to the latter definition of games, the solution must contain rules, artificial conflict and quantifiable outcomes.

According to requirements, all interaction between a player and the game should be provided using typescript code, which creates the first rule of the game - all interaction resulting in

achievement of results, must be committed using written code. Now that the main game rule is established, the quantifiable results must be defined. On top level we could say that in order to reinforce the learning process, the ultimate goal should be for the player-written code, upon execution, to accomplish requirements defined by the game. To define what requirements would the game prompt the player to accomplish, a theme of the game must be chosen first.

As computer games are a form of interactive media, with a visual component, in order to represent the goals and means to achieve them, visual design decisions had to be made. Author chose a style of two dimensional isometric game, where the player would have to control characters using code, in order to achieve goals. Creative portion of devising a make-believe theming for the game, resulted in a theme of walking robots being programmed by the player to achieve set goals. According to them, initial goals for the game have been chosen, and those are ‘reach destination’ and ‘disable opponents’. In order to create a form of inner conflict, the game should contain entities and obstacles that would hinder players progress, in order to push players to overcome them. Entities and obstacles can be represented according to previously established themes, such as buildings, rocks and stones, terrain, hostile robots, and points of interest that should be reached to accomplish ‘reach destination’ goal, represented as a distinct colored node, for ease of visual identification by the player.

As a result, the solution represents a game, where the player takes the role of a programmer, writing code for robots, that will allow those robots to achieve tasks and goals based on environment and hazards, represented by static objects and computer controlled robots.

2.6 Flow diagram

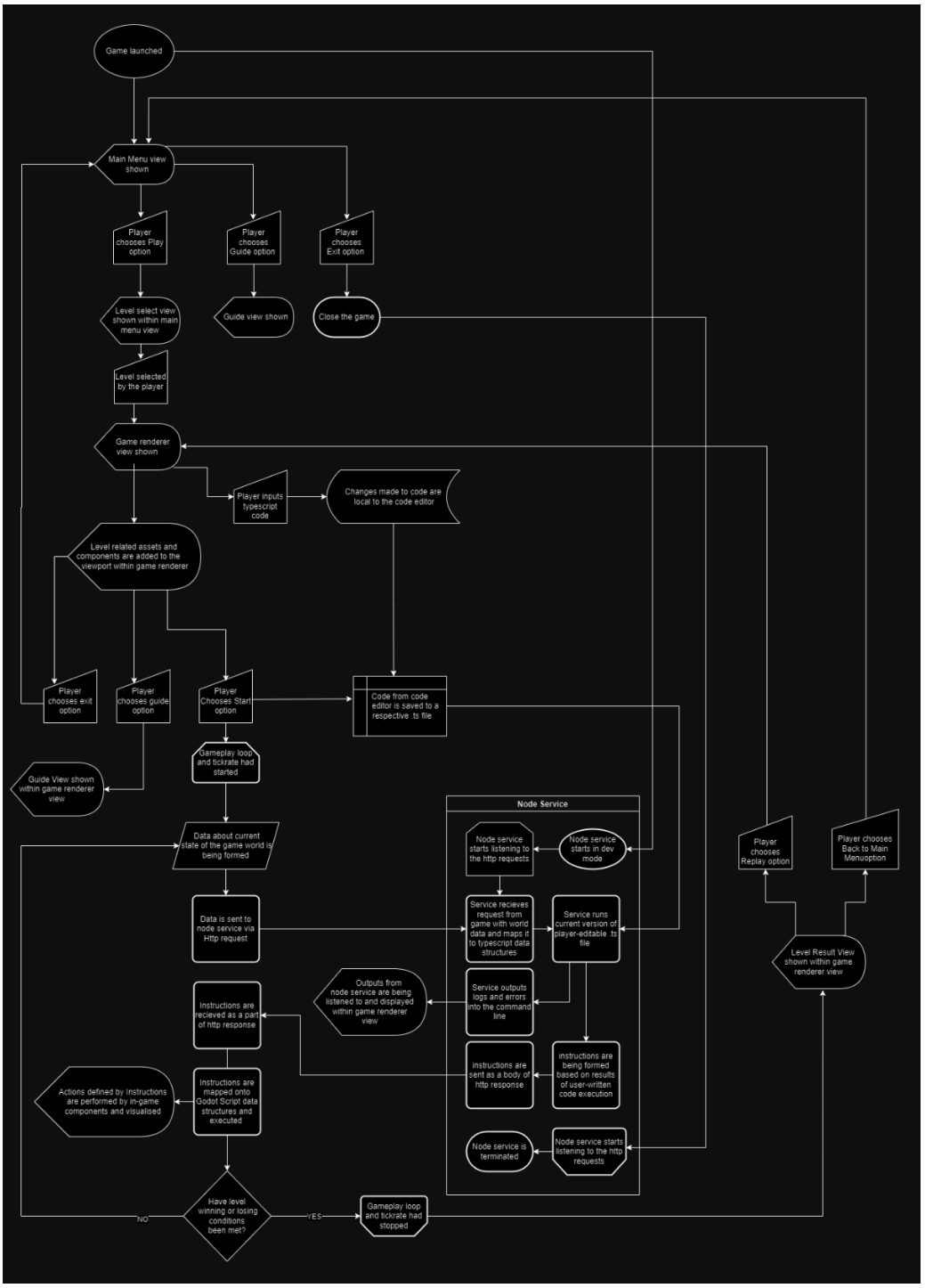

Figure 2. The solutions flow chart. Made by Author.

Flow diagram displayed on **Figure 2** outlines expected flow user interactions and application behavior.

2.7 Game components

The solution consists of three main components:

- Assets
- Godot game

- TypeScript node service

Assets

A collection of graphical files like images, that Godot game will use during rendering. Made by the Author using web application pixil.art and free image editing software GIMP. Stored within asset directory and represented by .png image format.

Godot game

Made in Godot Engine using Godot Script and .NET C# programming languages, this component handles rendering of graphics, defining playing fields, objects and entities within them, their properties and functions, human interaction and computer controlled entities' behavior and communication with TypeScript Node service using JSON format.

TypeScript node service

Written in TypeScript and node.js, made API service with express.js, and following RESTful design, is used for execution of player-written code, listening to Http requests from Godot game, translation of both incoming game data from Godot game and instructions generated by player-written code execution from and into JSON format.

3 Implemented Solution

This section describes results of TypeScript interactive learning game development results

3.1 Main menu

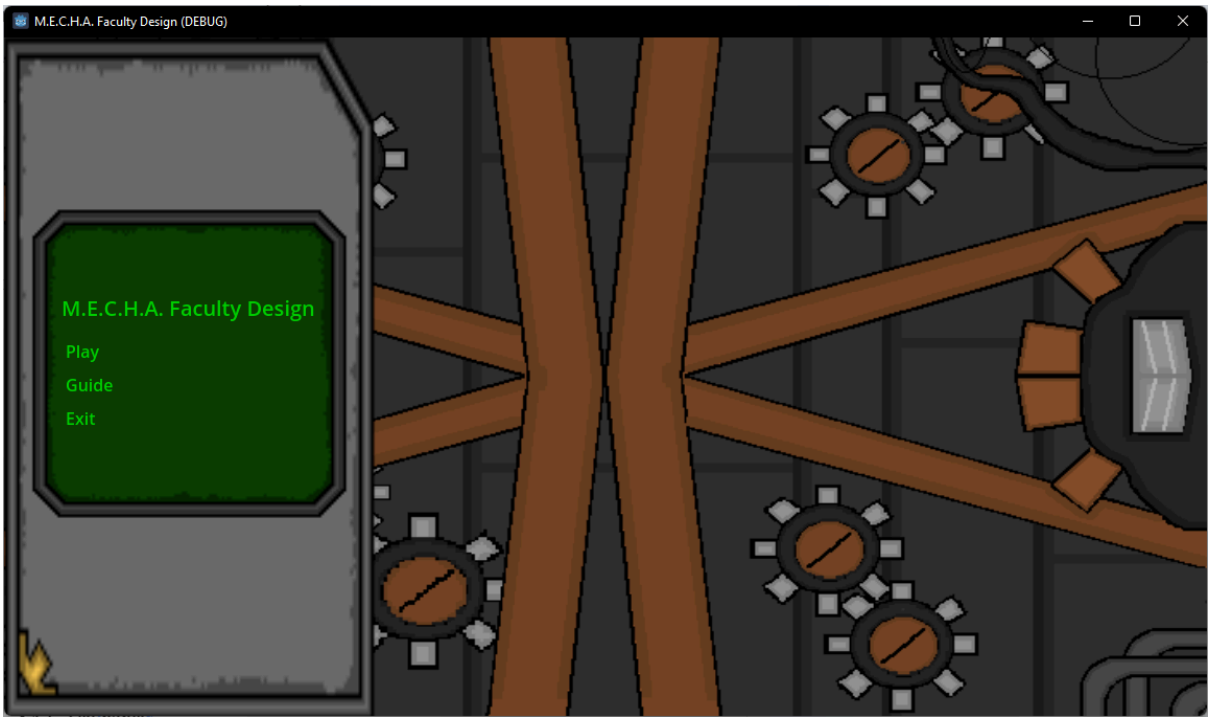

Figure 3. Main menu component. Screenshot made by Author.

As displayed in **Figure 3** above, the main menu component is the first component the player sees after booting the game. It serves the purpose of navigating players to different other components. It contains 3 options:

- Play - open the Level picker component
- Guide - open the guide component
- Exit - verify players choice using confirmation view, then close the game and all related processes (node application included)

It contains graphical sprites for the background and left menu panel, with sub-components being shown to the right of the menu panel. Additionally, Main Menu screen utilizes C# script in order to launch TypeScript service using Node “start dev” script, which allows for live code adjustments and keeping track of the service’s outputs.

3.2 Confirmation Dialog

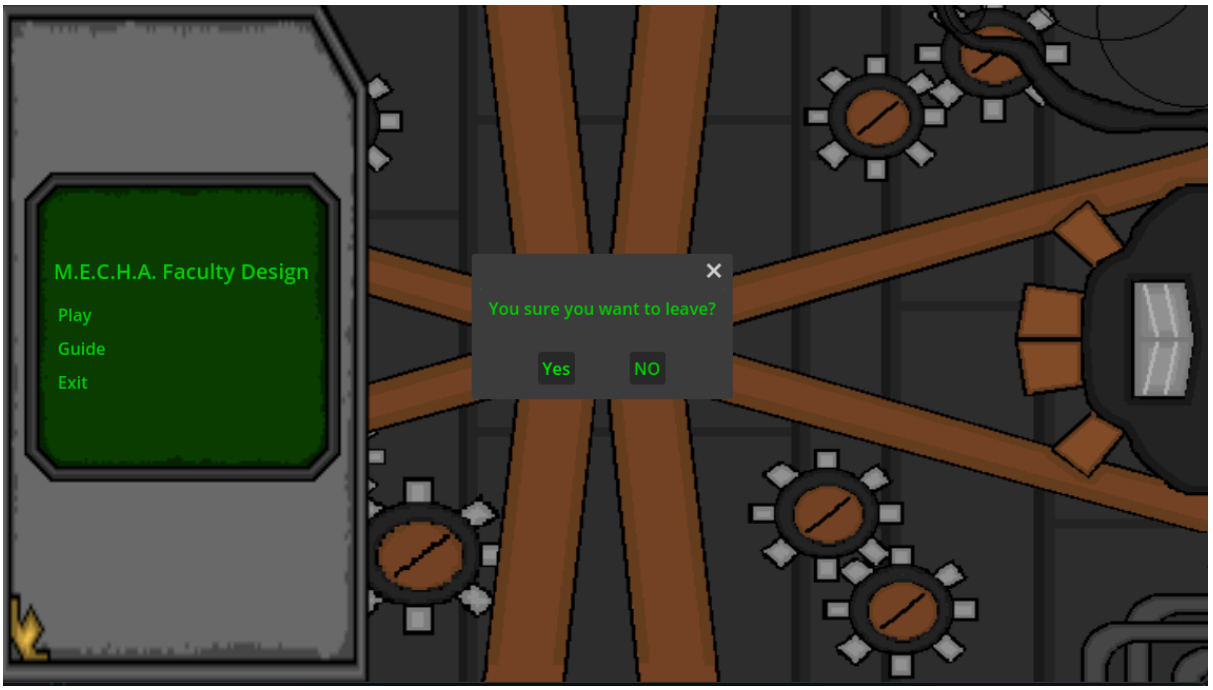

Figure 4. Confirmation dialog shown in the main menu. Screenshot made by Author.

As displayed in **Figure 4** above, confirmation dialog is a small component that allows the player to verify if they intended to close the game or not, to avoid unexpected or accidental application termination.

3.2.1 Guide

Figure 5. Guide tab shown in the main menu. Screenshot made by Author.

As displayed in **Figure 5** above, the guide view contains level guides that provide level goals, system introductions, educational materials, and tips to help players finish the level. Guide

may or may not be included into the level, and is represented by an additional component, containing the guide text, formatted with BBcode, for the level. Guide screen is filled dynamically based on whether the level contains the guide file or not. Map names are filling a list of buttons on the left, pressing on which opens corresponding guides to the right.

3.2.2 Level picker

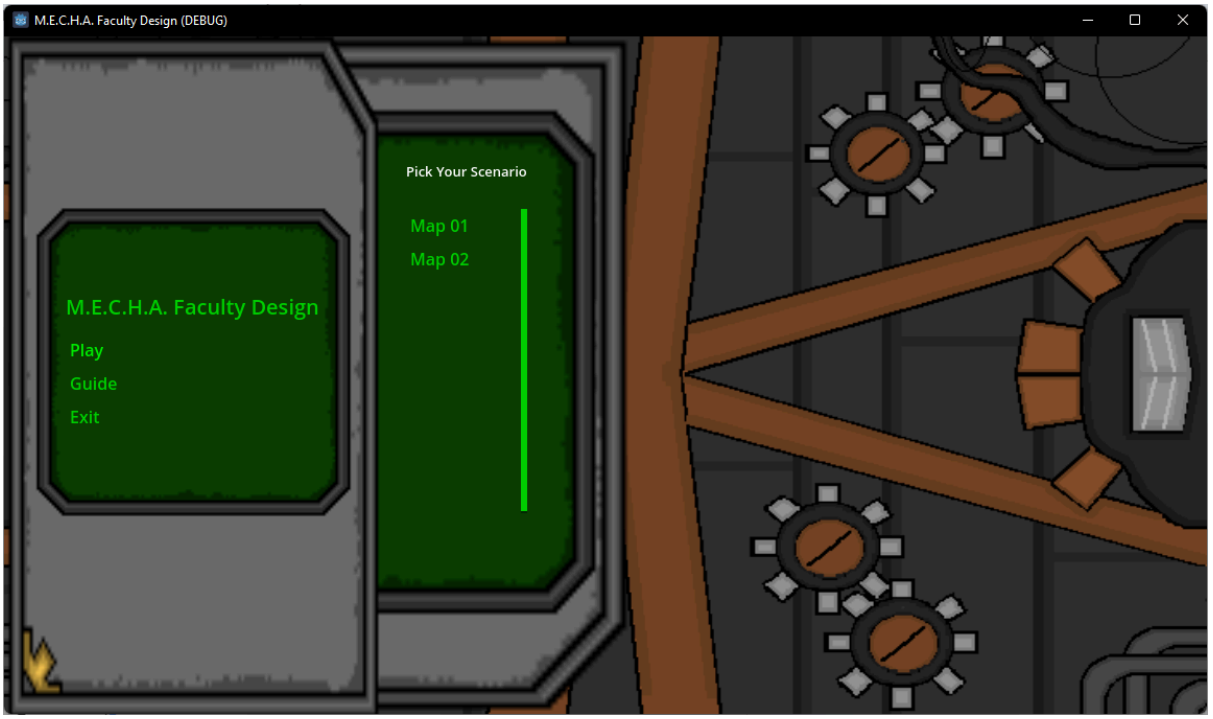

Figure 6. Level selection sub-component open within the main menu. Screenshot made by Author.

As displayed in **Figure 6** above, level picker contains a dynamically filled list of all levels contained within the project's ‘map’ folder. The game scans the games ‘map’ directory at start, filling out the list with all the levels of valid structure within the ‘map’ directory. Pressing any of the buttons in this list navigates the player to the Game render component, with a respective level rendered inside of it.

3.3 Game renderer

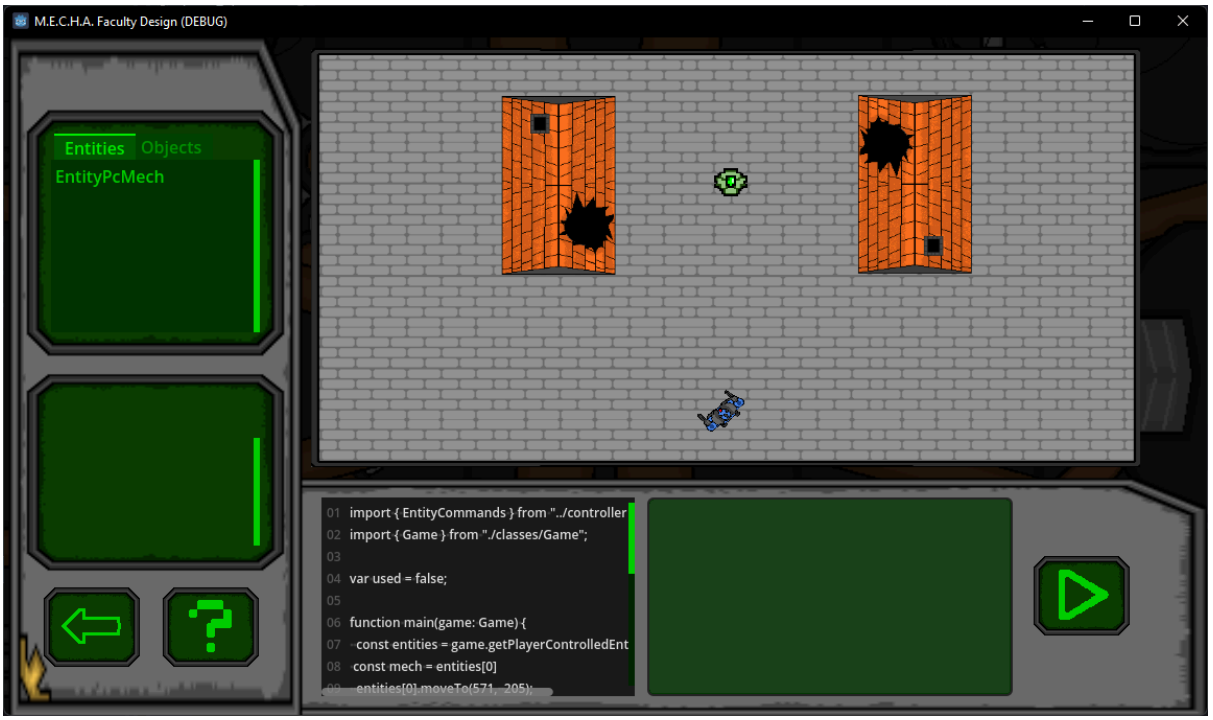

Figure 7. Game renderer component. Screenshot made by Author.

As displayed in **Figure 7** above, game renderer is a big component that is responsible for all in-game logic and functions. During start-up it renders all of its components, and prepares all of the functionalities to be initiated by the player. It contains all the components necessary for playing the game, which are:

- Left UI panel
- Bottom UI panel
- SubViewPort for map rendering
- Hidden component for data collection
- Hidden component for Http communication
- Hidden Victory Screen component

3.3.1 Left panel

Figure 8. Left panel within the game renderer view. Screenshot made by Author.

As displayed in **Figure 8** above, the left panel contains a graphical sprite, with two areas and two buttons on top of it. Top area is a list of entities and objects separated using two tabs. By pressing a tab, a list of buttons below them is replaced with a list buttons with names of objects or entities according to the player's choice.

Figure 9. Left panel with a result of pressing a button from an object list. Screenshot made by Author.

As displayed in **Figure 9** above, pressing a button in this list will fill the lower field with general data of a respective entity or object. Buttons below those two fields are the "Back" button represented by a left facing arrow icon, and the "Guide" button represented by a question mark icon. Pressing on the "Back" button will return the player to the main menu.

Figure 10. Guide shown inside of the subviewport container. Screenshot made by Author.

As displayed in **Figure 10** above, pressing the "Guide" button will open a Guide view within the SubViewPort container.

3.3.2 SubViewPort

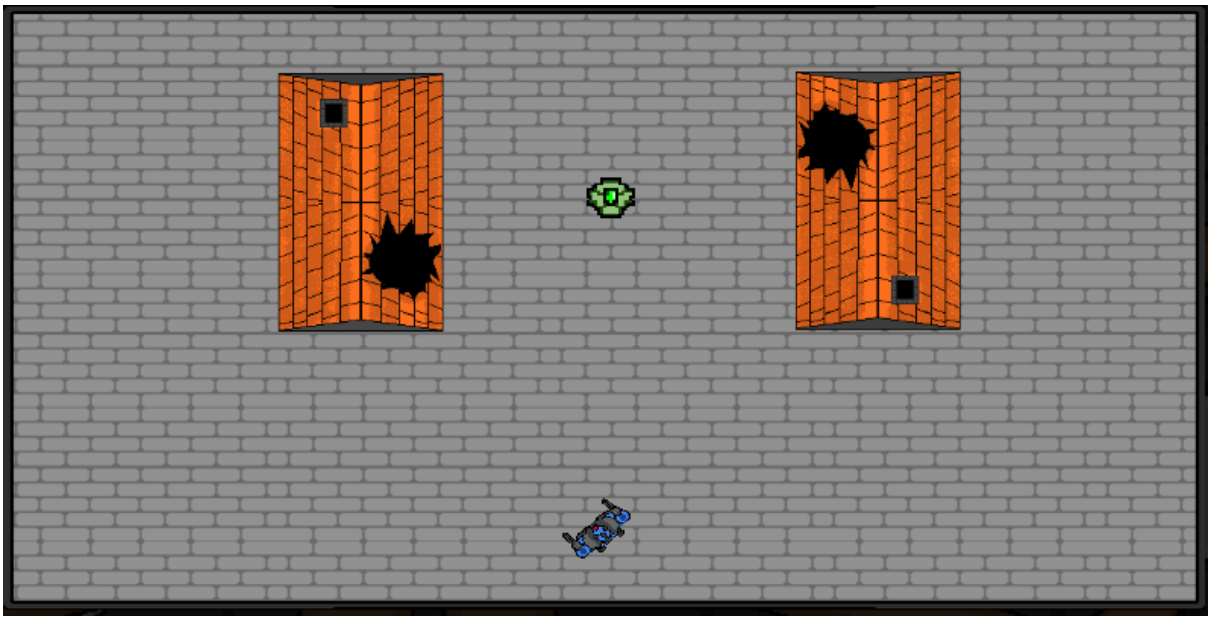

Figure 11. SubViewPort component. Screenshot made by Author.

SubViewPort container, displayed in **Figure 11** above, is a Godot provided component that is used for rendering scenes within UI structure. In this project, it is used to show the level with all of the level objects and entities, where all of the effects of user interaction are displayed.

3.3.3 Bottom panel

Figure 12. Bottom panel shown within the game renderer. Screenshot made by Author.

Bottom panel displayed in **Figure 12** above**,** is a renderer subcomponent that provides main methods of interaction with the game. It contains a simple code editor, console for logging results of typescript service's work, and a "Run" button represented by a play icon commonly seen in video, and music players. When running the parent GameRenderer component, main.ts file from typescript service's directory is automatically being opened and read, then a copy of main.ts file contents is made and inserted as text into the code editor for the player to change. When the player thinks that they are finished with code adjustment, they can press the "Run" button which will save the file, and activate the Http communication component.

3.3.4 Victory Screen

Figure 13. Victory screen component. Screenshot made by Author.

Victory Screen is an overlay subcomponent displayed on **Figure 13.** It is displayed over the level after winning or losing conditions are met. It contains a few labels with the result of the game, which can be "Success" or "Failure", name of the winning or losing condition that resulted in the game, and time spent on achieving the condition, counted from the moment the player pressed the "Run" button within the Bottom panel component. In the bottom of the component are two buttons, "Back to menu" button and "Retry" button. "Back to menu" button returns the player to the main menu, while the "Retry" button resets the current level state and allows the player to try again and improve the results of their input.

3.4 Maps or Levels

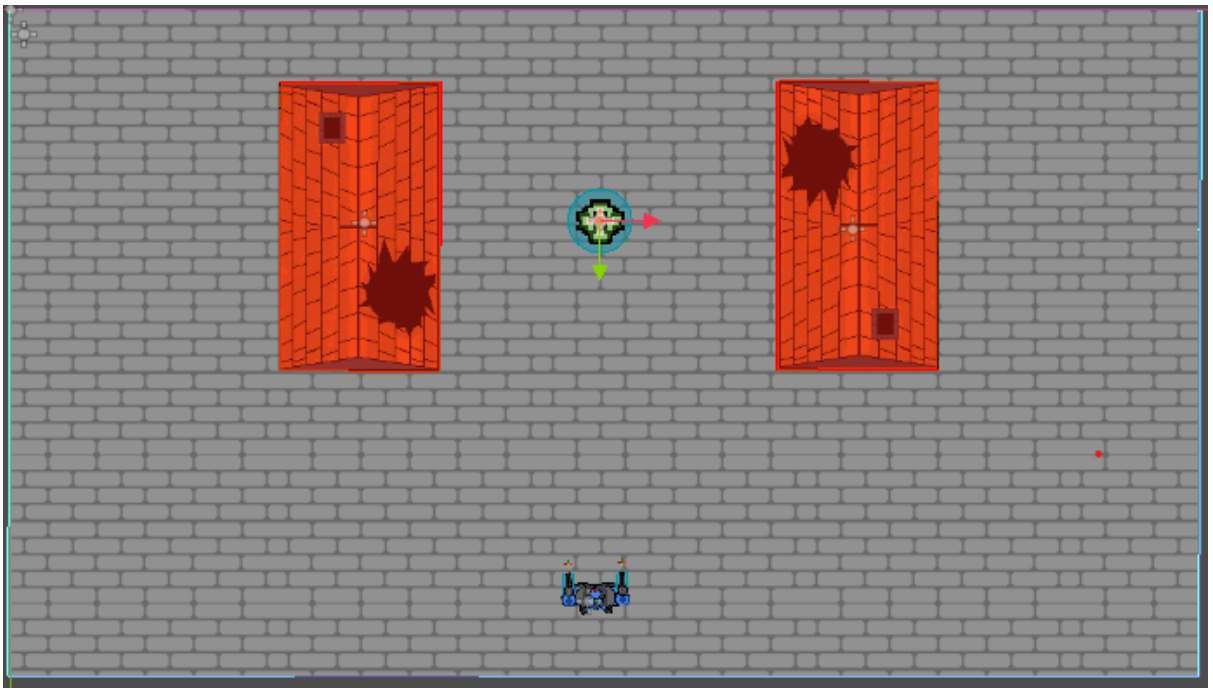

Figure 14. Map within the editor window. Screenshot made by Author.

Maps or Levels, displayed on **Figure 14** inside of the Godot editor window, are components that represent the playing field that the player interacts with. Levels can be filled with different combinations of objects, entities and goals resulting in different scenarios and challenges for players to overcome. The structure of a single level contains:

- Navigation region component
- Data collection component
- Http communication component
- Timer component

When the player presses the “Run” button during gameplay, the timer component is started with a configurable timeout set in seconds. If player instructions fail to accomplish any of the winning conditions during that time frame, losing conditions will be enforced.

As the component is initialized, all of its parts are prepared and a Navigation server is initiated with the navigation region passed into it. Navigation server is a provided functionality within Godot, that is capable of using AStar algorithm with adjustments for objects contained within the component, in order to provide powerful pathfinding functionalities to the game.

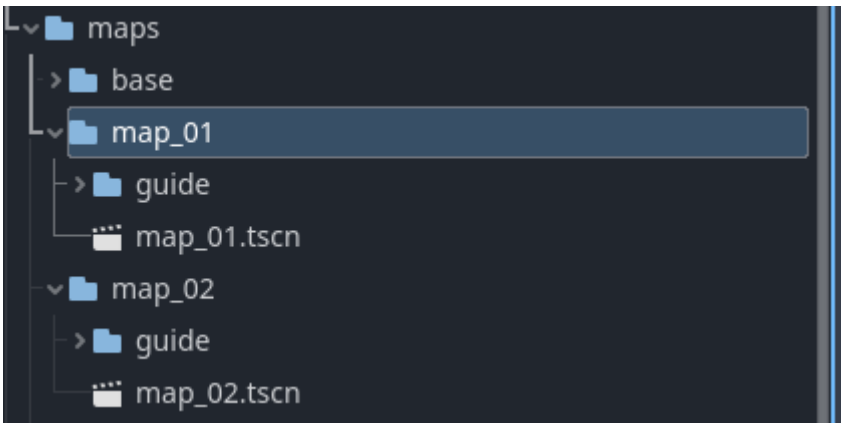

Figure 15. Map or level directory within Godot editor. Screenshot made by Author.

Levels can be added on demand by copying by following the internal structure of the scene, and directory structure displayed on **Figure 15**. By creating a new directory within maps directory and naming the main .tscn file of the map same as the parent folder, new levels can be added on demand within minutes. As outlined in the Main menu's Level selector and Guide sections before, those maps will be dynamically added to the game, and will be accessible to the player.

3.4.1 Navigation Region

Navigation region is a Godot provided component, that can be defined using 2D polygonal structure, that defines a region within which all of the placed objects will be accounted for during the pathfinding process.

3.4.2 Data collection component

Data collection component is a functional sub-component of game renderer that does not have visual representation, but contains a script that monitors and distributes states of all the objects and entities within a level. It is the component responsible for AI's situational awareness, data displayed in game renderer component's left panel, and deciding whether or not victory or failure conditions were met.

3.4.3 Http communication component

Http communication component is a functional sub-component of game renderer that does not have visual representation, but contains a script that once initiated communicates every second with TypeScript service using POST HTTP requests. It collects, formats and fills request forms with data describing the current state of the level; it expects the instructions from the service containing instructions generated from player-written code. It then executes the instructions by sending corresponding signals to the level component.

3.5 Entities

Base Entities that the project includes. Entities are split into two main categories:

- Player controlled - are entities that contain functionalities designed to respond to player interaction.
- Game AI or computer controlled - are entities that are controlled by the game AI, using simple scripts that allow them to react to player actions.

All entities consist of several reusable sub-components that can be mixed and matched in order to achieve desired results on the playing field. All entities have names that they can get referred to by the player.

- Collider - a Godot provided component, that defines a simplified computed shape of the visual component. Collider in mech type enemies represents a rectangle, and limits the movement of the mech by restricting movement through other colliders and is responsible for detecting enemy projectiles hitting the entity

3.5.1 Entity sub-components

Each entity includes reusable sub-components that partially define its abilities. There are currently two reusable components for entities:

Health - is a sub-component responsible for keeping track of and managing health points associated with a parent entity. It is capable of reducing, adding, or reporting current health points for the entity. It is also responsible for reporting when a parent entity runs out of health points. It can be adjusted to have different values per parent entity

Autocannon - is a sub-component, representing a weapon that is capable of spawning projectiles in front of it, that would be given velocity upon appearing and speed away forward from the component, if another entity's collider is hit by a projectile, a method called 'hit' of the respective entity or object will be called, if said entity or object has said method. It can be adjusted to have different fire rates per parent entity.

Collider - a Godot provided component that can be defined by a simple or complex shape, that acts as an area of an entity of an object, that cannot be traversed. Also it reports whenever any order object or entity touches it, allowing visual interaction between components.

3.5.2 Entity type: Mech

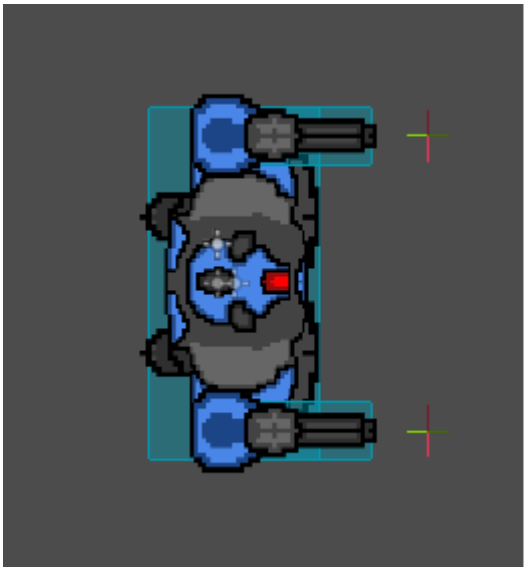

Figure 16. A mech type entity inside the Godot editor window. Screenshot made by the author.

Mech type entities displayed in **Figure 16**, are entities that are meant to represent battle robots, the game theme is centered around. Those are the player controlled entities that consist of several components, controlled by a single script:

- Health component

- Weapon components - in standard configuration, represented by two Autocannon sub-components
- Animated legs - Are a simple component that represents two sprites, animated using simple keyframe animation, which is initiated during the entity's movement. Always follows the direction of movement
- Visual base - A simple visual separator between animated legs and a top part of an entity, that is meant to visually create an illusion, that the robot has some sort of a pelvic region, instead of floating above two moving legs.
- Top part - An upper part of a robot that can rotate independently from the direction of movement.
- Collider

Mech type entities are capable of movement, facing a coordinate point on the playing field and firing their weapons, all of this functionality are controlled with instructions, provided by the player by writing TypeScript code. Additionally, the entity is setup, to react to being hit by projectiles by reducing its own health points and disabling its own function, and tinting itself gray whenever their health point count reaches zero. Disabling of all player controlled entities will enforce a losing condition.

3.5.3 Entity type: Turret

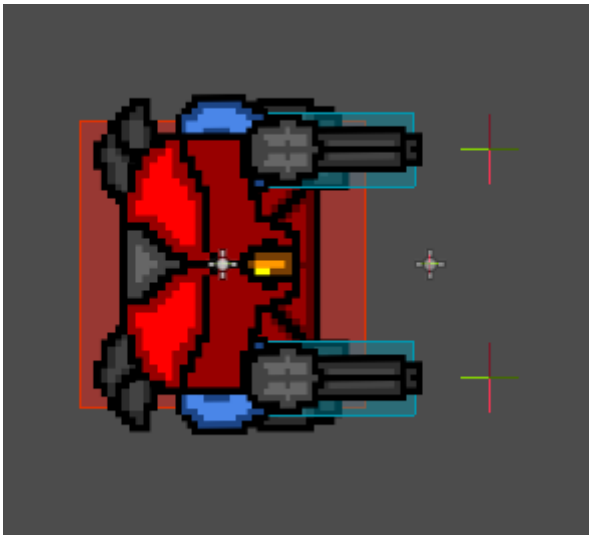

Figure 17. A turret type entity inside the Godot editor window. Screenshot made by the author.

Turret type entities displayed in **Figure 17** are game AI controlled entities that are meant to represent stationary enemies as an active obstacle to the player. Turret type entities consist for several components controlled by a single script:

- Lower part - a static visual part, located at the center base of the entity. Serves purpose of representing a stationary emplacement, upon which the top part is mounted
- Top part - a rotating visual part, located at the center top of the entity. Serves purpose of representing direction of aim of the entity
- Health - entity sub-component
- Weapon components - in default configuration, represented by two Autocannon sub-components
- Collider - entity sub-component

Turret type entities are stationary entities that will rotate their top part towards a closest player controlled entity within a configurable range per level, and will start firing their cannons after a short delay. Delay is added to give player controlled entities a window of time to react to the threat. Utilizes health components in the same way as a mech type entity. Will also get disabled and tinted gray upon reaching zero health points. Disabling all enemy entities within a map with a set goal of disabling all entities, will enforce a winning condition.

3.6 Objects

In this project, Objects are visual components, meant to represent static objects contained within the playing field. There are two types of objects currently implemented:

- Obstruction type
- Goal node type

All objects have names that they can get referred to by the player.

3.6.1 Object type: Obstruction

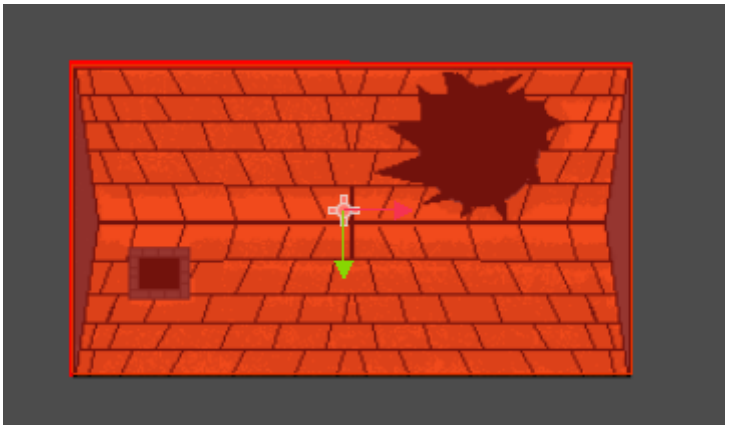

Figure 18. An obstruction type object within Godot editor. Screenshot made by Author.

Obstruction objects are objects consisting of a static sprite and a simple rectangular collider displayed in **Figure 18**. They are meant to block off paths in levels, to allow for pathfinding challenges. Their visual design represents a damaged building.

3.6.2 Object-type: Goal Node

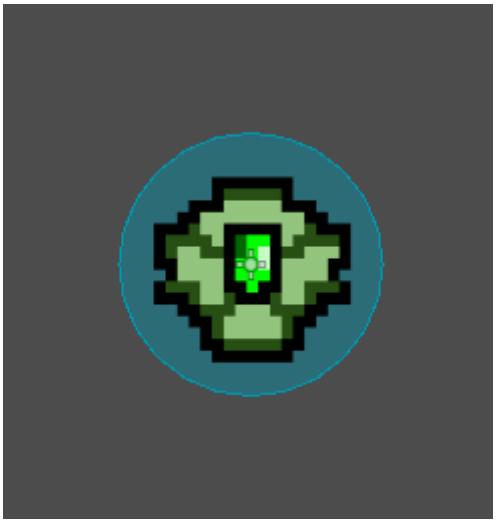

Figure 19. A goal node within Godot editor. Screenshot made by Author.

A goal nodes are objects consisting of a static sprite and Godot provided area component displayed in **Figure 19**. Area components represent areas with a set radius that upon contact will emit a signal that could be used by the game to react to. In case of goal nodes, if any of the player controlled entities touches a goal node, a victory condition is met.

3.7 TypeScript Node Service

TypeScript node service is run at the start of the game in development mode, which allows for live code changes to be represented during the game's operation without any additional actions needed from either game or the player. Using the Express.js package, the TypeScript application is continuously running in the background, listening to the endpoint under the address 'http://localhost:6060/game', which expects information about current state of the game world in JSON format, validates the contents of received data, runs player-written code located in the main.ts file and returns resulting instructions as a response back to the game in JSON format.

During that process, the service maps incoming JSON data into several classes that are meant to be used by the player in order to generate instructions for the game to interpret:

- Game
- World
- World Object
- Entity

3.7.1 Game class

Game class can be used by the player to provide the player world data represented by a world class and entity data represented by an array of entities on the level, represented by instances of Entity class by calling respective methods. Additionally Game class also automatically used to form and provide instructions for player controlled entities that player-written code generates.
Game class implements several QoL features, like ability to get specifically data of player-controlled entities only or to get an entity by its name using small respective method calls

3.7.2 World class

World class can be used by the player to provide an array of all objects on the level represented by instances of WorldObject class, object data by object name or get level width and height in pixels by calling individual respective methods.

3.7.3 World Object class

World Object class can be used by the player to provide objects name, position in x,y coordinates, and width and height in pixels by calling individual respective methods.

3.7.4 Entity class

Entity class can be used by the player to provide entities name, current position represented by coordinate points, end movement destination represented by coordinate points, current path towards destination represented by an array of coordinate points, type, player control status by their respective methods. Additionally for player controlled entities, players are allowed to set movement coordinates that will be converted into instructions to move respective entities within the level, request path towards a point, rotate entity body towards a

coordinate or open fire with weapons by calling respective methods. All of the instructions are recorded and then converted into instructions by the Game class.

3.8 Future Development

In this section, plans for future development and requirements for end-user testing will be outlined. As the current version serves as a technological base for the future product, containing important tools and building blocks, it still requires to be filled with more content in order to provide a more full package for the end-user to see and try.

Those requirements are:

- Higher Level amount - for the purposes of the current version, only few levels were finished and prepared for the demonstration, in order to provide the end-user with better learning and playing experience, more levels should be constructed out of existing and potentially new components.
- Mech AI controlled entity - Mech ai controlled entity that is planned to be represented within more difficult levels of the game, was supposed to be nearly identical to a player controlled mech, besides being controlled by game AI and hostile to the player.
- Drone AI/player controlled entity - Drone type entity is planned to be a flying entity controlled by either player or Game AI that would be capable of moving over any colliders, and attacking enemy entities, but was unable to accomplish goals like moving into the area of a goal node.
- Guide content quality improvement - for the purposes of the current version, guide sections were filled with basic information in order to showcase the main way of communicating educational content to the player. For end-user release, those guide sections should be filled with more in-depth information, better paced, and potentially include educational/training videos to make learning the coding language easier. All of the components necessary for this are already included, and can be used to achieve this goal.
- Settings view - Not included into the current version. End-user testing version should include it, in order to allow users to accommodate different screen resolutions, allow users to choose between full screen and windowed modes, and turn on or off certain features.
- Visual design improvement - Due to the Author's inexperience with digital art and pixel art, all of the art in the game is poorly scaled and stretched. This should be improved, using better drawing techniques and better planning, commissioning artistic assets to the professional digital art designer is also under consideration.
- UI implementation improvement - For the current version, the default resolution of the game is low, and the UI is not prepared for bigger resolutions. Bigger screen resolution should be accommodated up to 4K.
- Proper deployment method - Current version can only be launched using Godot’s build tool. A proper executable should be prepared and packaged in order to provide an end product to the end user in a simple form

Author believes that it is possible to achieve those requirements within the scope of the next sprint, targeting user-end testing release for the second quarter of 2024.

Summary

This topic was initially chosen for two reasons, the first reason is sufficiently outlined in the introductory portion of this document. Learning any programming language can be a daunting task, and a lot of people are giving up only needing a little bit of help to continue going, and becoming great developers. The second reason is the Author's passion for game development and digital media. Author notes that having an opportunity to develop a game that does not only have a potential as a source of entertainment, but also as an educational outlet that has potential to help countless people in future, is extraordinary.

Godot is a great tool that Author had enjoyed learning from scratch through the project's development. Author also did enjoy drawing simple images soon to become assets in pixel art style and refining ideas on how to make player interaction easier and more fun.

In Author's opinion, the project had reached its intended goals, and surpassed personal expectations Author had for it. The game featured simple graphics, animations, simple but convenient user interface and player interaction was realized using real TypeScript written by the player.

Created game provides all of the tools necessary for its growth. Most of the components are separate and composed, and a big effort was made to allow for the ability to dynamically add new content without needing to rewrite the underlying structure, which means, that within months the game might have dozens of levels, thoroughly explaining almost every basic aspect of software development using TypeScript language, while having a very smooth progressive difficulty to it.

And in the end, Author wants to note that scope of this thesis is not the end for the project itself. Near future plans include adding more content to the game, and distributing it for free on Itch.io or similar game distribution platform to collect user feedback and improve the game, until it will become a formidable product, that would help people to learn new things, or just have some entertainment, solving code puzzles.

Resümee

Käesoleva teema valikul oli kaks põhjust, millest esimene on nõudlus eriala järele. Programmeerimiskeele õppimine võib olla raske ning paljud, kes annavad alla, vajavad vaid natukene abi. Teine põhjus on autori kirg arvutimängude ja digitaalmeedia vastu. Autori arvates võimalus luua arvutimäng, mis pole ainult meelelahutus, vaid millel on ka hariduslik eesmärk mis võib tulevikus aidata lugematuid inimesi, on väga inspireeriv.

Godot on erakordne vahend, mille õppimist autor nautis projekti vältel. Autorile meeldis ka lihtsama piksel kunsti joonistamine ning interaktiivsuse ja õppeprotsessi rafineerimise protsess.

Autori arvamusel projekt mitte ainult saavutas oma eesmärgi, vaid ka ületas selle. Mängul on lihtne graafika, animatsioonid ja mugav interaktiivsus, mis saavutatakse mängija poolt kirjutatud Typescriptiga.

Mängu sees olevad tööriistad võimaldavad projekti kasvu. Enamik komponente on eraldatud ning on tehtud nii, et saaks elemente dünaamiliselt lisada vundamentaalse struktuurimuutusteta. See tekitab võimaluse, et mõne kuu jooksul saaks valmistada kõik vajalikud tasemed, et põhjalikult tutvustada tarkvaraarenduse fundamentaalkontsepte.

Autor soovib märkida, et antud tees pole selle projekti lõpp-punkt. Lähituleviku plaanid on mängule sisu lisamine ning selle tasuta levitamine läbi itch.io ja sarnaste mänguplatvormide. See võimaldab mängijate tagasiside kogumist ja mängu jätkusuutlikku arengut, mille tagajärjena saavad paljud inimesed kas õppida tarkvaraarendust, nautida puslede kokku panemist või lihtsalt oma meelt lahutada.

References

Katie Salen Tekinbas, Eric Zimmerman (2003, September 25) Rules of Play: GAME DESIGN FUNDAMENTALS, ISBN 9780262299930

Unity. (2024) *Unity Engine* Retrieved 21.01.2024 https://unity.com/products/unity-engine

GameRant. Trey Griffeth (2023, October 9) *Unity CEO is Stepping Down Following Pricing Controversy* Retrieved 21.01.2024 https://gamerant.com/unity-ceo-john-riccitiello-stepping-down-pricing-controversy/

Unreal Engine. (2024) *Unreal Engine 5* Retrieved 21.01.2024 https://www.unrealengine.com/en-US/unreal-engine-5

RpgMakerWeb. (2024) *Rpg Maker XP* Retrieved 21.01.2024 https://www.rpgmakerweb.com/products/rpg-maker-xp

Adobe. (2024) *Official Adobe Photoshop* Retrieved 21.01.2024 https://www.adobe.com/products/photoshop.html

GIMP. (2024) *About GIMP,* Retrieved 21.01.2024 https://www.gimp.org/about/

PixilArt. (2024) *About Pixilart,* Retrieved 21.01.2024 https://www.pixilart.com/about

Godot. (2024) *Introduction* Retrieved 08.01.2024, from https://docs.godotengine.org/en/stable/about/introduction.html

Checkio. (2024) *TypeScript coding challenges and exercises* Retrieved 08.01.2024 from https://js.checkio.org/

Godot. (2024) *GdScript reference Basics* Retrieved 08.01.2024 from https://docs.godotengine.org/en/3.0/getting_started/scripting/gdscript/gdscript_basics.html#doc-gdscript

Microsoft. (2024) *C# | Modern, open-source programming language for .NET,* Retrieved 08.01.2024 from https://dotnet.microsoft.com/en-us/languages/csharp

Mozilla. (2024) *What is JavaScript,* Retrieved 08.01.2024 from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

TypeScriptLang. (2024) *TypeScript: Javascript with syntax for types,* Retrieved 08.01.2024 from https://www.typescriptlang.org/

NodeJS. (2024) *Node.js,* Retrieved 08.01.2024 from https://nodejs.org/en

ExpressJS. (2024), Retrieved 08.01.2024 from https://expressjs.com/

License

Non-exclusive license to reproduce thesis and make thesis public

I, **Robert Efros**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
    1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
    1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Typescript interactive code learning game development,

supervised by **Andre Sääsk**,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Narva, **16.01.2024**