

Vilnius gediminas technical university

faculty of fundamental sciences

department of information technologies

Dmytro Teplov

**Bachelor thesis title in lithuanian**

**Procedural generation for diverse applications: creating a versatile 2D world map generator**

Final Bachelor Thesis

Information Technologies Study Programme, State code

Information Technolodies specialization

Study Field of Informatics

Vilnius, 2024

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Fundamentinių mokslų fakultetas

Informacinių technologijų katedra

PATVIRTINTA

Katedros vedėjo

doc. dr. Dmitrij Šešok

Dmytro Teplov

**Bachelor thesis title in lithuanian**

**Procedural generation for diverse applications: creating a versatile 2D world map generator**

Baigiamasis bakalauro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 6121BX033

? specialization

Informatikos studijų kryptis

|  |  |  |
| --- | --- | --- |
| **Vadovas** | Artūr Radzivilov |  |
|  | (Pedag. vardas, vardas, pavardė) |  |
| **Konsultantas** |  |  |
|  | (Pedag. vardas, vardas, pavardė) |  |

Vilnius, 2024

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

*Vilnius Gediminas Technical University*

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

*Faculty of Fundamental Sciences*

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

*Department of Information Technologies*

PATVIRTINA / *APROVED*

##### Katedros vedėjas / *Head of the department*

doc. dr. Dmitrij Šešok

**BAIGIAMOJO BAKALAURO DARBO UŽDUOTIS**

***ASSIGNMENT OF BACHELOR THESIS***

Vilnius

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Studentei | | Name Surname | | | |
| *For student* | |
| Baigiamojo darbo (projekto) tema: | | | Title in Lithuanian | | |
| *Bachelor thesis topic:* | | | *PROCEDURAL GENERATION FOR DIVERSE APPLICATIONS: CREATING A VERSATILE 2D WORLD MAP GENERATOR* | | |
|  | | | |  | |
| Baigiamojo darbo užbaigimo terminas | | | | Bakalauro baigiamasis darbas 2 atsiskaitymo data | |
| *The deadline for bachelor thesis* | | | | *Bachelor Thesis 2 exam date* | |
| BAIGIAMOJO DARBO (PROJEKTO) UŽDUOTIS: | | | | | |
| Assignment text in Lithuanian. | | | | | |
| *THE ASSIGNMENT OF BACHELOR THESIS:* | | | | | |
| Assignment text in English. | | | | | |
|  | | | | |  |
| Baigiamojo bakalauro darbo (projekto) konsultantai: | | | | |  |
| *The consultant(s) of bachelor thesis:* | | | | |
|  | | | | | (Pedag. vardas, vardas, pavardė / *Academic title, name, surname* ) |
| Vadovė |  | | |  |  |
| *Supervisor* |
|  | (Parašas / *Signature*) | | |  | (Pedag. vardas, vardas, pavardė / *Academic title, name, surname* ) |

There will be an annotation from mano.vilniustech.lt system (in Lithuanian). You can fill the table of add the one which will be generated in the system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vilniaus Gedimino technikos universitetas  **Fundamentinių mokslų** fakultetas  **Informacinių technologijų** katedra |  | | | |
|  | | | | |
| **Informacinių technologijų** studijų programos baigiamasis bakalauro/magistro darbas  Pavadinimas **Baigiamojo darbo pavadinimas** | | | | |
| Autorius **Vardas Pavardė** | | Vadovas pedag. vardas **Vardas Pavardė** | | |
|  | | | | |
|  |  | | **Kalba** | |
|  |  | |  | lietuvių |
|  |  | |  | anglų |
|  |  | |  |  |
| **Anotacija** | | | | |
| Baigiamajame magistro darbe nagrinėjamos investicinių projektų vertinimo problemos, vertinimo būtinumas ir tikslai, užsienio šalių patirtis šioje srityje. Išnagrinėti pagrindiniai principai, kuriais remiantis nustatomas investicinių projektų efektyvumas. Pateiktas investicinių projektų efektyvumo vertinimo klasifikavimas, išanalizuoti įvairių jo rūšių skirtumai. Projekto efektyvumas apskaičiuojamas analizuojant investicinio projekto pinigų srautus. Baigiamajame darbe nagrinėjama neapibrėžtumo bei infliacijos įtaka investicinio projekto efektyvumui. Vertinant infliacijos įtaką, išnagrinėtas dabartinių ir prognozuojamų kainų tarpusavio ryšys. Išnagrinėjus teorinius ir praktinius investicinių projektų vertinimo aspektus, pateikiamos baigiamojo darbo išvados ir siūlymai.  Darbą sudaro … dalys: įvadas, ……………………, išvados ir siūlymai, literatūros sąrašas.  Darbo apiUCis – 96 p. teksto be priedų, 24 iliustr., 16 lent., 34 bibliografiniai šaltiniai.  Atskirai pridedami darbo priedai. | | | | |
|  |  | |  | |
| **Prasminiai žodžiai:** bent 5 žodžiai | | | | |

There will be an annotation from mano.vgtu.lt system (in English).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vilnius Gediminas Technical University  **Faculty of Fundamental Science**  **Information Technology** department |  | | | |
|  | | | | |
| **Information Technology** bachelor study program final thesis  Title **Title of the thesis** | | | | |
| Author **students** **Name Surname** | | Supervisor prof. dr. **Name Surname** | | |
|  | | | | |
|  |  | | **Language** | |
|  |  | |  | Lithuanian |
|  |  | |  | English |
|  |  | |  |  |
| **Annotation** | | | | |
| Annotation text  … | | | | |
|  |  | |  | |
| **Keywords:** … | | | | |

Table of Contents

[Table of Contents 6](#_Toc165499272)

[Introduction 10](#_Toc165499273)

[Relevance of the thesis. 10](#_Toc165499274)

[Problem of the thesis 10](#_Toc165499275)

[1. Map generation algorithms and tools 11](#_Toc165499276)

[1.1 Map generation tools and technologies 12](#_Toc165499277)

[1.1.1 Inkarnate 12](#_Toc165499278)

[1.1.2 Watabou Procgen Arcana 13](#_Toc165499279)

[1.2 Algorithms 14](#_Toc165499280)

[1.2.1 Uniform point distribution algorithms over an area 15](#_Toc165499281)

[1.2.2 Bounding box calculation methods 16](#_Toc165499282)

[1.2.3 Procedural noise generation 16](#_Toc165499283)

[1.2.4 Wave function collapse 17](#_Toc165499284)

[1.2.5 Frustum culling 18](#_Toc165499285)

[1.3 Analysis of multiplatform desktop development technologies 19](#_Toc165499286)

[1.3.1 Programming Language 19](#_Toc165499287)

[1.3.2 Graphics Application Programming Interface 20](#_Toc165499288)

[1.3.3 small conclusions 21](#_Toc165499289)

[2. System Requirements Specifications 22](#_Toc165499290)

[2.1 Use case diagram 22](#_Toc165499291)

[2.2 Sequence diagrams 23](#_Toc165499292)

[2.3 Functional Requirements 24](#_Toc165499293)

[2.4 Non-functional requirements 25](#_Toc165499294)

[3. Architecture specification 26](#_Toc165499295)

[3.1 Component diagram 28](#_Toc165499296)

[3.2 Class diagram 28](#_Toc165499297)

[3.3 System description 29](#_Toc165499298)

[3.3.1 Startup 29](#_Toc165499299)

[3.3.2 Painting terrain 30](#_Toc165499300)

[3.3.3 Painting assets 32](#_Toc165499301)

[3.3.4 Saving 33](#_Toc165499302)

[3.4 UI description 33](#_Toc165499303)

[3.3 Description of practical part 35](#_Toc165499304)

[3.3.1 Setting up of the environment. 35](#_Toc165499305)

[3.3.2 Writing “Hello World” of computer graphics. 36](#_Toc165499306)

[3.3.3 Abstracting OpenGL commands into classes. 36](#_Toc165499307)

[3.3.4 Implementing UI. 36](#_Toc165499308)

[3.3.5 Developing all the predetermined features. 36](#_Toc165499309)

[3.3.6 Testing. 36](#_Toc165499310)

[4. Testing 37](#_Toc165499311)

[4.1 Manual Testing 37](#_Toc165499312)

[Conclusions 38](#_Toc165499313)

[References 40](#_Toc165499314)

[Annexes 42](#_Toc165499315)

**List of Images**

[Figure 1 Inkarnate interface 13](#_Toc165239298)

[Figure 2 Procgen Arcana interface 14](#_Toc165239299)

[Figure 3 Use Case Diagram 21](#_Toc165239300)

[Figure 4 Sequence diagram for all use cases 22](#_Toc165239301)

[Figure 5 User interface mockup 25](#_Toc165239302)

[Figure 6.2 GLFW official web site. 26](#_Toc165239303)

**List of tables**

[**Table 1.** All tables should have titles as well 13](#_heading=h.tyjcwt)

**List of abbreviations and terms**

|  |  |
| --- | --- |
| **RFID**  **NFC**  **QR Code**  **PICC** | Radio Frequency Identification  Near Field Communication  Quick Response code  Proximity Integrated Circuit Card |
| **USB**  **EEPROM**  **RAM**  **SRAM**  **…** | Universal Serial Bus  Electric erasable and programmable read-only memory  Random Access Memory  Static Random Access Memory  … |
|  |  |
|  |  |
|  |  |

Introduction

Relevance of the thesis.

In the world of in-game cartography, artists have always held the dominating power. Behind all of the world maps from iconic games like The Witcher 3: Wild Hunt, Red Dead Redemption 2 or Control there is always at least one artist, a team more likely. For small teams of game developers who could not afford to commission an artist for such task, there has always been a need for a tool that would help them achieve great results using minimal input. Additionally, artists working on map creation for video games, or even for board games, could be interested in a tool that would expedite their workflow. This tool could be used for layout prototyping or to create a finished map with their own stylized assets.

Problem of the thesis

The primary challenge this thesis aims to address is the creation of a user-friendly desktop application that enables individuals, regardless of their artistic abilities, to design stylized 2D world maps for various projects. These projects can range from board games like Dungeons and Dragons to in-game world maps for indie game studios. A significant feature of the proposed tool is the ability to upload custom assets. This allows users to create maps in any style they desire, making it particularly appealing to indie game studios that typically have at least one artist on the team capable of creating unique assets.

However, developing such a tool presents several challenges. These include ensuring ease of use for non-artists, implementing effective procedural generation algorithms for various map features, and allowing for the seamless integration of custom assets. This thesis will delve into these issues and propose solutions to overcome them.

**Objective of the thesis** – enhance the efficiency and quality of 2D map generation, for a variety of applications such as indie video game development, tabletop games, and literature by developing a procedural 2D map generator application.

**Tasks of the thesis:**

1. Analysis of existing tools for map generation and analysis of algorithms for area approximation, bounding box and uniform distribution.
2. Analysis of multiplatform desktop development technologies.
3. Design 2D map generation tool.
4. Develop and test the prototype.

1. Map generation algorithms and tools

Map creation is a historically labor-intensive endeavor. Before everything went digital, creation process involved painstakingly filling out a pieces of paper with depictions of relevant area, and even after the digitalization creating maps involves several steps that require time and thought. They are vital in numerous fields, providing a simplified and easily digestible way to represent spatial information and in some cases position of the user. Maps, be it digital or analog, are created by artists and require considerable amount of time. From that point on, the focus of this paper will be on digital 2D maps created for virtual worlds.

Currently, digital 2D map for virtual worlds creation process involves several steps. Creating initial prototype of a map is a first step and it would include rough blocking, where the artist would outline areas like continents, bodies of water, terrain types or, on the smaller scale, cities and towns, roads, forest areas and so on. Most of the time there are several prototypes carried out to figure out the layout of the map. Next step would be to flash out the artistic style of the map and intricacies regarding that. That would typically include the defining the distinct art style and producing various assets in this art style to eventually place and blend them to final canvas [2].

This paper heavily oriented at gaming industry, since it is a primal producer of virtual worlds that require maps, however the goal of the developed application is to satisfy the needs of any potential end-user requiring digital cartographic solution. A fantasy book writer and a board game developer should also be able to benefit from the developed tool.

Developing a tool that would allow to expedite parts of map creation process is not a new concept. However, this project is primarily geared towards accommodating individuals with varying levels of expertise, ranging from those with no prior experience with painting applications to seasoned, experienced artists.

There are several already existing tools and applications that help to accelerate parts or the whole process, but some are focusing too much on giving control to the user and getting too close to overwhelming them if they want to quickly create a map for their project. Others would restrict the process and make almost everything procedurally and leave only minor changes to the user, like background color change.

This is where this project would come as a solution. It will be appealing to both sides. The core mechanic will be the ability to outline the area types on the canvas by choosing an area in the user interface and paint the areas directly in the application. This 2D map generation tool will allow to convert quick brush strokes into a beautiful, stylized map regardless of user skill level. In addition to its user-friendly nature, the application will feature a comprehensive toolset designed to satisfy the needs of professionals as well. This toolset will help professionals to not feel any limitations due to the application's procedural nature, as it enables a level of fine-tuning akin to most popular painting applications.

The average independent game developer would find significant value in integrating this tool into their standard workflow. Doing so will alleviate the pressure of generating distinct maps for each project, especially due to the time and budget constraints often encountered by smaller studios.

Conversely, larger studios would also derive notable advantages from this tool since it would allow for finer adjustments and the importation of uniquely styled assets, a level of customization made feasible by the presence of dedicated 2D artists within the studio. An important aspect of the project lies in its open-source nature, signifying that individuals with sufficient resources have the capability to adapt the application to better align with their specific requirements.

1.1 Map generation tools and technologies

1.1.1 Inkarnate

Inkarnate is perhaps the most popular map creation tool. It gained popularity by focusing primarily on board game designers. Dungeons and Dragons, being by far the most widely known, is a fantasy tabletop role-playing game that invites players to create their own characters and embark on imaginary adventures, with one player serving as the Dungeon Master to guide the story and interpret the rules. It utilizes maps in rather a complicated manner: Dungeon masters often use Inkarnate to create detailed maps for their D&D campaigns. These maps can include entire worlds, specific regions, or even individual cities and villages [9].

Inkarnate’s strong suit include:

* + - * Quick asset distribution tool that allows to distribute any type of asset with brush strokes. It also allows to import up to 100 your own assets, but only for paid users.
      * Large asset library, including buildings, mountains, trees and many more.
      * The procedural terrain blending feature allows to paint terrain faster as the user is not concerned with blending between water and terrain, it happens automatically.
      * Road tool is used to create stylized paths on the map.

A screenshot of a computer

Description automatically generated

Figure 1 Inkarnate interface

While this tool may satisfy a wide range of customers, all of the more advanced features are locked by a subscription payment. The art style of the map can only be changed by subscribed customers, since the asset import is not available otherwise. The terrain blending settings, unfortunately, are not accessible to free users as well. This limitation renders the entire feature impractical, as the conspicuous brown border cannot be altered. Additionally, the procedurally generated terrain outline can only have one noise function, which results in terrain seams looking exactly the same for every map while also having delay between manipulating the terrain area and seeing the result.

1.1.2 Watabou Procgen Arcana

Watabou is another map creation tool, that is specifically designed to provide a generic, but at the same time with unique layout, map with basically no input required from the user. After launching the website user presented with a choice of map type: realm, city, district and village. These all differ in scaling and style for the map to be generated. After launching the generation of a certain map type the tool will fully generate the map and it will appear full screen for the user.

This tool is generating maps in a couple of stages starting with creating randomized graph tree, which will serve as a basis for the map. This graph is meticulously structured using equidistant points, arranged in such a manner that each point is interconnected via an edge to some of its nearest counterparts. This results in a graph that has almost cobweb-like structure, with some randomization. This graph serves as an excellent foundation for modeling towns or continents, cells that have been created by connecting points will be considered as areas and will have a certain type assigned to them. In case of towns, area can be assigned type town district, and hence will be populated by buildings. Other possible areas can be fortress, sea and just land. All of the areas are seamlessly blended together to output a fully procedurally generated map.

This approach differs from the approach in this paper and allows to create usable map without requiring any input, which is clearly advantageous for users who do not have specific preferences and for whom a generic map would be perfectly suitable.

The adjustments that the user can make to generated map include:

* + - * Adjustments to the graph using different tools like displace, pinch, bloat, rotate and so on.
      * Building arrangement recalculation. Changes the look of a specific area of buildings.
      * Reroll the labels placed on the map or change each label by hand.
      * Change the color scheme.

While these modifications may suffice for those seeking a quick and rudimentary map, they may appear incredibly restrictive to those accustomed to the high degree of flexibility offered by manual map creation. Such users will find graph structure limiting and it does not allow for curved edges and it may become an obstacle when trying to construct more advanced maps. Same goes for color adjustments as this map generator was created with a certain, plain, art style, it doesn’t support gradients and textures.

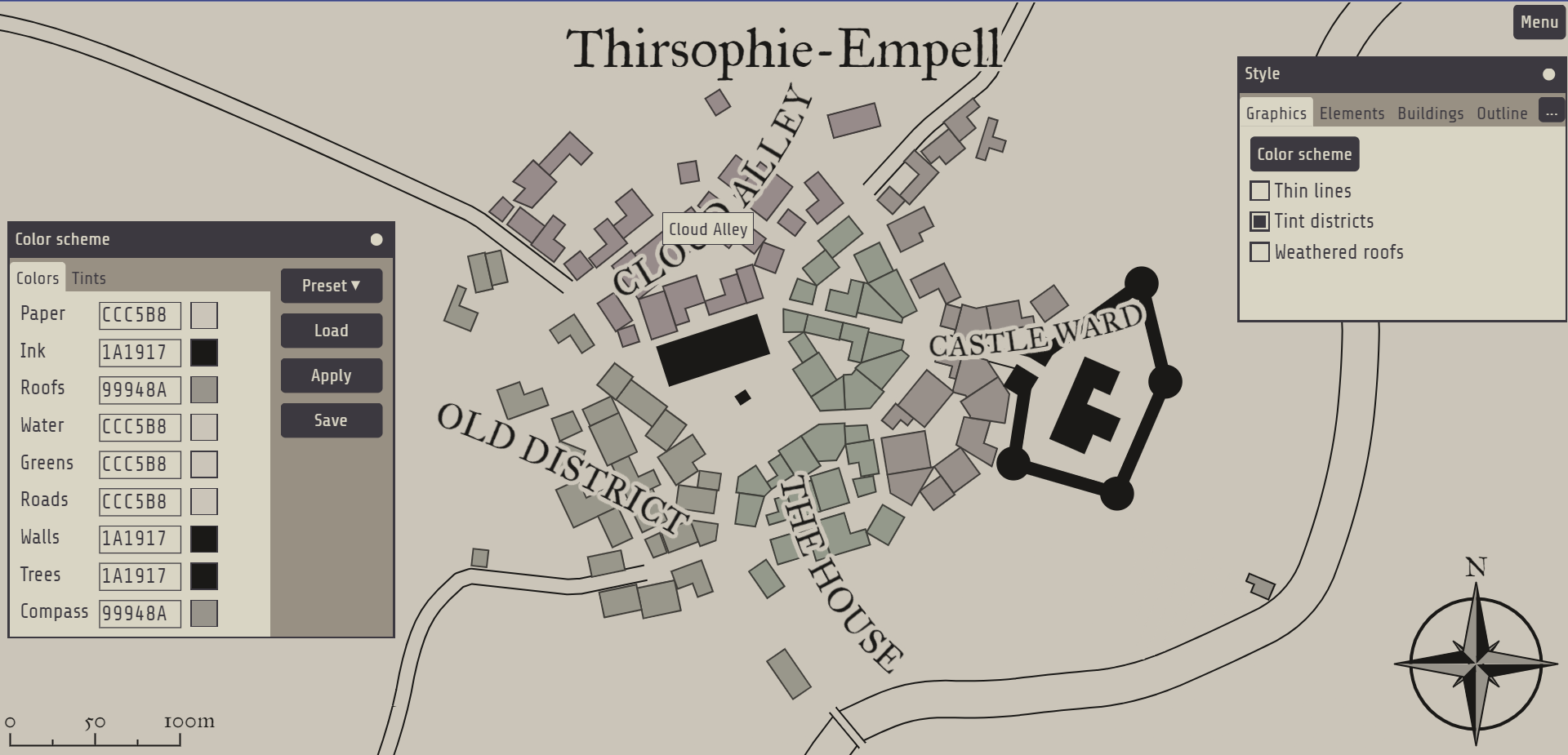


Figure 2 Procgen Arcana interface

1.2 Algorithms

Algorithmsare a core part of this project. And as such there will be extensive algorithm analysis to pick the most suited ones.

This section is divided in four parts to discuss algorithms for uniform point distribution over an area, noise generation and texture blending separately. Each section will comprehensively discuss a selection of algorithms, providing an analysis of the advantages and disadvantages associated with each algorithm.

1.2.1 Uniform point distribution algorithms over an area

Uniform point distribution algorithms are a vital part of the asset distribution during the map creation. Determining the placement of each building within the user-defined ‘town’ area in a manner that avoids overlap and maintains logical coherence presents a significant challenge.

In this section the discussion will be about Poisson-disk sampling and why it is better than basic noise distribution.

Poisson-disk sampling (PDS) is a method that ensures each sample is independent with a certain random distribution, while maintaining a minimum distance from each other [4]. The concept of distance in this context is quite abstract, as the definition of samples vary drastically from application to application. PDS is a method, a strategy, and thus requires an efficient algorithm to execute this method of sampling.

Maximal Poisson-disk sampling (MPDS): a set is considered maximal when no additional samples can be added to the sampling domain without breaching the minimum distance requirement, indicating that the domain is fully occupied [7].

First off, the method itself can be broken down into 5 steps, this unoptimized version is also called “dart throwing” technique [7]:

* + - * Define domain. Setting the boundaries of an area within which the samples should be generated is the first step towards generating the uniform distribution.
      * Choose value that will represent minimum distance between samples.
      * Randomly generate a sample on the domain, and check if it satisfies the minimum distance property. The sample is added to active list, a list containing all valid samples, in case it is located farther than distance r to other and discarded otherwise.
      * To satisfy the maximal condition, step 3 will be executed until there is no place to put an additional sample.

Such a method coded naively will perform badly, and in case if maximal condition must be satisfied – extremely bad.

There are two major algorithms for calculating MPDS and each introduce separate optimizations.

1.2.2 Bounding box calculation methods

Bounding box (BB) is an essential data about a certain area on the screen space or virtual space. It is defined as approximation of an area using arbitrary shapes like box or circle which allows to represent this area with minimal amount of data. In this project it will be used to approximate pixel areas like forest, urban area or castle defined by the user. Since storing all of the pixel positions as area attribute is extremely inefficient, after the user painted the area, BB will be calculated for this area, which will allow to store only four positions of said bounding box.

BB is usually calculated based on minimum and maximum x and y coordinate values present inside the area. Seems very manageable at first, however, it is only the case when only one area is assumed. This project requires to calculate bounding boxes for each separate area that needs to be populated with assets, like urban area, forest area and castle area. In cases of plain asset distribution calculating bounding box that encompasses all regions of certain type should suffice as based on the bounding box area the MPDS will be calculated and points that are outside of painted areas will just be dropped. As for area specific bounding box, the most complex part will be distinguishing separate painted regions that have the same color. Each type of area will be painted on blend mask with its distinct color, but since there can be several regions of same type, the algorithm for determining should be in place. Connected component analysis is the most widely used algorithm for distinguishing pixel areas of the same color. After distinguishing the areas bounding box can be calculated for each pixel area.

1.2.3 Procedural noise generation

Noise is a term used to describe the random or stochastic variations that occur within an expected signal [12]. Procedurally generated noise will serve as the foundational element for the entire map-making project. Every visual aspect presented to the user will be inherently influenced by this foundational noise. Blend mask, which is a raw mess of colors painted by the user, will be distorted heavily by the generated noise image, which will give much more natural output of the landscape.

There are several noise calculation methods:

* + - * Perlin noise
      * Simplex noise
      * Value noise
      * Voronoi noise

While value and Voronoi noises have interesting patterns, they have hard white-black characteristic, which is not desirable for terrain generation. Perlin and Simplex on the other hand are both gradient noises, and hence very frequently used for procedurally generating landscapes.

Perlin noise was developed in 1983 by Ken Perling and then published in CIGGRAPH paper in 1985 [13]. After Perlin noise was introduced, it had a significant impact on the field of computer graphics as it allowed more realistic imagery. It is widely used still, even after Simplex was introduced by the same author in 2001. Despite its name, simplex noise is more complex in understanding, but it was developed to remedy some shortcomings of Perlin noise, like large computation time in higher dimensions and directional artifacts [14]. Since the gain in computation time is negligible in 2 dimensions, Perlin noise is the choice for this project. The computation of this noise generation method can be broken down into such steps:

* + - * The algorithm starts with a grid of vectors. These vectors can be 1D, 2D, 3D, or even higher dimensions.
      * For each point in the space that needs to have its noise value computed, the dot product of the vector from each grid vertex to the point and the corresponding grid vertex’s vector is taken.
      * Interpolate between these dot product values, based on the relative location of the point within the grid cell.

This algorithm will produce black and white image resembling smoothed out cloud-like patterns which with increased complexity will allow to use its pixel color as a basis for the terrain.

1.2.4 Wave function collapse

The wave function collapse method is used widely in procedural generation because of its simplicity and high quality of output results. The method is designed to generate an output based on provided rules, for example it can generate a poem by using provided words and rules like which word rhymes with which. Similarly, it is used to generate any type of imagery, using unique tiles and rules for each tile border telling how selected tile can be attached to other tiles.

This type of generation will be valuable for this project since one feature will heavily rely on it. This feature being castle generation, although due to user’s ability to import custom assets it can be anything else.

Wave function collapse method for this project would have such steps:

* + - * Divide bounding box of pixel area into squares. These will become our tiles. In case of plain bounding box after generating all the squares, some of them might not contain area pixels, in such cases “empty” squares will be discarded.
      * Choose square with least entropy and assign to it a tile randomly chosen from allowed tiles. The lower the number of allowed tiles for a square the lower its entropy. In the case of castles, border squares will have least entropy at the start as not all tiles will fit for the border.
      * Reflect the consequences of the choice on all the adjacent squares that have not yet been decided.

1.2.5 Frustum culling

Frustum from geometry is a truncated pyramid, the apex of it is cut off. In rendering, the space that is visible on the screen is called frustum.

A diagram of a prism

Description automatically generated

Figure 3 perspective projection frustum visualization

Frustum is defined by the six planes that encapsulate the region. They are called far, near, right, left, top and bottom planes. For perspective projection these planes are defined by a point in space and a normal vector. This information will be sufficient for determining single plane, and since we will have them for each of the six planes, the frustum region can be determined at any moment.

For the purposes of this application the frustum will have a little bit different look, since the application is 2D only, instead of perspective projection, orthogonal projection will be used. In the case of orthogonal projection, the frustum region will be a cuboid instead of prism and hence, defining the planes becomes much easier. The planes will be solely determined by points in 3D space. The application is 2D, however, the third axis will still be used for ordering the objects drawn.

The frustum culling is a process of discarding the elements to draw if they appear to be outside of the frustum volume. Assets are the main elements to be discarded since they are all separate objects with each having scree-space position. If the canvas has thousands of assets placed on it, and the user is zooming in, without culling each of the assets will still be drawn, even the ones outside the window space. Because of the zooming each asset will occupy a lot of pixels and each pixel will have to be calculated, even though most of them are outside of the screen. To avoid this, before executing draw calls for the assets, all of them will be validated by checking if they are inside the frustum.

1.3 Analysis of multiplatform desktop development technologies

1.3.1 Programming Language

The 2D map generation tool, as any other graphically intensive desktop application, consists of 2 major parts: graphics programming interface and programming language used to deploy the commands to graphics card. In this section different possible programming languages will be analyzed to ultimately decide which language to use. This project is focused on real time map creation and hence the key deciding factor will be high computation speed, among others. The programming languages considered for this project are Java, C++, C# and Python.

Java is a programming language primarily used for android application development and creating proprietary automation solutions for large corporations, like banks or telecommunication services providers. It allows to have easier time of developing as opposed to something like C++ as it has some things like garbage collection automated, which can be tedious when using C++. This gives some advantage to Java, but at the same time for real-time application like this project, Java will have noticeable performance hit due to it being more high-level.

C++ is a widely utilized programming language, particularly in the development of applications where performance plays crucial role and in scenarios that require a more direct interaction with CPU, GPU and memory. For applications such as games and streaming services, where even the slightest performance improvement is crucial, C++ is the predominant language used for development. While programs written in C++ are often more complex to develop compared to those written in other languages, the benefits of significantly improved performance and granular control over memory and other PC components have allowed it to remain highly competitive.

Next in line is C# programming language and its use cases. C# is mainly used in 3 ways:

* + - * .NET Core enterprise development using C#
      * Microsoft WPF – a Windows desktop application development
      * Unity – the second most popular game engine, uses C# as its scripting language.

Considering proximity of this project to game development, with its focuses on performance and real-time calculations and rendering, Unity might have been a valid tool to develop the map generator. However, Unity will be an absolute overkill for a project of such scale, prolonging the development time, increasing the required disk space for installing the application and increasing battery consumption of the user’s device running the application.

Python is an interpreted programing language, meaning it requires an interpreter program to run the script. The interpreter program steps line-by-line over written script during runtime and interprets every line into machine code. This might not make python the fastest programming language, but it allows for rapid development cycle and fast prototyping. In the industry python is used extensively for automating all sorts of tasks, machine learning and web application development, but very rarely for desktop application development.

The final decision was made in favor of a language that enables the creation of high-performance applications and allows direct management of all computer hardware. The language chosen was C++.

1.3.2 Graphics Application Programming Interface

A Graphics Application Programming Interface (API) is a set of instructions that specify, what function should have what output and how it should perform [16]. These instructions are created by owners of the API, and it is up to the graphics card manufacturers to implement these functions. Choosing graphics API that suits best for this project is very important as major part of 2D map generator features will be developed using the certain graphics API. There are several graphics API’s available for building such an application, the most popular of which are DirectX, OpenGL, Metal and Vulkan.

Microsoft DirectX was first introduced in 1995 and with its latest version twelve comprises a set of application programming interfaces (APIs) designed to manage various multimedia-related tasks, with a primary focus on game programming and video applications, specifically tailored for Microsoft platforms. While it is usually the best choice for developing Win32 applications, the platform on which map generator will be developed, it does not satisfy the key requirement of multiplatform support.

OpenGL (Open Graphics Library) is another graphics API, it is developed by Kronos Group and released in 1992. From that point on, OpenGL became extremely popular among graphics developers and received several version updates. Main strengths of OpenGL nowadays include large community of developers, ease of use, built-in shader compiler and more. OpenGL is suitable for this project as it allows for multiplatform development with considerably easier setup and development process compared to others.

Metal, an API developed by Apple, is a low-level, low-overhead solution for hardware-accelerated 3D graphics and compute shaders. It made its debut with the release of iOS 8 in 2014 and have since then been used in iPhone, iPad and their computers. With recent addition of ray tracing support, Metal is becoming very powerful, allowing developers to fully utilize Apple’s diverse silicone portfolio. As advanced as Metal API is, it is not suitable for this project for the same reason as DirectX – it is a single platform solution.

Vulkan is an API also developed by Kronos Group and introduced in 2016 as a substitution for OpenGL. Vulkan was created to provide better support for new generations of graphics cards and give even more low-level interface for developers. This approach of Kronos Group ensured that Vulkan was never meant to be easy in its use, but with more complicated approach it can be faster in some cases than OpenGL. A lot of processes that were automated in OpenGL were instead left for developer to implement them. As its predecessor, Vulkan as well multiplatform, and hence is a valid choice for this project.

The choice between Vulkan and OpenGL was ultimately influenced by several factors. These included the more accessible learning curve of OpenGL, bolstered by a substantially larger pool of learning resources. Additionally, the deployment process for OpenGL was significantly simpler. While Vulkan may offer faster performance, the difference was not so revolutionary as to outweigh these considerations.

1.3.3 small conclusions

2. System Requirements Specifications

The proposed solution in this paper will consist of few parts that will together compound convenient and powerful tool for map creation. These parts are as follows: Easy-to-use user interface, assets customization, procedural terrain generation based on user-defined areas, painting toolset and fast real-time generation.

2.1 Use case diagram

Main system functions and actors are presented in Fig.2.1. and Table.2.1. Use cases were created based on main functionality and include drawing areas for map generation, asset management and saving procedure.

A diagram of a person

Description automatically generated

Figure 4 Use Case Diagram

Table.2.1 Use Cases

|  |  |
| --- | --- |
| No. | Description |
| UC-1 | User is presented with start screen where they can adjust canvas size and choose whether to import previous saved state. |
| UC-2 | User can draw on the generated result with selected brush type (terrain, water, town etc.) |
| UC-3 | User can save painted result into file. |
| UC-4 | User can save current state of the canvas into a binary file, to restore it later. |
| UC-5 | User can adjust noise scale and complexity during the painting of the map. |
| UC-6 | User can upload .png images of custom assets to use for the painting. |
| UC-7 | User can change resolution of the canvas at the application startup. |
| UC-8 | User can open previously saved state of the canvas from binary file. |
| UC-9 | User should input file name to save the image. |
| UC-10 | User can review the assets during the importing. |
| UC-11 | User can adjust assets settings like density, size and variation. |
| UC-12 | User can adjust terrain and water settings like color, gradient, texture. |

2.2 Sequence diagrams

All system tasks are further described in sequence diagrams. Map creation process using this project’s solution is presented in Fig. 2.2.

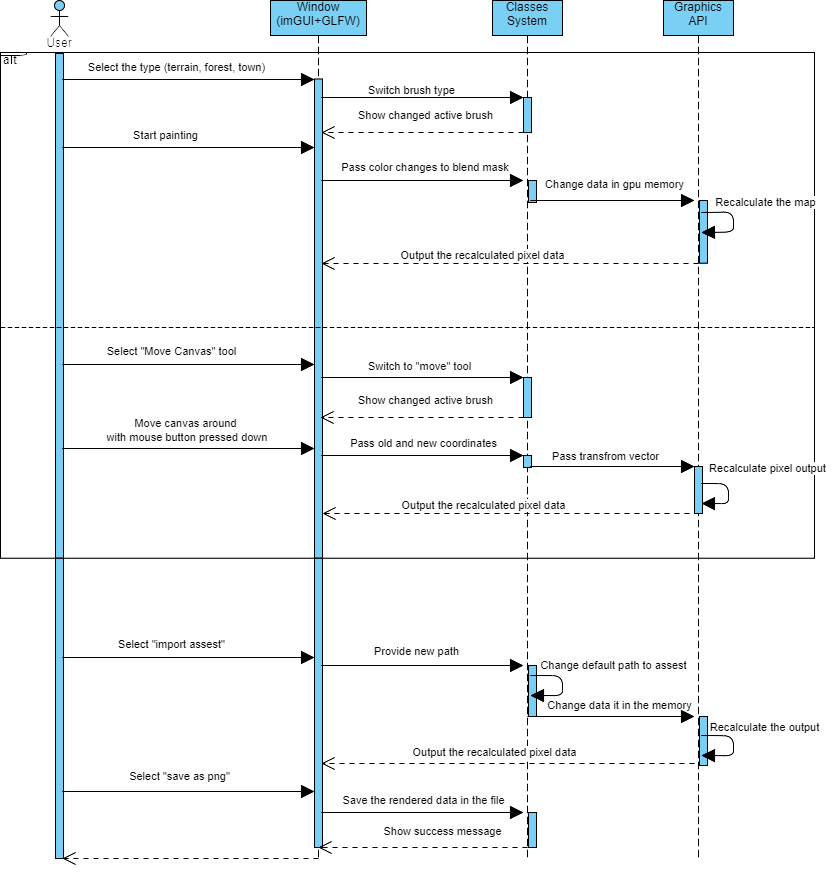


Figure 5 Sequence diagram for all use cases

The user selects the brush tool and selects the type of brush from the pre-defined list of types. The types include terrain and water, allowing to paint them with color or user-defined texture, forest and town, allowing equidistant distribution of default or custom assets. After the left mouse button is pressed down, the changes to the blend texture (a texture with information about defined areas) will be updated every frame on the CPU side and passed to GPU without recalculation of the map. Recalculation will happen after the user releases the mouse button. In the meantime, the user will see unedited brush stroke. Following the release of the left mouse button the application will reflect the updates to blend image to the entire map. After performing as many map adjustments as user wishes to, they can select option “Save map” from graphical user interface. This option allows to save the map locally on the PC and after selection of save directory application will perform rendering of the final output into a file. After saving the user will be presented with success message and will have the option to continue working on the map.

2.3 Functional Requirements

Severity: 1 – critical, system will not be able to perform and will crash. 2 – component crashes, but not the app, 3 – minor, application will still perform, just minor issues.

Priority: H – mandatory requirement, M – valuable, but not mandatory, O – optional.

Dependency: H – closely related to other functions, M – closely related to other functions, optional requirement, L – does not depend on other functions, does not require additional adjustments to other components.

Table. 2.2. Functional requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Description** | **Priority** | **Severity** | **Use cases** |
| FR-1 | When system starts, it presents user with the empty canvas and GUI. | H | 1 | None |
| FR-2 | The system allows to choose brush type. | H | 1 | UC-2 |
| FR-3 | The system allows to save current map into file. | H | 2 | UC-3 |
| FR-4 | The system registers brush strokes done with mouse. | H | 1 | UC-1 |
| FR-4 | When user start brush stroke, the system demonstrates the stroke with plain color. | M | 2 | UC-1 |
| FR-5 | The system allows to switch between painting and moving canvas. | M | 2 | UC-1 |
| FR-6 | The system allows to import custom assets. | H | 2 | UC-6 |
| FR-7 | The system allows color adjustments and texture selection for each area. | M | 3 | UC-12 |
| FR-8 | If custom assets import failed, the system informs user and reset to default assets. | H | 1 | UC-6 |
| FR-9 | If user tries to move canvas out of the window frame, the system prevents it. | O | 3 | UC-1 |
| FR-10 | The system handles window resizing. | H | 2 | None |
| FR-11 | The system allows to set canvas size. | H | 1 | UC-7 |
| FR-12 | The system allows to save current progress | M | 2 | UC-4 |
| FR-12 | The system allows set density, size and variation to paintable assets | M | 2 | UC-11 |
| FR-13 | The system allows restoration of previous progress | M | 3 | UC-8 |

2.4 Non-functional requirements

Table. 2.3. Technical system requirements

|  |  |
| --- | --- |
| No | Description |
| NFR-1 | System is developed using GLFW and GLEW libraries for C++ to support OpenGL. |
| NFR-2 | System use STB library for C++ to handle image data. |
| NFR-3 | System use imGUI library for C++ to create graphical user interface. |
| NFR-4 | System use GLM library to perform matrix calculations. |
| NFR-5 | System use OpenGL specification for execution of GPU commands. |
| NFR-6 | System has multiplatform support. |

Table. 2.4. Reliability and Maintainability requirements

|  |  |
| --- | --- |
| No | Description |
| NFR-7 | System saves the map currently opened in case of crash. |
| NFR-8 | System has modular structure for easy maintainability and better reliability. |
| NFR-9 | When one module fails system should not allow for cascading failure effect. |

Table. 2.5. Usability requirements

|  |  |
| --- | --- |
| No | Description |
| NFR-10 | System has clean, convenient and understandable UI |
| NFR-11 | System employs English as its UI language. |
| NFR-12 | In case application encounters error during the runtime, the user is informed in an intuitive way |

Table. 2.6. Portability and Scalability requirements

|  |  |
| --- | --- |
| No | Description |
| NFR-13 | System is built for Windows and Linux. |
| NFR-14 | System is shipped with all necessary libraries included. |
| NFR-15 | System performance scales according increased complexity of the map |

3. Architecture specification

The application preliminary user interface design is featured in figure below. The image features a powerful yet simple design that employs extremely legible font and understandable labeling.



Figure 6 User interface mockup

The structure of this project in terms of technologies used and how they interconnect will be discussed next. The system setup was performed by installing the Microsoft Visual Studio application with C++ module. Visual Studio Community is an integrated development environment (IDE) of choice, and it will be used for C++ project building. The additional libraries GLFW, GLEW, STB and GLM were consequently added to the project. All the extension libraries were placed in single directory named “Dependencies” in the project solution directory to use relative paths when linking libraries. Employing relative paths is crucial, as it will allow to launch application with a degree of flexibility, ensuring they can be initiated without being constrained by computer’s directory’s structure.

GLFW and GLEW setup was achieved by downloading library files from the official web site in both cases. Figure 6 features the main page of the GLFW website.

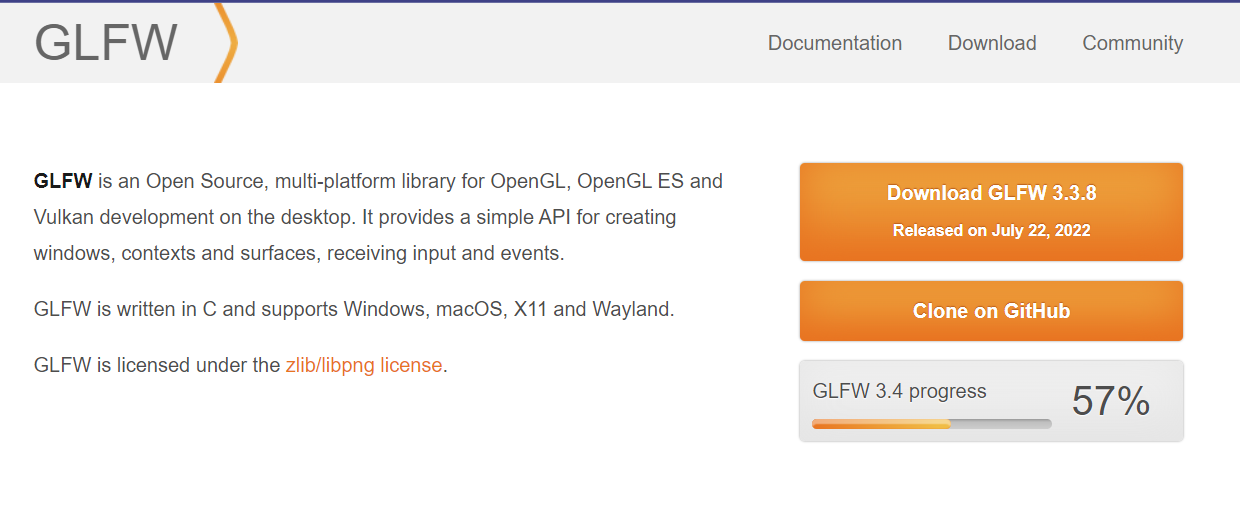


Figure 7 GLFW official web site.

Similarly, in case of GLEW, the source files were downloaded from their web site and added to Visual Studio project solution.

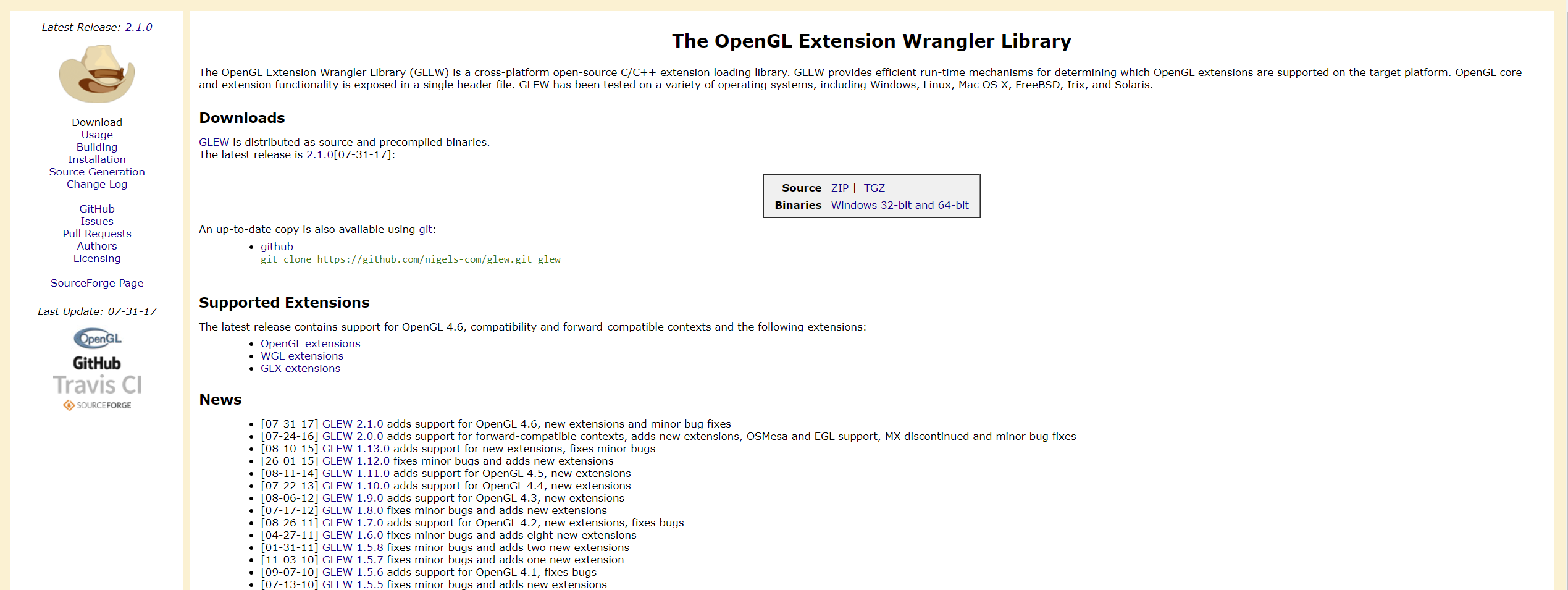


Figure 8 Official web site for GLEW

Adding the files to the project’s build came down to dynamically linking the dependencies folder and its contents in the project’s properties inside Visual Studio.

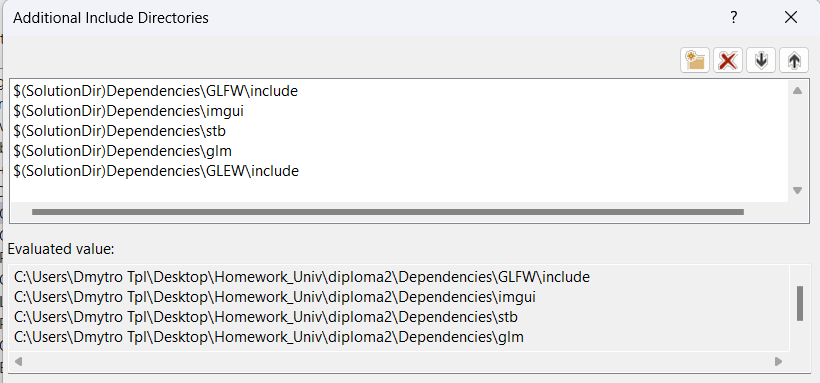


Figure 9 included directories for project

In case of STB and GLM libraries, the up-to-date versions of them are hosted on GitHub and requires pulling the repositories. After the pull, files were placed in corresponding folders in the dependencies directory and linked in the project’s settings as well.

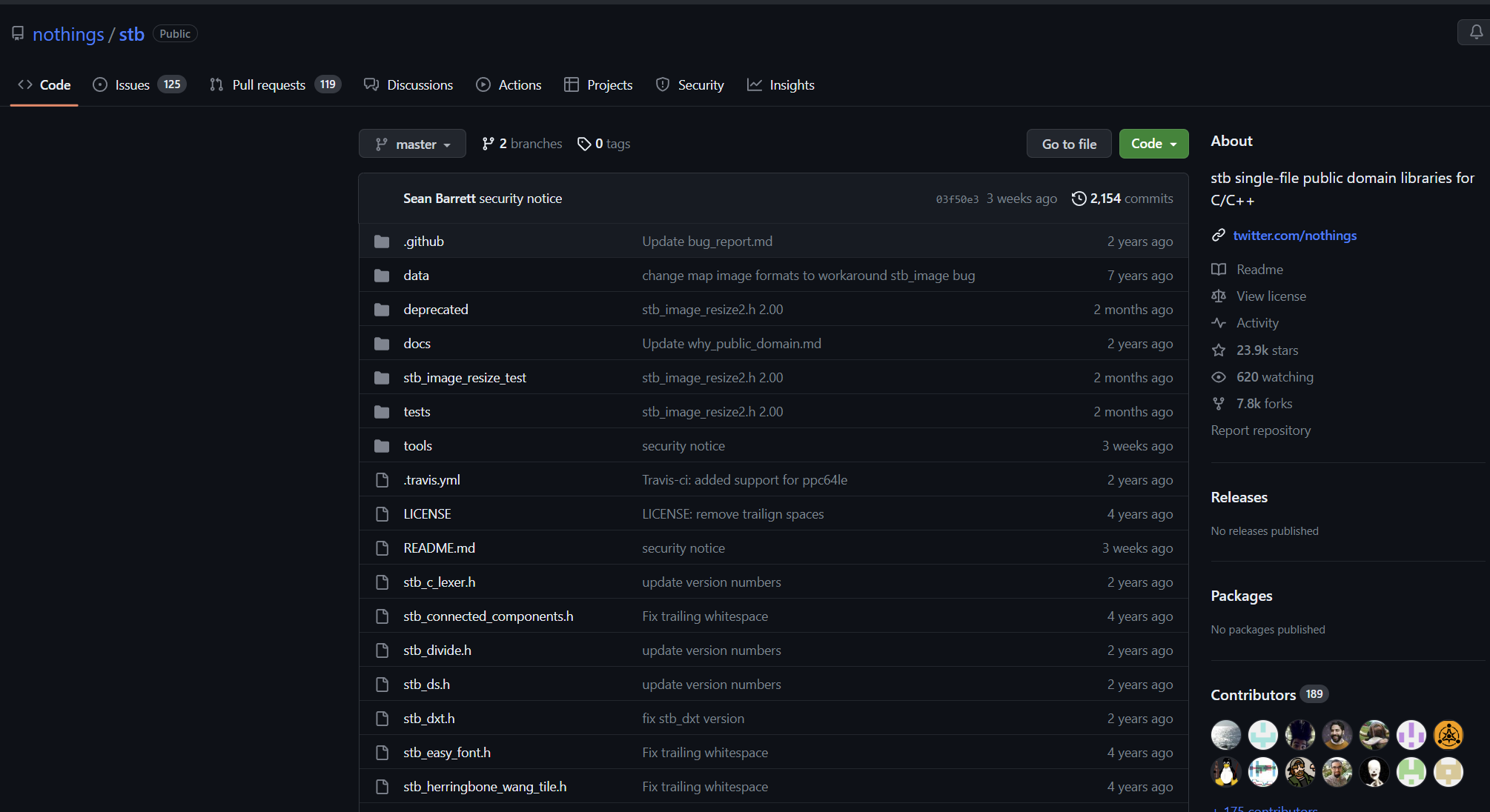


Figure 10 STB GitHub repository

3.1 Component diagram

3.2 Class diagram

A diagram of a computer

Description automatically generated

Figure 11 Class diagram

Figure 10 depicts a class diagram for the developed application. The breakdown of what each class is responsible for is described in the flowing table. Overall, the main class is State Handler, which is responsible for managing application during runtime. Switching brushes, tracking mouse clicks and drags, saving the map – are only some functions that this class is executing.

Table 3.2.1 Class descriptions

|  |  |  |
| --- | --- | --- |
| № | Class | Description |
| 1 | State Hander | Manager class that will contain most of information about current application state like what brush is chosen, is mouse pressed, canvas transform etc. |
| 2 | Quad | Basic class for simplifying management of OpenGL resources to draw a quad. Quad have information about each vertex, transformation matrix, texture etc. |
| 3 | Canvas | Subclass of Quad, a canvas entity. This class holds all canvas related data like pixel RGBA information, noise scales, gradient usage and colors used for terrain and water. |
| 4 | UI Handler | Helper class to deal with imGUI conveniently. |
| 5 | Painter | Class responsible for the action of painting the canvas. Deals with raw pixel data. |
| 6 | Asset Handler | Asset class to create and draw an asset type. Currently only three asset types are created at the start, but this solution will also allow to make an infinite number of unique asset types. |
| 7 | Vertex Buffer, Index Buffer and Frame Buffer | Classes to easily manage different buffers’ creation, initialization and destruction. |
| 8 | Frustum | Frustum struct is added to simplify the frustum management during runtime. The data it stores is helping with occlusion culling of the out of screen assets. |

3.3 System description

3.3.1 Startup

The developed system is a complicated application with a plethora of separate systems and features. The system starts with a generation window for the host operating system. The library GLFW takes care of creating windows, contexts and surfaces, receiving input and events. Next the system is compiling shaders. There are 5 shader programs total – heightmap generation shader, shader for startup screen, asset shader, cursor shader and final terrain shader.

When shaders are compiled, assuming no errors, the system initializes all the entities needed to present the initial screen like background quad, state handler entity, canvas entity and ui handler. After this, the main application loop starts. Initially the startup quad is drawn and UI done with ImGUI library is drawn afterwards. When the user presses “Start” button the system checks the state of the “load saved state” toggle and in the case when user wants to load previous state the system is deserializing the data from the binary file stored in the application’s directory. The deserialization process is basically retrieving crucial state handler and canvas attributes from the file while considering the order of the data packed there.

When all is ready to present the user with main application’s interface the system initializes frame buffers along with initial asset positions (not yet shown, only generated).

A diagram of a software process

Description automatically generated

Figure 12 Flowchart of system startup

3.3.2 Painting terrain

From now the user can start the painting process. This process comes down to adjusting raw RGBA data of the canvas or the assets texture. The application does not store raw RGBA data of the final map, instead the proxy map is stored, since it will be the one directly influenced by the user. Before the final version of the canvas can be drawn on the screen, the application needs to generate a height map of the terrain. This needs to be done to have real-like appearance of the final terrain look. A height map is being generated by accepting proxy map where the red channel contains mask for the terrain, and inversion of that mask is considered water. In the future, possible improvements will include assigning two other types of terrain to the green and blue channels. By applying different Perlin noises and other effects to the terrain mask the final height map is generated and it is passed down to terrain shader. This is done by drawing canvas with raw heightmap applied to it as its texture to the frame buffer.

A diagram of a data flow

Description automatically generated

Figure 13 Flowchart of heightmap generation

The terrain shader is responsible for managing all the settings presented to the user. The colors, outline, optional gradients all are managed inside this shader.

A diagram of a flowchart

Description automatically generated

Figure 14 Flowchart of final terrain rendering

3.3.3 Painting assets

The assets are painted similarly to the terrain. There is an additional RGBA texture the size of the canvas stored in the canvas entity. The different channels of this texture responsible for different assets. They are basically act as visibility masks for the assets. This means that all assets are existing even if user don’t see them, they are just not drawn. The density value is set by the user in the settings and after regeneration of evenly distributed points the system is discarding the points that are masked by corresponding channel of the asset texture. Assets represented in the system by their position and index, and the latter is needed to assign them specific texture in the shader. The instance ID is used to determine texture of the asset instance, since they have to be at least pseudo random.

The early version of this pipeline had a major flaw as before assigning each asset instance their unique id manually the gl\_instanceID (provided by OpenGL itself) was used in the shader. During the painting process the amount of assets drawn on the screen is constantly changing, with changes to their order as well, resulting in gl\_instanceID having different values for the same instances. This resulted in the inconsistency in the assigned textures when assets are being painted and visually unpleasant experience for the user.

3.3.4 Saving

3.4 UI description

At the very start of the application, the user is presented with a start screen where they can input the canvas size and make a choice whether to import previously saved state. It’s important to note that user must input the correct canvas size that was used in the saved state, otherwise it will lead to incorrect interpretation of the save state. Here below the start screen is presented.

A screenshot of a computer

Description automatically generated

Figure 15 start screen of the developed application

Whilst inside, the developed application has some differences in user interface than it was designed in the pre-production stage (beginning of the chapter 3). Here, in the image below the final user interface is presented.

A screenshot of a video game

Description automatically generated

Figure 16 screenshot of the final user interface

Its worth noting that some features declared at the pre-production stage were left out of scope. For example, asset replacement and review during the runtime is not possible inside the application. Asset replacement is still possible, however to perform this use case the user will have to manually replace the assets however, file in the application resources directory. The assets are not stored in separate files and due to the performance optimizations assets are stored in one file, compiled in an atlas of textures.

A grid of black objects

Description automatically generated with medium confidence

Figure 17 asset atlas opened in Photoshop

On the picture above is depicted how the assets are laid out in the atlas. The atlas can contain 8 unique types of assets with each having 8 tiles for variation. This system is easily scalable and future-proof since adjusting its dimensions can be easily added.

The feature of reviewing the asset ended up not implemented due to time limitation and small impact on the product itself.

The UI includes 2 main panels, main on the right and an additional one on the left. The right panel features all the available modes and tools. There are two main modes, moving mode and painting mode. Moving tool provides more free space by not having seconds panel, this allows user to inspect the map more conveniently. The second mode for painting includes all brushes available. The brushes available are as follows:

1. Terrain up brush.

2. Terrain down brush.

3. Buildings brush.

4. Flora brush.

5.Mountains brush.

Each of the brushes has an additional panel responsible for showing settings for that brush and settings for the entity this brush is painting. For example, “Terrain up” shows brush size, opacity and hardness, but also all the terrain settings available like main color, secondary color, toggles for gradient and gradient type, and outline settings. “Terrain down” on the other hand features water settings along with brush settings. Asset brushes are similar as well, but their additional panel is exposing asset settings like asset size and density.

The main panel has also settings for canvas, which are presented below all of the brush types. At any point in making the map the user can adjust the noise complexity and sizes for two noise textures used to make the terrain. After that there are saving options, saving to a png or saving current state to a binary, to possibly return to making the map later.

3.3 Description of practical part

Development of the application during this project will be carried out in multiple steps:

3.3.1 Setting up of the environment.

The IDE of choice is Microsoft Visual Studio, it will be used to write the source code and build the application for windows. Inside this program, all of the dependencies will be linked so that additional libraries will be built along the original source code.

3.3.2 Writing “Hello World” of computer graphics.

In computer graphics, an application “Hello World” is a program that is creating a simple window displaying simple colored triangle in the middle. This is mildly difficult procedure as it requires setting up all of the triangle data in GPU-friendly way as well as setting up the window.

3.3.3 Abstracting OpenGL commands into classes.

This step is necessary to expedite the further development as the source code for simple triangle is already quite clunky, and considering that this project requires hundreds of them, the code will very quickly become unmanageable without this step.

3.3.4 Implementing UI.

UI for this application was developed using imGUI library for C++ programming language. It is both simple and powerful, so it was an easy choice.

3.3.5 Developing all the predetermined features.

The features like painting, asset distribution, terrain generation are written after OpenGL is abstracted into convenient classes and are built upon that foundation. Every interaction of the user with UI or canvas will be followed by changing or adjusting class members. For example, class asset will contain positions, size, texture etc.

3.3.6 Testing.

During the testing, the developed application will be tested for bugs and usability. Every developed feature must work as expected and will be fixed or adjusted in case some unexpected issues appear. The testing stage will also help with understanding the usability of the developed solution.

4. Testing

4.1 Manual Testing

Table 4.1. TST-1 test case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | TSC-1 | | | |
| Test scenario | TSS-1 | | | |
| Step No | Action | Input data | Expected result | Status |
| 1. | User click folder icon |  | Open explorer for file selection | Success/Fail |
| 2. | Select and saves | images | Images saved in server |  |

|  |  |  |
| --- | --- | --- |
| Test No. | Functional No | Scanario |
| TSC-1 | FR-1 | Check if assets can be selected and save in app. |
|  |  |  |

Conclusions

1. Analysis of existing map generation solutions revealed that existing tools have an extremely different approach to creating maps. Inkarnate gives the creator the asset placement tools and ready-to-use brushes to increase the speed of work. Procgen arcana on the other hand, takes liberty to generate the map for the user and allow for some adjustments. However, one of them skew heavily in proceduralism and allows for minimal customization due to technology it’s built on and other lacks automation and the quality of the output largely dependent on user. Therefore, the solution provided in this project takes the best from both approaches and allows for a great time saving as well as for fine adjustments.
2. Algorithms analysis allowed to determine basic algorithm stack that will be used in the project like uniform point distribution, bounding-box calculations and wave function collapse generation. Furthermore, the desktop development technologies analysis helped to determine best programming language and graphics API. Following an assessment of most popular technologies, the decision was made to in favor of C++ programming language based on the speed of the code execution and OpenGL graphics API based on ease of use, huge community around it and its speed being on par with much more complicated counterparts.
3. The modeling stage allowed to determine main use cases and sequence flow, which has helped to greatly reduce the development time by setting explicit targets. The use cases include picking the brush type from the pre-defined set, painting on the canvas with it, saving the result into a file and changing map style by adjusting colors and asset textures. The sequence diagram helped to determine how the data flows in the system and what steps have to be executed to achieve desired functionality. Furthermore, the functionality and appearance were described even more precisely with description of functional and non-functional requirements. That allowed to solidify the scope of the project and clarify the prioritization of features development process.
4. The development stage was centered around developing the application structure, figuring out the internal operation logic and implementing all scoped features. During this stage some functionality was proved to be easier than others in its development, which resulted in additional time spent on implementation of such functions. Since the application revolves around the ability to paint with different brushes, additional attention was dedicated to implementing this function. Asset review feature ended up not implemented due to time constraints.
5. The testing stage of the development of this project allowed to evaluate the developed application and iron out any kind of issues that would potentially hurt the user experience. During this stage the application went through comprehensive testing of all its features and use cases, which allowed to uncover some minor problems that consequently were resolved by updating the source code.

References

1. T. Gao, J. Zhang and Q. Mi, Procedural Generation of Game Levels and Maps: A Review, 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Korea, Republic of, 2022, pp. 050-055, doi: 10.1109/ICAIIC54071.2022.9722624.
2. Röhl, Tobias & Herbrik, Regine. (2008). Mapping the Imaginary - Maps in Fantasy Role-Playing Games. First publ. in: Forum Qualitative Sozialforschung 9 (2008), 3, Art. 25. 9.
3. Yao, X., Zhang, Y., Bao, F. et al. The blending interpolation algorithm based on image features. Multimed Tools Appl 77, 1971–1995 (2018). <https://doi.org/10.1007/s11042-017-4379-5>
4. Wang, T. (2021). Poisson-Disk Sampling: Theory and Applications. In: Lee, N. (eds) Encyclopedia of Computer Graphics and Games. Springer, Cham. <https://doi.org/10.1007/978-3-319-08234-9_398-1>
5. Yan, DM., Guo, JW., Wang, B. et al. A Survey of Blue-Noise Sampling and Its Applications. J. Comput. Sci. Technol. 30, 439–452 (2015). <https://doi.org/10.1007/s11390-015-1535-0>
6. R. Bridson. Fast Poisson Disk Sampling in Arbitrary Dimensions, SIGGRAPH 2007
7. David Cline, Kenric B. White, Parris Egbert, "Poisson Disk Point Sets by Hierarchical Dart Throwing", Symposium on Interactive Ray Tracing, pp. 129-132, Sept. 27.
8. Quan, W., Yan, D-M., Guo, J., Meng, W., & Zhang, X. (n.d.). Maximal Poisson-disk Sampling via Sampling Radius Optimization. NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing, China.
9. ERGÜN A., Examination of the usability of digital maps according to the geography course curriculum. <https://doi.org/10.32003/igge.1301845>
10. Medieval fantasy city generator by Watabou. itch.io. (n.d.). <https://watabou.itch.io/medieval-fantasy-city-generator>
11. Zand, M., Etemad, A., & Greenspan, M. (2022). Oriented Bounding Boxes for Small and Freely Rotated Objects. IEEE Transactions on Geoscience and Remote Sensing, 60. <https://doi.org/10.1109/TGRS.2021.3076050>
12. Abraham, S. (2023). Noise Functions. University of Texas. <https://www.cs.utexas.edu/~theshark/courses/cs354/lectures/cs354-21.pdf>
13. Perlin, K. (1985). IMAGE SYNTHESIZER. Computer Graphics (ACM), 19(3), 287–296. <https://doi.org/10.1145/325165.325247>
14. Gustavson, S. (2005). Simplex noise demystified. Linköping University, Schweden, 17. Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Simplex+noise+demystified#0>
15. Naim, R., Nizam, M. F., Hanamasagar, S., Noureddine, J., & Miladinova, M. (2010). Comparative Studies of 10 Programming Languages within 10 Diverse Criteria. Citeulikeorg (p. 126). Retrieved from <http://arxiv.org/abs/1009.0305>
16. Vries, J. de. (2020). Learn OpenGL. Learn OpenGL - Graphics Programming (Vol. 42, pp. 194–196).
17. Bailey, D. G., & Klaiber, M. J. (2019). Zig-zag based single-pass connected components analysis. Journal of Imaging, 5(4). <https://doi.org/10.3390/jimaging5040045>

Annexes

* + - 1. **Title of the Annex**

Pagrindinis tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas.

* + - 1. **Priedo pavadinimas**

Pagrindinis tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas tekstas.