FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Sheepless - An Open-source 2D Adventure Game in Unity
Student:	Robert Badronov
Supervisor:	Ing. Marek Skotnica
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2020/21

Instructions

Sheepless is an open-source art game about a Shepherdess from Prague. EbSynth is a state of the art image synthesis technology developed at DCGI FEL CTU. This technology is intended to make a hand drawing animation easier. A goal of this thesis is to explore how to take advantage of this technology to design a prototype of a 2D game in Unity.

Steps to take:

- Review the EbSynth technology and Unity.
- Design game mechanics and game architecture.
- Create an open-source proof-of-concept implementation.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague December 12, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Sheepless - An Open-source 2D Adventure Game in Unity

Robert Badronov

Department of Software Engineering
Supervisor: Ing. Marek Skotnica

June 4, 2020

Acknowledgements

I would like to thank my supervisor, Ing. Marek Skotnica, for his invaluable help, patience, and also for the opportunity to combine the topic of creating games that is interesting for me with the writing of this work. I would like to thank my family and friends for their moral support. I would also like to thank my colleagues with whom the work on the Sheepless project is being carried out.

Declaration

In Prague on June 4, 2020 …………………

Citation of this thesis

Badronov, Robert. *Sheepless - An Open-source 2D Adventure Game in Unity.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Hlavním cílem této práce je zkoumání možností využití technologie EbSynth při vytváření art her pomocí Unity Engine. EbSynth se používá pro zpracování videí zaznamenaných ve hře, pro vytváření stylizovaných cutscén. Dalším cílem je analýza a navrh herních mechanik. Byly analyzovány a navrženy mechaniky, jako je pohyb, mapa, hraní cutscén, stejně jako hlavní mechanika kreslení na 2D spritech. Byl vytvořen prototyp obsahující všechny výše uvedené mechaniky. Implementace je kompatibilní s operačním systémem Android.

Klíčová slova Unity engine, EbSynth, Cutscéna, Mechanika kreslení, Vývoj her, Android, Prototyp

Abstract

The main aim of this work is to study the possibilities of using EbSynth technology while creating art games on the Unity Engine. The EbSynth is used for processing videos recorded in the game, for creating stylized cutscenes. Another aim is to analyze and design the game mechanics. Mechanics such as moving, map, playing cutscenes, as well as the main mechanics of drawing on 2D sprites were analyzed and designed. A proof-of-concept implementation containing all of the mentioned above was created. Implementation is compatible with the Android operating system.

Keywords Unity engine, EbSynth, Cutscene, Drawing mechanic, Game development, Android, Prototype

Contents

List of Figures

Introduction

Motivation and objectives

In an interview with Time magazine [1], when discussing the scope of work while creating hand-drawn animations, Cuphead's game lead artist Chad Moldenhauer said: *"over 25 minutes of work goes into one frame"*. Taking a cutscene, 10 seconds long, 30 frames per second, there are already 300 such frames, and this is only one cutscene. Using EbSynth technology, time and effort that the process of creating cutscenes requires can be reduced. One of the goals of this work is to consider this tool and its usage when creating a cutscene for a game built on the Unity game engine.

Talking about art games in a hand-drawn style, it would be good to add into the game some relatively new game mechanics that suit that style. Such mechanics is the real-time drawing on sprites. Attempts to find games with similar genre and mechanics were unsuccessful. For this reason, the goal to design and implement this mechanic was set. Another goal, related to the previous is to design and implement various drawing tools such as a color picker and an eraser. Another goal is to implement drawing brushes.

To demonstrate the results of what is written above, a proof-of-concept implementation of the game level will be made. A short video recorded in the game and processed using EbSynth will be added to the game. Also, to have basic functionality, essential game components such as the movement mechanism, game map, and control settings will be designed and implemented.

The mobile game market accounts for almost a third of the total number [2]. For this reason, the Android OS version of the implementation will be available.

The results of this work can be used in a game that has drawn style graphics or a theme related to drawing. One of these games can be a game called Sheepless.

Only part of the possibilities of using EbSynth and creating game mechanics for such a game was covered in this work. Other opportunities were covered by my colleagues Jan Klicpera [3] and Ian Mustiats [4], in their works.

Aims of the work

The main goal of this work is to study the possibilities of using EbSynth technology in creating art games. The first task is to study the Unity game engine and EbSynth technology. Further, it is necessary to find out in which specific areas of art games this technology can be applied. Next, the proof-of-concept implementation should be made to demonstrate the application of the technology. In addition, it is necessary to design and implement the mechanics of drawing on sprites. Basic mechanics, such as movement, map, and playing cutscenes, should also be implemented. Finally, it is necessary to design and implement a graphical interface.

Work structure

The first step to take is to consider the Unity game engine crucial components that would be necessary for further work. It will be done in Chapter 1.

The second step is to study how to synthesize the video using EbSynth technology. In Chapter 2 it will be explained what EbSynth consists of, how the graphical interface looks like and how to use it for video processing.

After that, in Chapter 3, the proof-of-concept game level requirements analysis, documentation, and design will be made.

The last step is to implement a proof-of-concept game level implementation. The process of creating will be described in Chapter 4.

Chapter 1

Unity Engine

Unity is a cross-platform game engine. It supports more than 20 platforms, such as PC, Android, iOS, and WebGL. In addition to its popularity in game development, the engine has been also used in areas which are not focused on games, such as movies, architecture, construction, and engineering.

Unity offers developers its main scripting API in C#. [5] For both game and Unity editor, it is possible to write own scripts and use them as plugins. Among other things, Unity offers the opportunity to import sprites, textures, and to use an advanced world renderer. Another powerful thing in Unity are the animation tools that make it simple to create cutscenes in both 2D and 3D game worlds. [6]

In the year 2018 almost half of all the new mobile games were developed on Unity. [7]

In this Chapter, the Unity game engine essential components that would be necessary for further work will be considered.

1.1 Unity editor

The Project window The files related to the project can be found in the Project window. The other thing that can be found in the Project window is the browser toolbar, which can be used for assets searching.

The Hierarchy window One of the important places is the scene objects list. In the Unity editor, this place is the Hierarchy window which contains all GameObjects of the scene that is currently opened.

If a game creates or removes some GameObject in runtime, it will appear or disappear in the Hierarchy window.

The Inspector window Each object is made up of other components. The place where the GameObject components can be found is the Inspector window. In addition to the components themselves, properties of the particular component can be found and changed here.

Similarly, the Inspector window contains necessary information about GameObject, such as name, tag, layer, visibility flag, and static flag.

The Scene view The scene view is the window where most of the work is done. Here, the game objects are located the way they should be in the game. Objects parameters, such as position or rotation, can be managed here.

The Game view A game view is a demonstration of how the game will look like when completed. The output image from the game cameras can be found here.

The Toolbar The toolbar is not a separate window. Various tools can be found here. Some of them are designed to work with the scene objects — others, to control the demonstration playback of the game. There are also tools to manage the Unity account and services.

1.2 GameObject

GameObject is the fundamental element in the Unity engine. Every object that can be seen inside any scene is a GameObject. Game objects are made up of various components. Let us consider some of them in more detail:

- **Transform**. Every GameObject contains a **Transform** component which is used to configure object position, rotation and scale. [8]

- **Sprite Sprite** is a 2D object that contains some texture inside it. By default, sprites are used in 2D game world, but it is possible to use them in 3D game world as well.

  In order to see sprite texture in the game, the **SpriteRenderer** should be attached to sprite GameObject. [8] Sprites in Unity contain two important things that are worth a close look:

  2D texture. A texture consists of a set of pixels. Each pixel represents a specific color recorded in ARGB format, where A is the alpha channel (pixel transparency), R, G, and B are the color components - red, green, and blue, respectively. With the **Texture2D** class help it is possible to get texture width and height. Methods such as GetPixel and SetPixel can be used for getting or setting the pixel color. [9]

Pivot - is the conditional center of the sprite. The coordinates of the Transform component are the coordinates of the pivot.

- **RigidBody**. Since Unity has a powerful physics 39 engine, using it can save the time. For an object to be affected by the physics engine it is necessary to add the **RigidBody** component to this object. As soon as this is done Unity will add this object into the physics engine and start applying forces and collisions on it.

- **Collider** is an object component which is an invisible shape that defines object borders for collision detection. When a collision occurs, callback methods are called.

  If it is necessary to avoid the collision, but we still need to know if a collider enters another one's space, it is possible to configure the collider as a Trigger. [10]

1.3 Camera

Camera is what the player sees during the game. The camera in Unity is a game object that contains the Camera component. Let us consider two, important for the present work, camera settings:

- **Projection** - camera projection mode. It includes two modes: Perspective (similar to real world) and Orthographic. [11]

- **Target texture** - here we can set up a texture into which the image from the camera will be drawn.

1.4 Asset

Asset is any item that can be used in the project. This may be the thing that is got from outside of Unity, for example some image, audio etc. Assets can also be some items that are got from Unity, for example animation clips, animation controllers etc. One of the special cases of assets is **Prefab**.

Prefab is an asset which can be created from some object with some components and properties, and reused as much times as wanted. If we want to make a change in all objects of this prefab we can just edit this prefab and set this changes for all objects of this prefab. [6]

1.5 Raycasting

Raycasting is the action of shooting with an invisible ray (line) from some point, to a determined direction to detect if there are any colliders laying in the path of the ray. When shooting with a ray, in case of a hit on a collider, the hit information will be recorded into the RaycastHit object. Among other things, this object contains a collision position in world coordinates (coordinates relative to the beginning of the game world). [10]

1.6 Unity animations

Unity provides very convenient tools for creating animations. So, in order to create and manage animations in Unity, it is necessary to learn two things:

1. **Animation clip** - is the basic element in Unity animation system. The animation clip itself is an animation. Animation can be made using two variants:

   - An animation that is composed of several pre-drawn sprites that are played one after another over time, forming an animation.

   - An animation that is made by changing some of the sprite parameters over time (for example object position can be changed to simulate moving). [8]

2. **Animator controller** - is an instrument that can be used for animation management. It contains a state machine inside which every state is an animation clip and every transition from one state to another is a condition that must be met in order to move to another state. [8]

1.7 Unity Timeline

Unity timeline is a tool in Unity that allows to create cutscenes using in-game objects. The timeline object contains the PlayableDirectior component. This component contains methods for controlling the playing of the timeline.

One of the important parts of Unity Timeline is **Animation track** - a track for playing animations.

For creating a cutscene using Unity Timeline, the Animation Track should be created first. Next, a GameObject with the animator controller component should be put into the track. Then, animation clips can be placed on the track line. [6] The totality of the animation clips placed on the track line defines the timeline.

1.8 Unity VideoPlayer

A video player is a game object containing the Video Player component. The Video player component allows to play a video. Let us consider some of the video player important settings:

Video clip - is a video clip that will be played.

Wait for first frame - if this checkbox is enabled, video will be played after the first game frame.

Camera - camera to which the video will be played.

Render mode - camera render modes. It allows to choose how the camera will render the video.

For playing videos using Unity VideoPlayer, it is enough just to put the VideoPlayer inside the scene and put the video asset into the VideoPlayer's "Video clip" parameter window. [6]

1.9 Unity Asset Store

Unity Asset Store is a big store where a lot of assets of different types, both paid and free, can be found.

Chapter 2

EbSynth

EbSynth [12] is a state of the art video synthesis software developed at DCGI FEL CTU. This technology makes it possible to style a sequence of video frames using just a few stylized frames. EbSynth can significantly reduce the time needed for the creation of a graphical part of the game. At the time of writing, the technology is in the alpha version.

Firstly, here EbSynth work principles will be explained. Secondly, the graphical interface of the program will be considered. After that, the usage of EbSynth for the video processing process will be described. At the end of this Chapter there is the analysis of where and how EbSynth technology can be used in the game made using the Unity Engine.

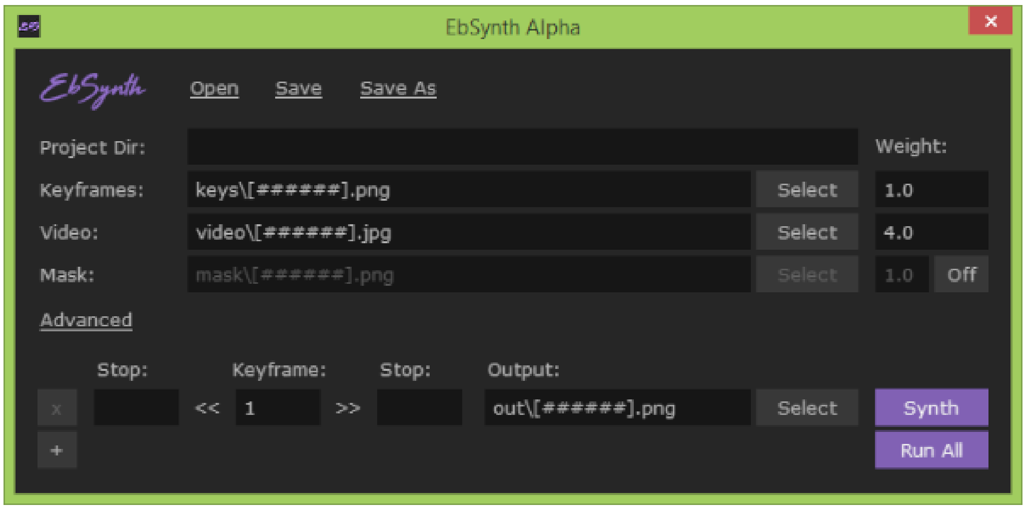
2.1 EbSynth work principles

EbSynth does not use neural networks. It is an example-based video stylizing software. The analogy-based approach allows to stylize the input video into a style preferred by an artist. To stylize the entire video, one or more stylized frames must be provided. This stylized frame is called a keyframe. These keyframes should cover as many objects as possible. It has to be specified manually for which frames the specific keyframes should be used. Using these keyframes and other information, EbSynth algorithms stylize all other frames of the video. If EbSynth encounters an object in the frame that is not present in the related keyframe, it will figure out itself how to synthesize this object. This fact may contribute to an incorrect output. [13]

2.2 Workspace

As we can see in the image 2.2, several elements need to be studied:

- **Keyframes** - here, we need to fit a directory containing stylized frames, from which all other frames will be drawn.
- **Video** - here, we need to put a directory that contains a sequence of video frames.
- **Mask** - is the directory where the frame masks might be placed.
- **Weight** - is the weight of the keyframe or video frame. The higher the weight of the keyframe/video is, the more the processed frame will look like a keyframe/video.
- **Keyframes mapping** - here, we specify for which frames a particular keyframe should be used.
- **Output** - this is the directory where the synthesis results will be added.


EbSynth Alpha

Project Dir:
Keyframes: keys\[######].png — Weight: 1.0
Video: video\[######].jpg — Weight: 4.0
Mask: mask\[######].png — Weight: 1.0 Off

Stop: Keyframe: 1 Stop: Output: out\[######].png

Figure 2.1: EbSynth Workspace

2.3 Processing video

In order to synthesize a video, it should be split into frames using some third-party software, for example, Blender. [14] The video frames should be placed into the video folder. After that, the stylized keyframes should be placed into the keyframes folder. In order to style only a particular area of the frame, frame masks can be added into the mask folder. As soon as all these things are done, the path to all these elements should be specified inside the program. Optionally, the weight settings might be specified. The last step is to map keyframes to the related frames. After doing that, clicking on the synth button will start the video sequence synthesis process. After some time, the synthesized frames will appear in the out folder. In order to compile the video frames sequence into the final video, a third-party software, such as Blender [14] can be used. [13]

2.4 EbSynth usage

Since video-game activities are quite closely related to graphics, EbSynth could significantly reduce the resources spent on creating the graphic part of the game. One of these areas is animation. The video of the movement of a living creature or object can be recorded, split into the frames, synthesized and added to the game. This approach is used by my colleague Jan Klicpera in his work. [3] Another area are the cutscenes. There are different options. One of them is, similarly as it is described above, to record and synthesize the out-game video. The second option is to create, record, and synthesize an in-game video. Unity Timeline can be used for creating a timeline that can be recorded afterward.

Chapter 3

Analysis and design

In this Chapter, the preparation for the proof-of-concept game level implementation will be made. In the first half of this Chapter, the game demo level mechanics analysis and documentation will be created. The requirements will be recorded, the usecases will be created, and the requirements coverage will be controlled. The results of these actions will be recorded in diagrams. In the second half of this Chapter, based on the requirements and usecases from the first part, the game demo level design will be carried out. Firstly, a design for the drawing system will be suggested. Secondly, a simple game design document will be compiled. Based on this document, a class diagram, a component diagram, as well as an activity diagram describing the implementation logic will be made up and described. The demo game level objects will be shown and described. Also, there will be a design for the graphical interface of the game. The result of this Chapter will be the documented analysis and design of the game demo level mechanics.

3.1 Before starting

Due to the lack of 2D sprites suitable for the game, as well as by reason of the desire to study and implement a 3D game creation, it was decided to make the proof-of-concept implementation in 3D. However, the drawing mechanics will still be implemented for 2D sprites. The possibilities of usage of this mechanic in the 3D game world will be described in section 3.6.1.

3.2 Functional requirements

Let us make a list of functional requirements for mechanics.

3.2.1 Drawing system

F1.1 - Drawing on sprite:

It should be possible to draw on 2D sprites. Drawing should be carried out by clicking the mouse or clicking on the touch screen of the phone. Also, continuous drawing should be possible without releasing the mouse/finger. The picture should be displayed only on the sprite on which it was drawn (sprites should not be interconnected). Pictures may not be saved on sprites when restarting the game.

F1.2 - The choice of color:

The player must be able to choose the color with which they will draw. The color should be selected in a separate game window (drawing tools window). The choice of color in the color palette should be carried out by clicking in the palette area.

F1.3 - Color picking:

The color picker tool has to be implemented. The tool should determine the pixel color at some point and set this color to the actual color of drawing. The pixel for determining the color is selected by clicking on the sprite. If the pixel is transparent, the color does not change.

F1.4 - Erasing drawing:

It should be possible to erase the drawing. Erasing the picture is carried out by clicking the mouse or clicking on the touch screen of the phone. Also, continuous erasing should be possible without releasing the mouse/finger. Erasing is carried out by using a brush of a certain shape.

F1.5 - The choice of brush:

It should be possible to select a brush that implements a specific shape for drawing. It is necessary to implement at least three brushes (circle, square, triangular).

F1.6 - Drawing at an angle:

Drawing should be possible and should be correct for any Y axis rotation of the sprite.

F1.7 - Import of existing brushes:
It is necessary to implement a mechanism for importing brushes from files in the file system. It is necessary to come up with a file structure that contains a brush. Brushes import is available only for game developers.

3.2.2 GUI

F2.1 - Current tool icon on screen:
The currently selected drawing tool should be displayed on the game screen.

F2.2 - Drawing tools window:
It is necessary to implement a window that will contain all the drawing tools. There should be a color palette, an available brushes list, an eraser switcher, and a color picker switcher in this window. The drawing menu should be opened by clicking on the button in the right upper corner of the screen.

F2.3 - Settings window:
It is necessary to implement a window with the game settings. The settings window should be accessible from the main menu.

F2.4 - Joysticks inversion setting:
It is necessary to implement the possibility to swap joysticks (available only for Android OS version).

F2.5 - Control sensitivity changing:
It is necessary to implement the ability to change the player speed from the settings window.

F2.6 - Rotation sensitivity changing:
It is necessary to implement the ability to change the player rotation speed from the settings window.

F2.7 - Map button on screen:
It is necessary to implement the map button. This button should be visible only in Android OS version.

F2.8 - Current color on screen:

The actual selected color should be visible on the game level view screen.

F2.9 - Joysticks for Android version:
It is necessary to implement graphic elements - joysticks. The joysticks should be available only for the Android OS version.

F2.10 - Controls list:
The game settings window should contain a list of controls settings. It should not be allowed to change the game settings, with the exception of a few parameters described in other requirements.

F2.11 - Show minimap on screen setting:
It should be possible to enable/disable the minimap in the settings menu.

F2.12 - Game menu:
It is necessary to implement a game menu. It is necessary that the game menu is available while the game is running.

3.2.3 Control

F3.1 - Movement acceleration:
It is necessary to implement the player movement (available only in the PC version).

F3.2 - Movement controlling:
It is necessary to implement a system for controlling the movement of an object.

F3.3 - Game pause:
There should be an option to pause the game.

F3.4 - Rotation controlling:
It is necessary to add the rotation of the object.

3.2.4 Map

F4.1 - Player position on map: The position of the player should be visible on the game map. The position mark must move with the player and always correspond to the position of the player at the game level.

F4.2 - Minimap:
It is necessary to implement the minimap with the player position.

F4.3 - Minimap rotation: The minimap should rotate with the player and correspond with the player view direction.

F4.4 - Map:
It is necessary to implement a game level map.

3.2.5 Cutscenes

F5.1 - Cutscenes playing:
It is necessary to implement a system that plays a certain, pre-implemented cutscene. During the cutscene, control must be turned off. The cutscene should be played only once per game.

F5.2 - Playing video:
It is necessary to add the possibility to play a short video at the beginning of the game, before the start of the level.

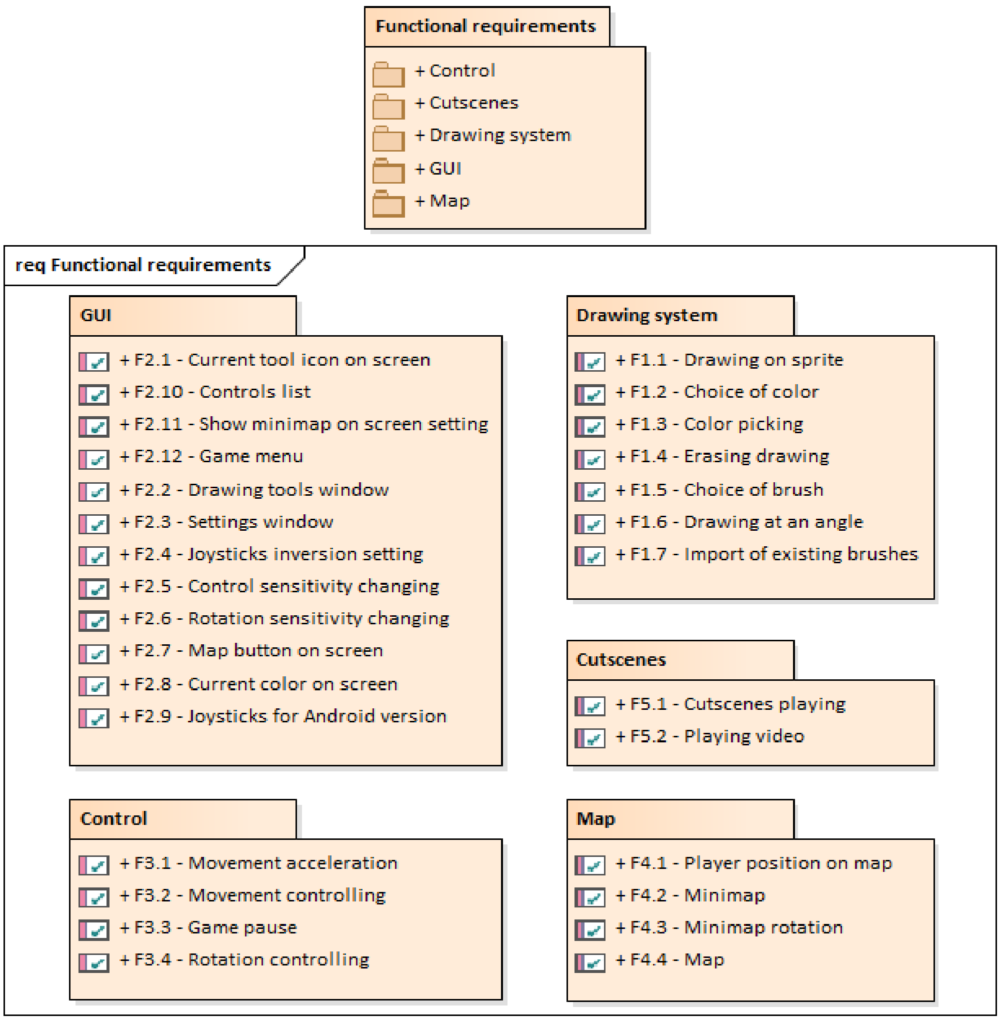

Functional requirements
+ Control
+ Cutscenes
+ Drawing system
+ GUI
+ Map
req Functional requirements
GUI
+ F2.1 - Current tool icon on screen
+ F2.10 - Controls list
+ F2.11 - Show minimap on screen setting
+ F2.12 - Game menu
+ F2.2 - Drawing tools window
+ F2.3 - Settings window
+ F2.4 - Joysticks inversion setting
+ F2.5 - Control sensitivity changing
+ F2.6 - Rotation sensitivity changing
+ F2.7 - Map button on screen
+ F2.8 - Current color on screen
+ F2.9 - Joysticks for Android version
Drawing system
+ F1.1 - Drawing on sprite
+ F1.2 - Choice of color
+ F1.3 - Color picking
+ F1.4 - Erasing drawing
+ F1.5 - Choice of brush
+ F1.6 - Drawing at an angle
+ F1.7 - Import of existing brushes
Cutscenes
+ F5.1 - Cutscenes playing
+ F5.2 - Playing video
Control
+ F3.1 - Movement acceleration
+ F3.2 - Movement controlling
+ F3.3 - Game pause
+ F3.4 - Rotation controlling
Map
+ F4.1 - Player position on map
+ F4.2 - Minimap
+ F4.3 - Minimap rotation
+ F4.4 - Map

Figure 3.1: Functional requirements

3.3 Non-Functional requirements

Let us make a list of non-functional requirements that should be carried out by mechanics.

N1 - Easy and intuitive controls:

The game controls should be simple and intuitive for the player.

N2 - Simple installation:

Installing of this mechanics (system) into an existing game, according to certain conditions, should not be complicated.

N3 - Android OS Support:

Game should support Android OS.

N4 - Adaptation to different screen resolutions:

It is necessary to implement the game the way it can scale to different screen resolutions (first of all, we are talking about mobile devices).

N5 - Level loading speed:

The loading of the game level should not be long. The maximum level loading time is 15 seconds.

N6 - Wide choose of colors for drawing:

It is necessary to provide the player with a wide selection of colors for drawing. The minimum number of colors is 20.

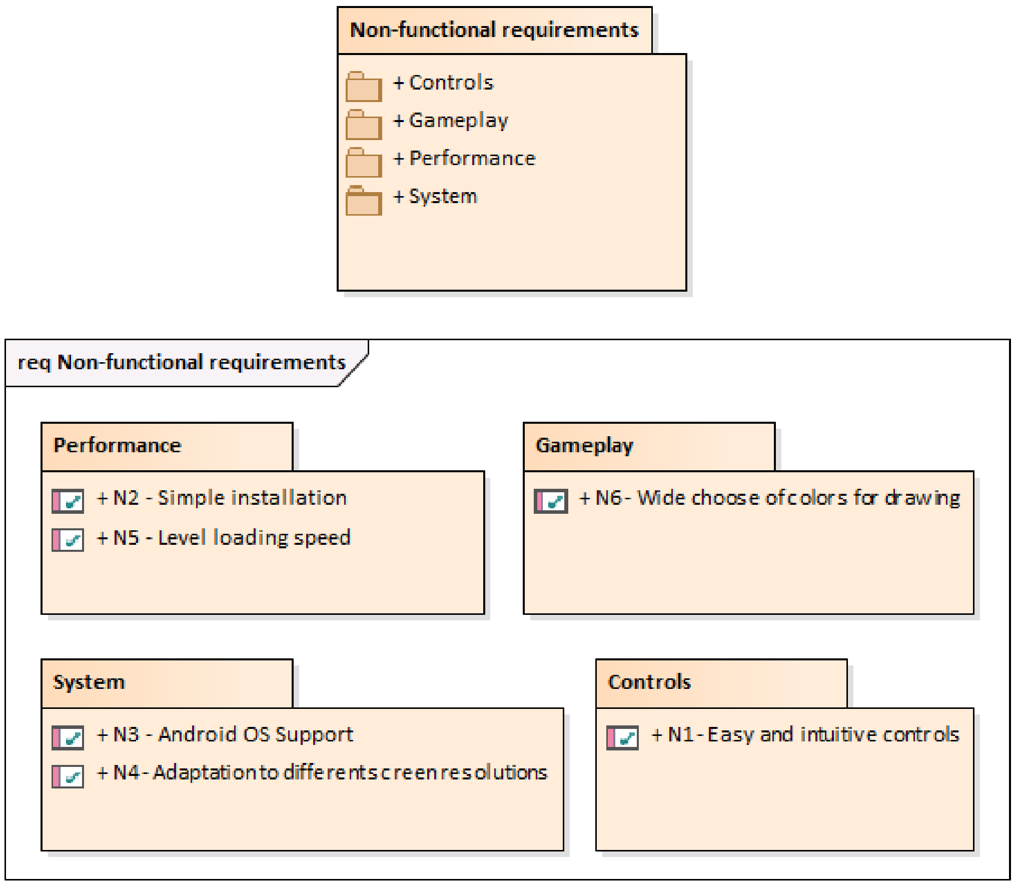

Non-functional requirements
+ Controls
+ Gameplay
+ Performance
+ System
req Non-functional requirements
Performance
+ N2 - Simple installation
+ N5 - Level loading speed
Gameplay
+ N6 - Wide choose of colors for drawing
System
+ N3 - Android OS Support
+ N4 - Adaptation to different screen resolutions
Controls
+ N1 - Easy and intuitive controls

Figure 3.2: Non-functional requirements

3.4 Usecases

List of the usecases that should be implemented.

3.4.1 Drawing system

UC1.1 - Sprite point color pick

This usecase allows player to pick the color of some point on the sprite in order to paint with that color in future.

The player opens the drawing tools window. The player clicks on the color picker tool checkbox. The player closes the drawing tools window. The player clicks on some point on a sprite. The color of this point will be picked.

UC1.2 - Drawing on inclined surface

This usecase shows a possibility to draw on sprites that are not perpendicular to the ground.

The player should find some inclined sprite. The player draws something on sprite the same way as it written in UC1.4.

UC1.3 - Sprite part erasing

This usecase allows the player to erase a specific area of the sprite. This usecase may be necessary, for example, if the player wants to erase something they painted before.

The player opens the drawing tools window. The player clicks on the eraser tool checkbox. The player closes the drawing tools window. Holding the left mouse button or finger on the phone screen, the player erases the areas they want.

UC1.4 - Drawing on sprite

This usecase allows the player to draw something on a sprite.

The player opens the drawing tools window, chooses color, chooses brush. The player closes the drawing items window. The player draws something on sprite.

UC1.5 - Brushes importing from file

This usecase allows the game developer to import a brush from a file in the file system. It is also necessary to come up with the brushes file structure.

The developer writes brushes directory path into the special place. The game will import brushes from the files from the directory at the game start.

3.4.2 GUI

UC2.1 - Starting game

This usecase gives the player an opportunity to enter the main game level.

The player clicks on the start game button in the main menu. The game will start.

UC2.2 - Exiting game

This usecase allows the player to exit the game.

The player clicks on the quit game button in the main menu. The game will stop working.

UC2.3 - Changing moving sensitivity

This usecase gives the player the ability to change the moving speed settings as they want. This usecase is necessary as it adds some diversity to the control system.

The player opens the setting menu window. The player moves ”Movement sensitivity” slider as they want. Movement speed will change.

UC2.4 - Changing rotation sensitivity

This usecase gives the player the ability to change the rotation speed settings as they want. This usecase is necessary as it adds some diversity to the control system.

The player opens the setting menu window. The player moves ”Rotation sensitivity” slider as they want. Rotation speed will change.

UC2.5 - Picking color from color palette

This usecase allows player to pick the color from color palette in order to paint with that color in future.

The player opens the drawing tools window. The player clicks on the color palette. The color that is located at the player click position will be picked.

UC2.6 - Actual instrument discovering

This usecase allows the player to discover which drawing instrument is enabled at the moment.

The player can see the actual enabled instrument in the top right corner of the game window.

UC2.7 - Player rotation in Android OS version

This usecase gives the player an opportunity to rotate around the Y axis in Android OS game version.

The player rotates around the Y axis using one of the two joysticks on the screen.

UC2.8 - Menu window opening/closing

This usecase shows the opportunity to open/close the main menu window.

The player clicks on the ”Settings” button in the main menu. The settings window will be opened. The player presses the Escape or Back button. The settings window will be closed.

UC2.9 - Turning on/off color picker instrument

This usecase gives the player an opportunity to turn on/off the color picker tool.
The player opens the drawing tools window. The player clicks on the color picker tool checkbox. The color picker tool is enabled. The player clicks on the color picker tool checkbox again. The color picker tool is disabled now.

UC2.10 - Player moving in Android OS version

This usecase gives the player an opportunity to explore the game level in Android OS game version.

The player moves around the game level using joystick on the screen.

UC2.11 - Map opening/closing

This usecase shows the opportunity to open/close the map.

The player presses the ”M” keyboard button or clicks on the ”Map” button (in Android OS version). The map window will be opened. The player presses the Escape or Back button. The map window will be closed.

UC2.12 - Settings window opening/closing

This usecase shows the opportunity to open/close the settings menu.

The player clicks on the ”Settings” button in the main menu. The settings window will be opened. The player presses the Escape or Back button. The settings window will be closed.

UC2.13 - Actual color discovering

This usecase allows the player to discover which color is selected.

The player can see the actual color in the top right corner of the game window.

UC2.14 - Turning on/off eraser instrument

This usecase gives the player an opportunity to turn on/off the eraser tool.

The player opens the drawing tools window. The player clicks on the eraser tool checkbox. The eraser tool is enabled. The player clicks on the eraser tool checkbox again. The eraser tool is disabled now.

UC2.15 - Controls settings discovering

This usecase gives the player an opportunity to discover the game controls settings.

The player opens the setting menu window. The player discovers the game controls settings.

UC2.16 - Minimap enabling/disabling

This usecase shows the opportunity to enable/disable the minimap.

The player opens the setting menu window. The player clicks on the ”Minimap enabled” checkbox. The minimap is enabled and visible in the game view window. The player clicks on the ”Minimap enabled” checkbox again. The minimap is disabled now.

UC2.17 - Drawing tools window opening/closing

This usecase allows the player to open/close the drawing tools window. This window contains all drawing tools.

The player clicks on the button in the top right corner of the game view window. The drawing tools window is opened. The player clicks on the button again. The window is closed now.

UC2.18 - Joysticks inverting

This usecase shows the possibility to swap joysticks in Android OS version. This usecase is necessary as it adds some diversity to the control system.

The player opens the settings menu window. The player clicks on the ”Invert joysticks” checkbox. The joysticks are swapped now.

UC2.19 - Movement acceleration in Android OS version

This usecase shows the opportunity to accelerate moving speed of the player in Androis OS game.

The player leads the joystick to the joystick border.

3.4.3 Movement system

UC3.1 - Player rotation

This usecase gives the player an opportunity to rotate around the Y axis.

The player rotates around the Y axis using QE keyboard buttons.

UC3.2 - Moving acceleration

This usecase shows the opportunity to accelerate moving speed of the player.

The player holds the ”Left shift” keyboard button.

UC3.3 - Moving when map is enabled

This usecase shows the opportunity to move the player when the map window is enabled.
The player opens map. The player moves the same way as described in UC3.4.

UC3.4 - Player moving

This usecase gives the player an opportunity to explore the world using movement system.

The player moves around the game level using WASD keyboard buttons.

3.4.4 Map

UC4.1 - Discovering player rotation using minimap

This usecase allows the player to discover their direction using the minimap.

The player can see their direction on the minimap that is located in the top left corner.

UC4.2 - Discovering player position on map

This usecase allows the player to discover their position at the game level using the map.

The player opens the map as described in UC2.11. The player discovers their position. The player closes the map.

UC4.3 - Discovering player position on minimap

This usecase allows the player to discover their position at the game level using the minimap.

The player can see their position on the minimap that is located in the top left corner.

3.4.5 Cutscenes system

UC5.1 - Playing video at the game start

This usecase allows the developer to play some video at the game start.

The developer puts some video into the special place. The video will be played at the game start.

UC5.2 - Playing cutscene when reaching some trigger position

This usecase will play some cutscene when the player reaches some trigger on the map.

3.4.6 Actors

Actors participating in usecases: **Player**, **Developer**. Using 3.3 and 3.4 usecase diagrams, let us consider which actors are participating in usecases. As we can see, most of the usecases are made for the player.

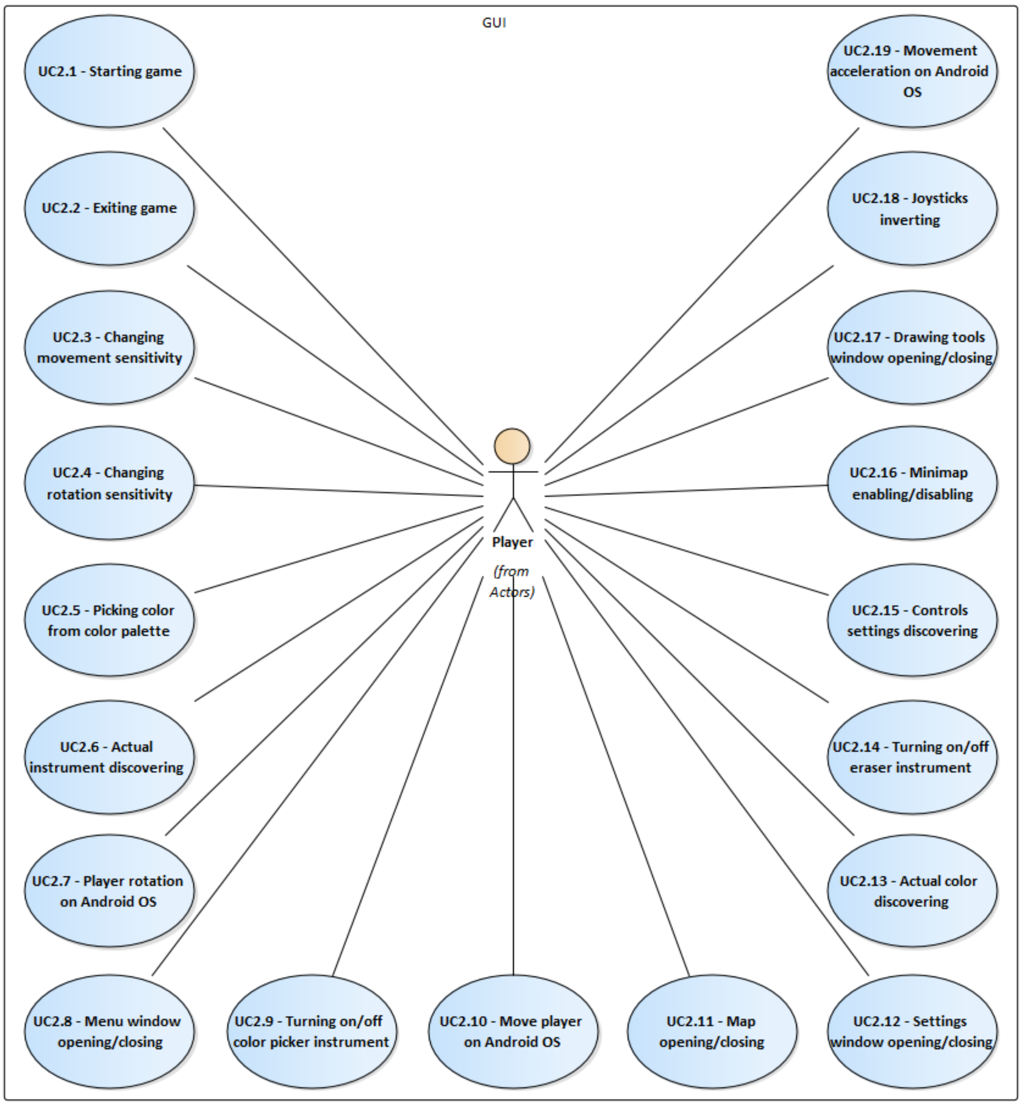

GUI
UC2.1 - Starting game
UC2.2 - Exiting game
UC2.3 - Changing movement sensitivity
UC2.4 - Changing rotation sensitivity
UC2.5 - Picking color from color palette
UC2.6 - Actual instrument discovering
UC2.7 - Player rotation on Android OS
UC2.8 - Menu window opening/closing
UC2.9 - Turning on/off color picker instrument
UC2.10 - Move player on Android OS
UC2.11 - Map opening/closing
UC2.12 - Settings window opening/closing
UC2.13 - Actual color discovering
UC2.14 - Turning on/off eraser instrument
UC2.15 - Controls settings discovering
UC2.16 - Minimap enabling/disabling
UC2.17 - Drawing tools window opening/closing
UC2.18 - Joysticks inverting
UC2.19 - Movement acceleration on Android OS
Player
(from Actors)

Figure 3.3: GUI usecase diagram

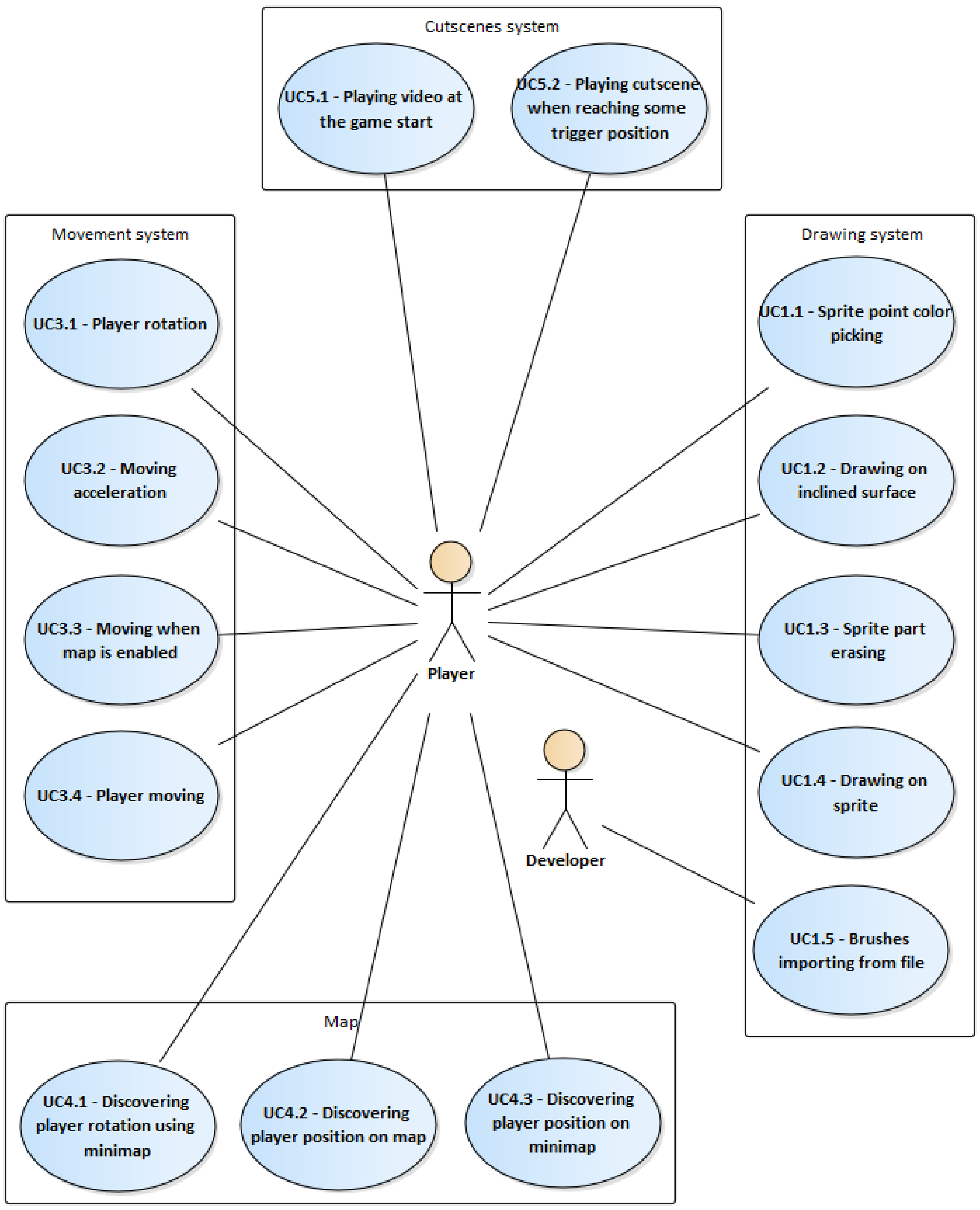
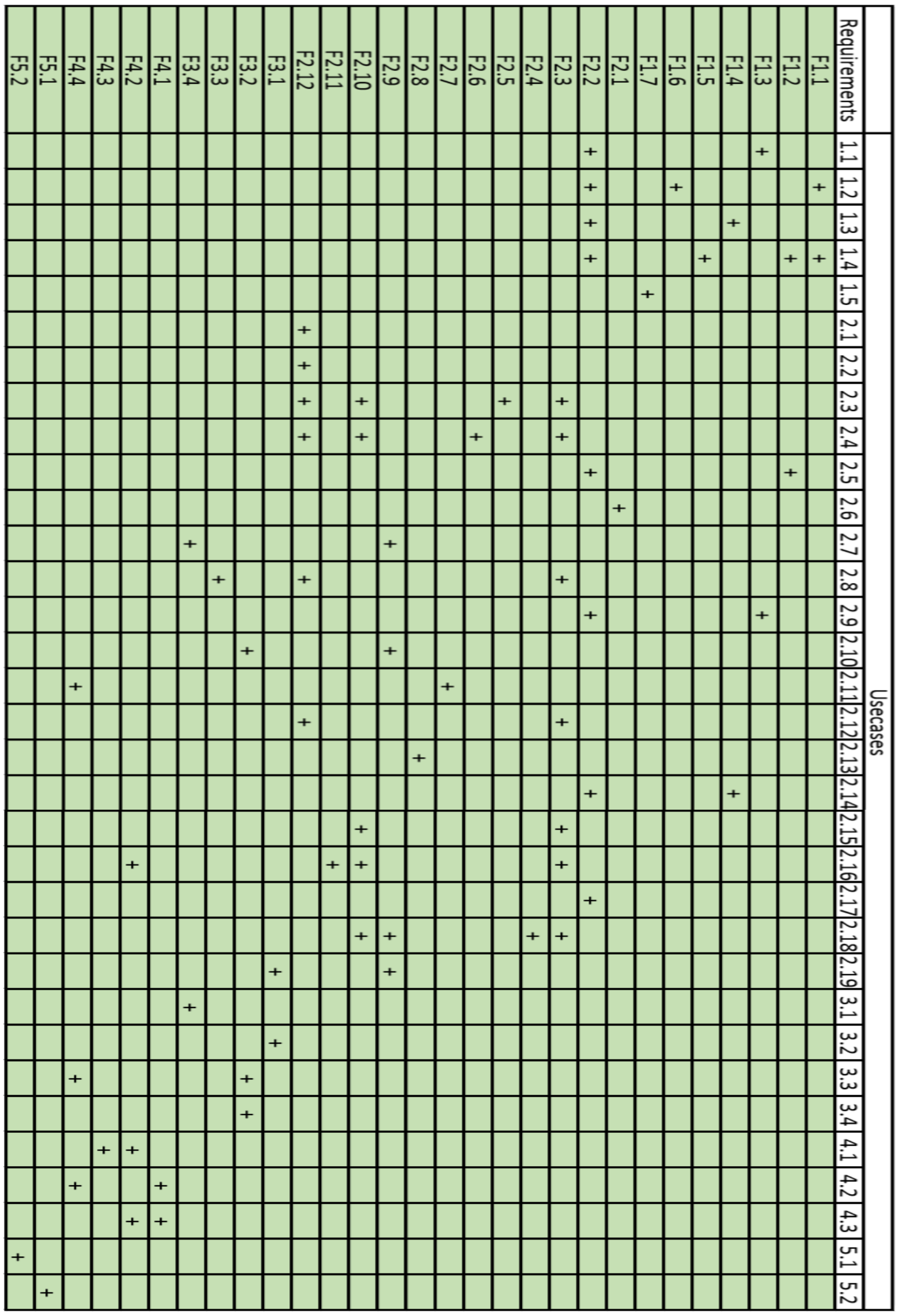

Cutscenes system
UC5.1 - Playing video at the game start
UC5.2 - Playing cutscene when reaching some trigger position
Movement system
UC3.1 - Player rotation
UC3.2 - Moving acceleration
UC3.3 - Moving when map is enabled
UC3.4 - Player moving
Drawing system
UC1.1 - Sprite point color picking
UC1.2 - Drawing on inclined surface
UC1.3 - Sprite part erasing
UC1.4 - Drawing on sprite
UC1.5 - Brushes importing from file
Player
Developer
Map
UC4.1 - Discovering player rotation using minimap
UC4.2 - Discovering player position on map
UC4.3 - Discovering player position on minimap

Figure 3.4: Cutscenes system usecase diagram. Movement system usecase diagram. Drawing system usecase diagram. Map usecase diagram.

3.5 Requirements fulfillment

As we can see in the table 3.5, all requirements were covered by usecases.



	Usecases																															
Requirements	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	2.11	2.12	2.13	2.14	2.15	2.16	2.17	2.18	2.19	3.1	3.2	3.3	3.4	4.1	4.2	4.3	5.1	5.2
F1.1		+		+																													
F1.2				+						+																							
F1.3	+													+																			
F1.4			+																+														
F1.5				+																													
F1.6		+																															
F1.7					+																												
F2.1											+																						
F2.2	+	+	+	+						+				+					+			+											
F2.3								+	+				+				+			+	+		+										
F2.4																							+										
F2.5								+																									
F2.6									+																								
F2.7																+																	
F2.8																		+															
F2.9												+			+								+	+									
F2.10								+	+											+	+		+										
F2.11																					+												
F2.12						+	+	+	+				+				+																
F3.1																								+		+							
F3.2															+												+	+					
F3.3													+																				
F3.4												+													+								
F4.1																														+	+		
F4.2																					+								+		+		
F4.3																													+				
F4.4																+											+			+			
F5.1																																	+
F5.2																																+	

Figure 3.5: Functional requirements fulfillment diagram

3.6 Drawing mechanic

This section will cover the main components and details of the drawing system. The drawing principles will be described. Next, the problem of finding the position of the pixel that the player clicked on will be considered. Options for solving this problem will be proposed. Also, a graphic element called ”Color palette”, and the problem of finding a player’s click position on it will be formulated. Finally, the principle of the brush will be described.

3.6.1 Main principle

According to the information described in section 1.2, the texture is a two-dimensional array of pixels. Based on this information, the drawing effect can be achieved by changing the color of the pixel that the player clicked on. At this stage, the issue of finding the position of the pixel that the player clicked on arises.

Since there are no 2D sprites for drawing in this game, it was decided to make an invisible (transparent) sprite on which it will be possible to draw. This sprite will be placed on some buildings walls. Thus, the drawing on the walls of 3D buildings effect will be achieved.

3.6.2 Click position detection

This problem can be divided into two stages. The first is to get the click position in world coordinates. This stage can be performed using the Ray-cast described in section 1.5. The second stage is to convert the position into two-dimensional texture pixel array coordinates.

3.6.3 Brush

One way to implement a paintbrush is to create a pixel map. [15] A pixel map can be implemented using a two-dimensional array of colors. However, to simplify the task, a version using an array of bits - a bitmap - will be used. [16]

3.6.4 Color palette

A color palette is a tool using which a color for drawing can be selected. This tool consists of areas containing various colors. One of the ways to implement the color palette is to place an image with colors on the UI canvas component. When choosing this method, a click position detection problem, similar to what was mentioned above (in Chapter 3.6.2), comes up. However, because the color picker element is a GUI element, the solution will differ.

The solution is complicated because several elements and parameters should be taken into account. The first thing to consider is that the image of the color palette is on the canvas, which has its own scale value. The image itself also has a certain scale. The position of the palette object is set using the scale of the canvas. However, the values of the width and height that we can get do not take into account the scale, so it is necessary to make certain transformations and calculations.

3.7 Game design document

This section will describe the basic mechanics, structure, graphical interface, as well as additional features of the demo level. The result will be the design and documentation in the form of GDD.

Before proceeding to writing the GDD, it is worth noting that since the demo level is not a full-fledged game, the structure and content of this document will be simplified.

3.7.1 Basic gameplay

The gameplay of the demo level will consist of the player moving and rotating around the level using the keyboard or joysticks and studying various objects at the level. When entering the level there will be shown a small video showing the level. One of the main actions inside the level will be the drawing on the walls of buildings.

- **Movement**. The movement will be carried out using the WASD buttons or the joysticks (in the case of the version for Android). The game movement will be carried out in four directions: forward (W or joystick up), backward (S or joystick down), left (A or joystick left), and right (or D or joystick right).

  The game will have the opportunity to speed up the movement. For doing this, the left shift key should be pressed. In the case of the version for Android OS, the joystick should be moved to the edge in the direction of movement.

- **Rotation**. Rotation is carried out using the QE buttons or, in the case of the version for Android, using the joysticks. Rotation will be possible only in two directions: left (Q or joystick left) and right (E or joystick right).

- **Drawing**. Drawing is carried out by holding the left mouse button (or finger on the phone screen, in the case of the Android OS) and drawing shapes on the sprite. Also, it will be possible to change the paintbrush. A drop-down list of available brushes will be located in the drawing tools window. Also, there will be an opportunity to erase the drawings. For doing this, the eraser tool must be turned on. Also, it is possible to take the color of a sprite point. For doing this, the color picker tool should be turned on. Turning on and off the eraser and color picker tools is possible in the drawing tools window. Another element of the drawing tool window is a color palette that allows to select a color. A color selection is carried out by clicking on a specific color on the color palette.

  Brushes will draw certain shapes (for example, a square or a smiley). Brushes will not be able to resize.

- **Cutscene**. Before entering the level, a short video that partially shows the objects at the level will be played. This video will be made in a drawn style, using the EbSynth technology. After entering the level, going forward and reaching a certain trigger point, another cutscene which also shows parts of the level will be played.

3.7.2 Level structure

. The level will be a small town located in a sandy area surrounded by mountains. There will be two types of objects inside the level:

1. **Environment objects** - these are objects that form the picture of the city (houses, cars, traffic lights, etc.). The player cannot interact with these objects.

2. **Interaction objects** - these are objects with which the player can interact, for example, sprites on which they will draw. Also, there will be invisible objects that will also interact with the player, for example, an invisible timeline object that will start playing the cutscene when the player will enter the trigger.

3.7.3 GUI

- **Drawing tools button** - this button will open/close the drawing toolbar. The button will be located in the top right corner of the screen.
- **Map button** (available only for Android OS game version) - this button will open/close the map. The button will be at the bottom of the screen.
- **Joysticks**(available only for Android OS game version) - these are joysticks for controlling the character's movement and rotation. Joysticks will be placed at the bottom, on the left and right sides of the screen
- **Drawing tools window** - a panel containing drawing tools such as:
  - Color palette - the color palette needed for color selection. The palette will be placed on the left side of the toolbar.
  - Available brushes list - available brushes list. There will be the brush that is selected now at the first place of the list. The list will be to the right of the color palette.
  - Color picker checkbox - This checkbox turns on/off the color picker tool. The item will be on the right side of the drawing tools window.
  - Eraser checkbox - This checkbox turns on/off the eraser tool. The item will be on the right side of the drawing tools window.
- **Actual instrument selected** - the image with the currently selected tool will be on the drawing tools button.
- **Actual color selected** - This is a small square containing the color that is currently selected. This element will be located in the lower right part of the drawing tools button.

Images with GUI design can be found in attachments.

3.7.4 Game menu

. The game menu consists of two windows, and the first is the **main menu** window.

The game **main menu** is a window that contains three buttons:

- **Start game button** - this button will start the new game level.
- **Settings button** - this button will open the settings window.
- **Quit game button** - this button will stop the game.

All buttons will be in the middle of the screen

The second window - is the **settings** window. This window contains the following items:

- Movement sensitivity slider - this is a slider for changing the character speed (the righter the slider, the greater the speed).
- Rotation sensitivity slider - this is a slider for changing the character rotation speed (the right, the greater the speed).
- Invert joysticks checkbox (available only for Android OS game version) - this is a checkbox, the choice of which swaps the joysticks on the screen.
- Minimap enabled checkbox - selection of this checkbox turns on/off the minimap displaying.
- Controls values - actual controls setting are shown here. (the settings differ depends on platform).

Images of the game menu and the settings window design can be found in attachments.

3.7.5 Features

- **Map and minimap**. On the game map (as well as on the minimap) the game level objects will be displayed. Also, there will be the player's position in the middle of the map. The minimap will be located in the upper left corner and will rotate with the player. The minimap can be turned on and off using the appropriate checkbox in the game settings window. The map can be turned on and off using the map button. In the case of a PC, this is the M button. In the case of the Android OS, this button is at the bottom of the screen.

3.8 UML class diagrams

Before proceeding to the implementation, it is necessary to describe the structure of the classes used to implement the mechanics, as well as describe how they interact with each other.

Let us consider the structure of classes using the UML class diagram.

At the diagram 3.6 we can see a class diagram for drawing system. It contains six classes:

1. **DrawingController** - This is the main class, the controller, for the drawing system. This class implements the drawing on sprite. In addition to drawing, this class is responsible for detecting a click on a sprite, calculating the position of a click on the image (conversion from 2D/3D coordinates of the click to 2D coordinates of the image). Also, this class is responsible for the initial initialization of the sprite, on which drawing will be possible. This class actively interacts with the DrawingPalette class to obtain relevant brushes, tools and colors.

2. **IdGenerator** class is used for generating unique ID that will be used in copied textures names in DrawingController class.

3. **Brush** - is a class that contains fields for storing paint brushes.

4. **DrawingPalette** - is the class containing the implementation of the toolbar for drawing. The class contains getters and setters for various drawing tools, as well as brush switches. This class is used by DrawingController class.

5. **BrushImporter** - class containing the implementation of the import of brushes into the drawing toolbar(DrawingPalette).

6. **DefaultBrushes** is a class that contains that methods for three default brushes creating.

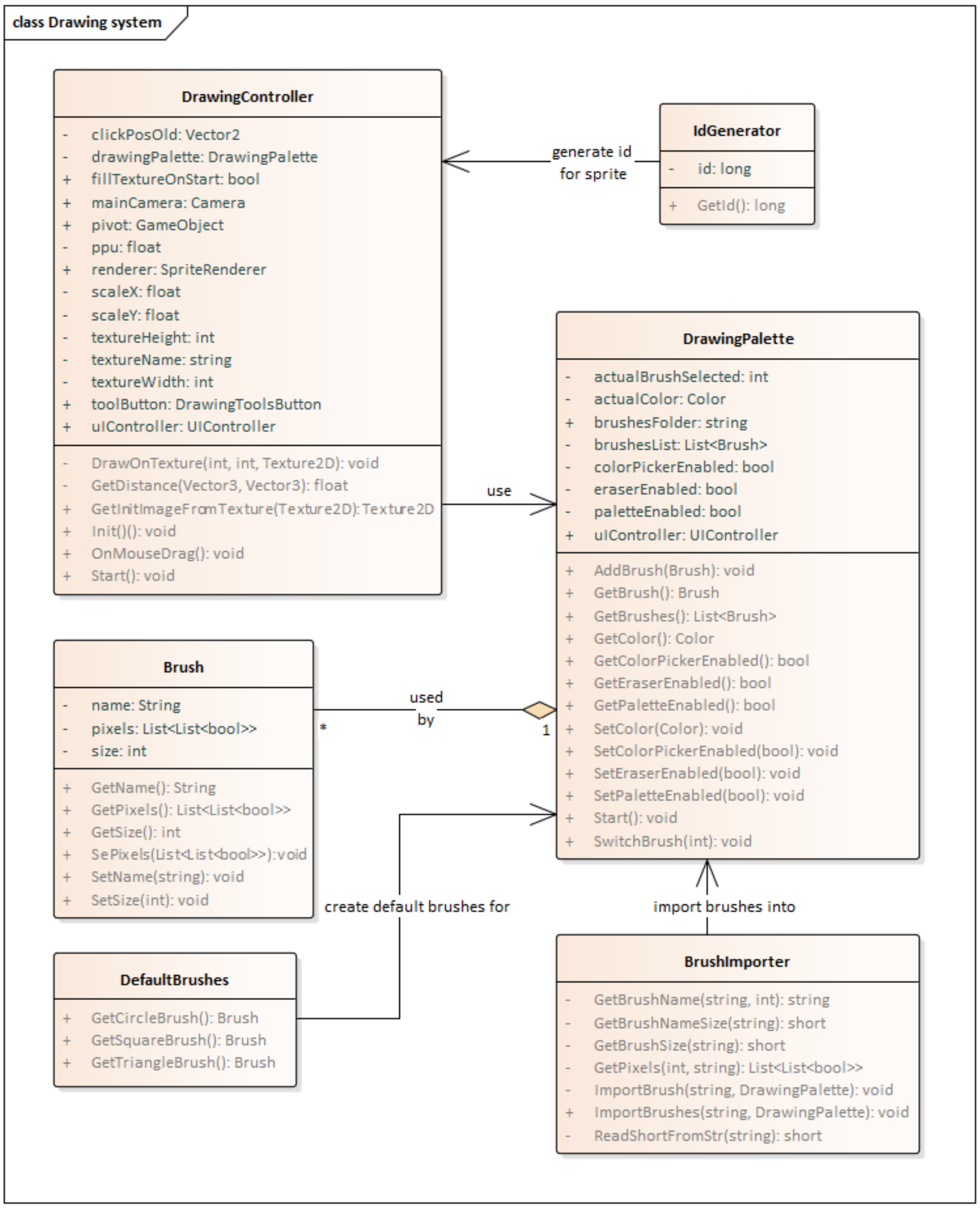

class Drawing system
DrawingController
- clickPosOld: Vector2
- drawingPalette: DrawingPalette
+ fillTextureOnStart: bool
+ mainCamera: Camera
+ pivot: GameObject
- ppu: float
+ renderer: SpriteRenderer
- scaleX: float
- scaleY: float
- textureHeight: int
- textureName: string
- textureWidth: int
+ toolButton: DrawingToolsButton
+ uIController: UIController
- DrawOnTexture(int, int, Texture2D): void
- GetDistance(Vector3, Vector3): float
+ GetInitImageFromTexture(Texture2D): Texture2D
+ Init()(): void
+ OnMouseDrag(): void
+ Start(): void
IdGenerator
- id: long
+ GetId(): long
generate id for sprite
use
DrawingPalette
- actualBrushSelected: int
- actualColor: Color
+ brushesFolder: string
- brushesList: List<Brush>
- colorPickerEnabled: bool
- eraserEnabled: bool
- paletteEnabled: bool
+ uIController: UIController
+ AddBrush(Brush): void
+ GetBrush(): Brush
+ GetBrushes(): List<Brush>
+ GetColor(): Color
+ GetColorPickerEnabled(): bool
+ GetEraserEnabled(): bool
+ GetPaletteEnabled(): bool
+ SetColor(Color): void
+ SetColorPickerEnabled(bool): void
+ SetEraserEnabled(bool): void
+ SetPaletteEnabled(bool): void
+ Start(): void
+ SwitchBrush(int): void
Brush
- name: String
- pixels: List<List<bool>>
- size: int
+ GetName(): String
+ GetPixels(): List<List<bool>>
+ GetSize(): int
+ SePixels(List<List<bool>>):void
+ SetName(string): void
+ SetSize(int): void
used by
*
1
create default brushes for
import brushes into
DefaultBrushes
+ GetCircleBrush(): Brush
+ GetSquareBrush(): Brush
+ GetTriangleBrush(): Brush
BrushImporter
- GetBrushName(string, int): string
- GetBrushNameSize(string): short
- GetBrushSize(string): short
- GetPixels(int, string): List<List<bool>>
- ImportBrush(string, DrawingPalette): void
+ ImportBrushes(string, DrawingPalette): void
- ReadShortFromStr(string): short

Figure 3.6: Drawing system class diagram

Further, the 3.7 diagram contains the classes necessary to control the GUI part. Let us take a closer look at these classes:

1. **UIController** - This is the central class (controller) for managing GUI elements. This class has many direct links to GUI objects. The class contains many functions for enabling, disabling, and initializing various GUI elements. Almost all other components interact with this class.

2. Class **BrushesListController** is the controller for the list of brushes in the toolbar for painting. This class initializes a list of brushes, and also responds to the selection of a brush from the list.

3. **DrawingToolsButton** - This is a class that implements the logic of the drawing tools button. In this class we can turn on/off the drawing tools button window. Also we can change the button image.

4. **ColorPalette** - the class contains the implementation of the logic of taking colors from the color palette.

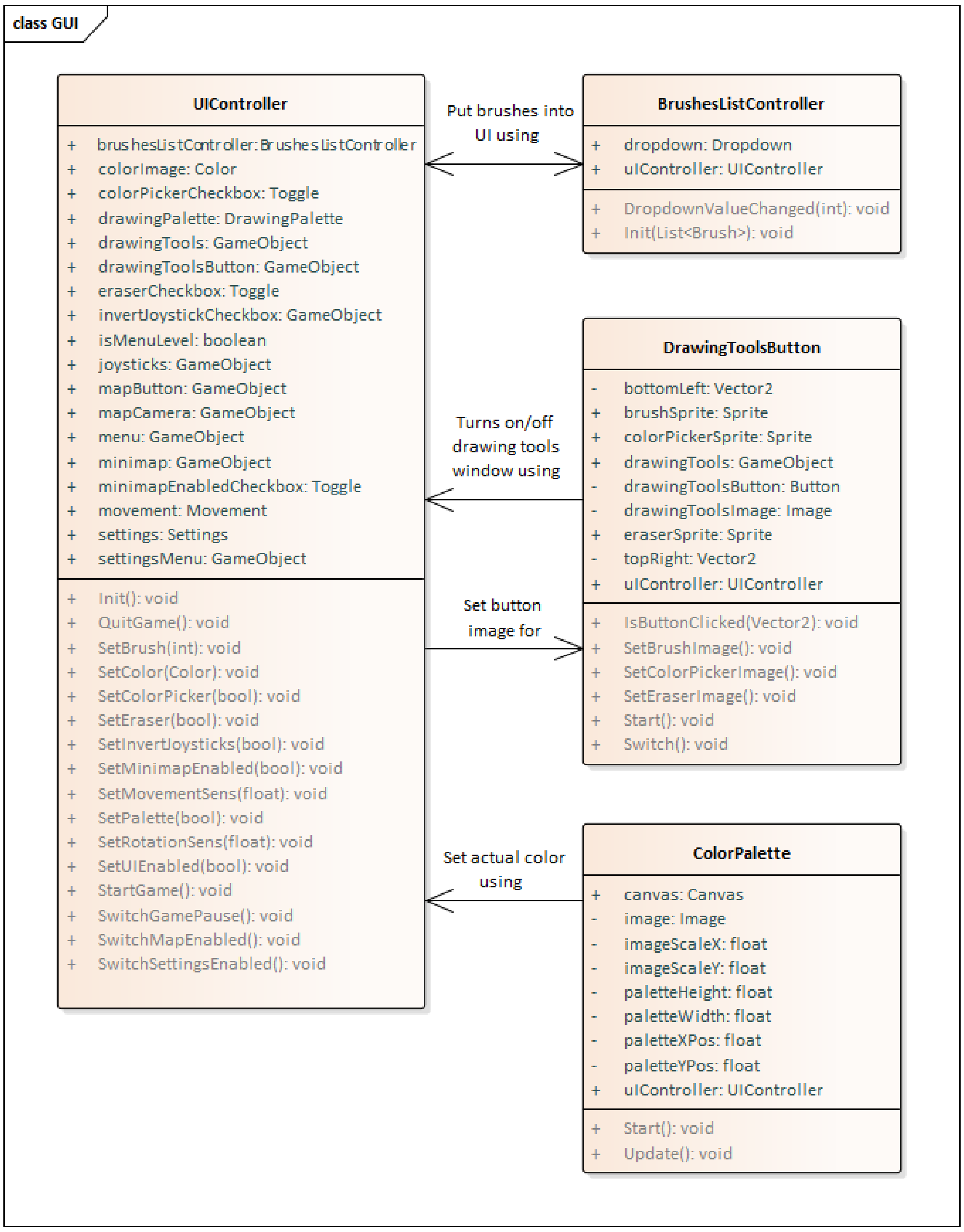

class GUI
UIController
+ brushesListController: BrushesListController
+ colorImage: Color
+ colorPickerCheckbox: Toggle
+ drawingPalette: DrawingPalette
+ drawingTools: GameObject
+ drawingToolsButton: GameObject
+ eraserCheckbox: Toggle
+ invertJoystickCheckbox: GameObject
+ isMenuLevel: boolean
+ joysticks: GameObject
+ mapButton: GameObject
+ mapCamera: GameObject
+ menu: GameObject
+ minimap: GameObject
+ minimapEnabledCheckbox: Toggle
+ movement: Movement
+ settings: Settings
+ settingsMenu: GameObject
+ Init(): void
+ QuitGame(): void
+ SetBrush(int): void
+ SetColor(Color): void
+ SetColorPicker(bool): void
+ SetEraser(bool): void
+ SetInvertJoysticks(bool): void
+ SetMinimapEnabled(bool): void
+ SetMovementSens(float): void
+ SetPalette(bool): void
+ SetRotationSens(float): void
+ SetUIEnabled(bool): void
+ StartGame(): void
+ SwitchGamePause(): void
+ SwitchMapEnabled(): void
+ SwitchSettingsEnabled(): void
Put brushes into UI using
BrushesListController
+ dropdown: Dropdown
+ uIController: UIController
+ DropdownValueChanged(int): void
+ Init(List<Brush>): void
Turns on/off drawing tools window using
DrawingToolsButton
- bottomLeft: Vector2
+ brushSprite: Sprite
+ colorPickerSprite: Sprite
+ drawingTools: GameObject
- drawingToolsButton: Button
- drawingToolsImage: Image
+ eraserSprite: Sprite
- topRight: Vector2
+ uIController: UIController
+ IsButtonClicked(Vector2): void
+ SetBrushImage(): void
+ SetColorPickerImage(): void
+ SetEraserImage(): void
+ Start(): void
+ Switch(): void
Set button image for
Set actual color using
ColorPalette
+ canvas: Canvas
- image: Image
- imageScaleX: float
- imageScaleY: float
- paletteHeight: float
- paletteWidth: float
- paletteXPos: float
- paletteYPos: float
+ uIController: UIController
+ Start(): void
+ Update(): void

Figure 3.7: GUI class diagram

The 3.8 diagram contains the classes responsible for the control system.

1. **Movement** the class is the central character controlls class. This class contains the movement and rotation logic of the character for both the PC version and the mobile version. This class interacts directly with joysticks (in the game version for Android OS).
2. Class **Settings** responsible for the current settings of the game controls.
3. **ControlsController** - This is a class that responds to keystrokes.

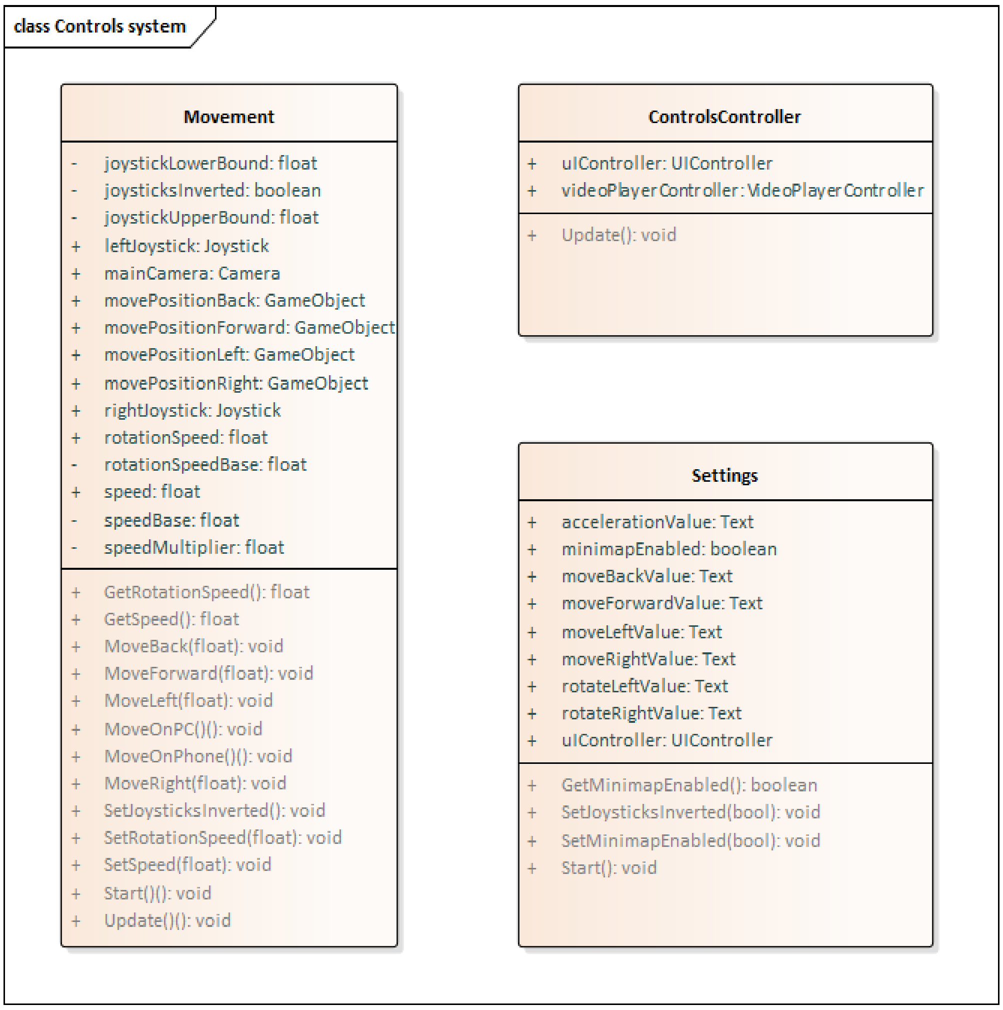

class Controls system
Movement
- joystickLowerBound: float
- joysticksInverted: boolean
- joystickUpperBound: float
+ leftJoystick: Joystick
+ mainCamera: Camera
+ movePositionBack: GameObject
+ movePositionForward: GameObject
+ movePositionLeft: GameObject
+ movePositionRight: GameObject
+ rightJoystick: Joystick
+ rotationSpeed: float
- rotationSpeedBase: float
+ speed: float
- speedBase: float
- speedMultiplier: float
+ GetRotationSpeed(): float
+ GetSpeed(): float
+ MoveBack(float): void
+ MoveForward(float): void
+ MoveLeft(float): void
+ MoveOnPC()(): void
+ MoveOnPhone()(): void
+ MoveRight(float): void
+ SetJoysticksInverted(): void
+ SetRotationSpeed(float): void
+ SetSpeed(float): void
+ Start()(): void
+ Update()(): void
ControlsController
+ uIController: UIController
+ videoPlayerController: VideoPlayerController
+ Update(): void
Settings
+ accelerationValue: Text
+ minimapEnabled: boolean
+ moveBackValue: Text
+ moveForwardValue: Text
+ moveLeftValue: Text
+ moveRightValue: Text
+ rotateLeftValue: Text
+ rotateRightValue: Text
+ uIController: UIController
+ GetMinimapEnabled(): boolean
+ SetJoysticksInverted(bool): void
+ SetMinimapEnabled(bool): void
+ Start(): void

Figure 3.8: Controls system class diagram

In the 3.9 diagram, class diagrams for components such as Video, Cutscenes, and Map, can be seen.

1. **TimelineController** - This is a cutscene controller. This class reacts to the player entering a specific cutscene area.
2. **VideoPlayerController** - A controller for playing video at the beginning of the game level.
3. **MapMoving** - A class that moves and rotates (in the case of a minimap) a map and the player on it.

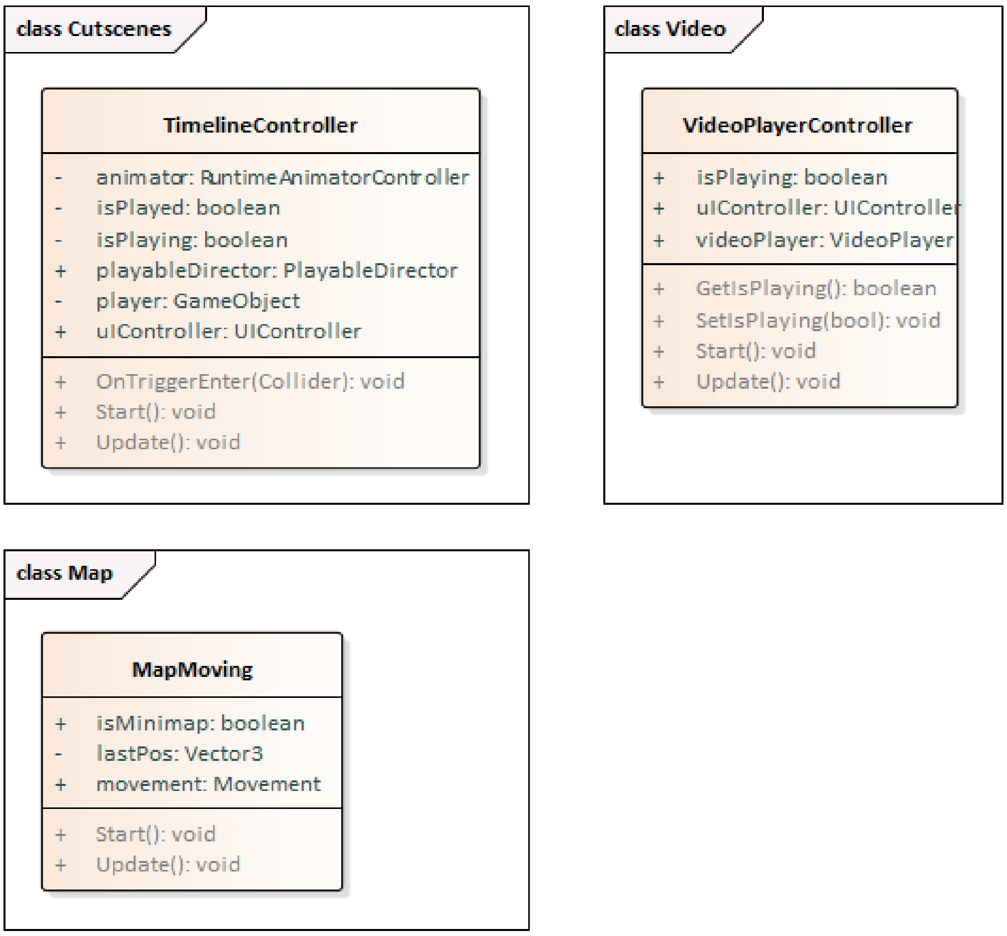


Figure 3.9: Cutscenes class diagram, Video class diagram, Map class diagram

3.9 Component diagram

Let us consider how the components interact with each other using the 3.10 diagram.

Almost all components interact with the GUI component, because the game constantly requires interaction with GUI elements. The Video and Cutscenes components access the GUI to disable/enable UI elements while playing the video or the cutscene. The Controls system component accesses the GUI for various checks. The drawing system component accesses the GUI to change the currently selected color on the screen. The GUI component itself interacts with components such as the Drawing system (reaction to actions in the drawing toolbar) and the Controls system (control settings changing). The Map component interacts with the Controls system component (since the movement of the camera depends on the movement of the player).

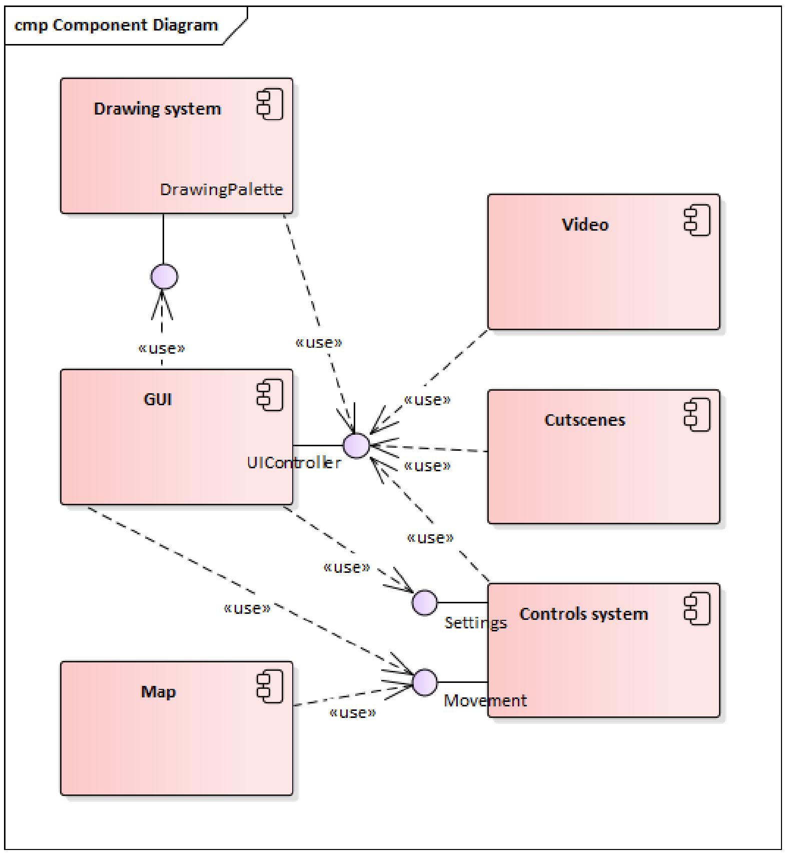


Figure 3.10: Component diagram

3.10 Scene objects

Let us take a look at some objects inside the scene. Figure B.1 shows a list of scene objects.

- **Environment** - this is the object that contains all world objects that are visible for the player (houses, traffic lights, cars, etc.).
- **Drawables** - this is an object that contains all the sprites that we can use for drawing.
- **MapCamera** - this is the camera that is used for the map implementation.
- **Canvas** - the canvas on which all other GUI elements of the game are located.
- **MinimapCamera** - this is the camera that is used for the minimap implementation.
- **VideoPlayer** - an object that contains a video player, as well as the controller for controlling video playback.
- **Timeline** - an object containing components for playing the cutscene.

3.11 Activity diagram

The game starts from the main menu, where it is possible to click on one of the three graphic buttons or the Escape (Back) button. If a player clicks on the game's start button, the video will be played, after which the game will start and the player will get to the game level. If the player presses the exit button, the game will be stopped. If the the player clicks on the settings button, a window with the settings will be opened. If the player presses the Escape (Back) button, the further depends on whether the player has started the game before (in other words, whether the game is paused). If so, the player goes to where they were before. Also, it is possible to go to the menu from any other screen by pressing the Escape (Back) button. The transition from the window of the game level is carried out in two ways: 1) pressing a certain key, 2) achieving a trigger to start a cutscene. If the transition was carried out by pressing a button, then it is possible to go back to the game level window by pressing the same button. In the case of a cutscene, the transition is automatic after the end of the cutscene.

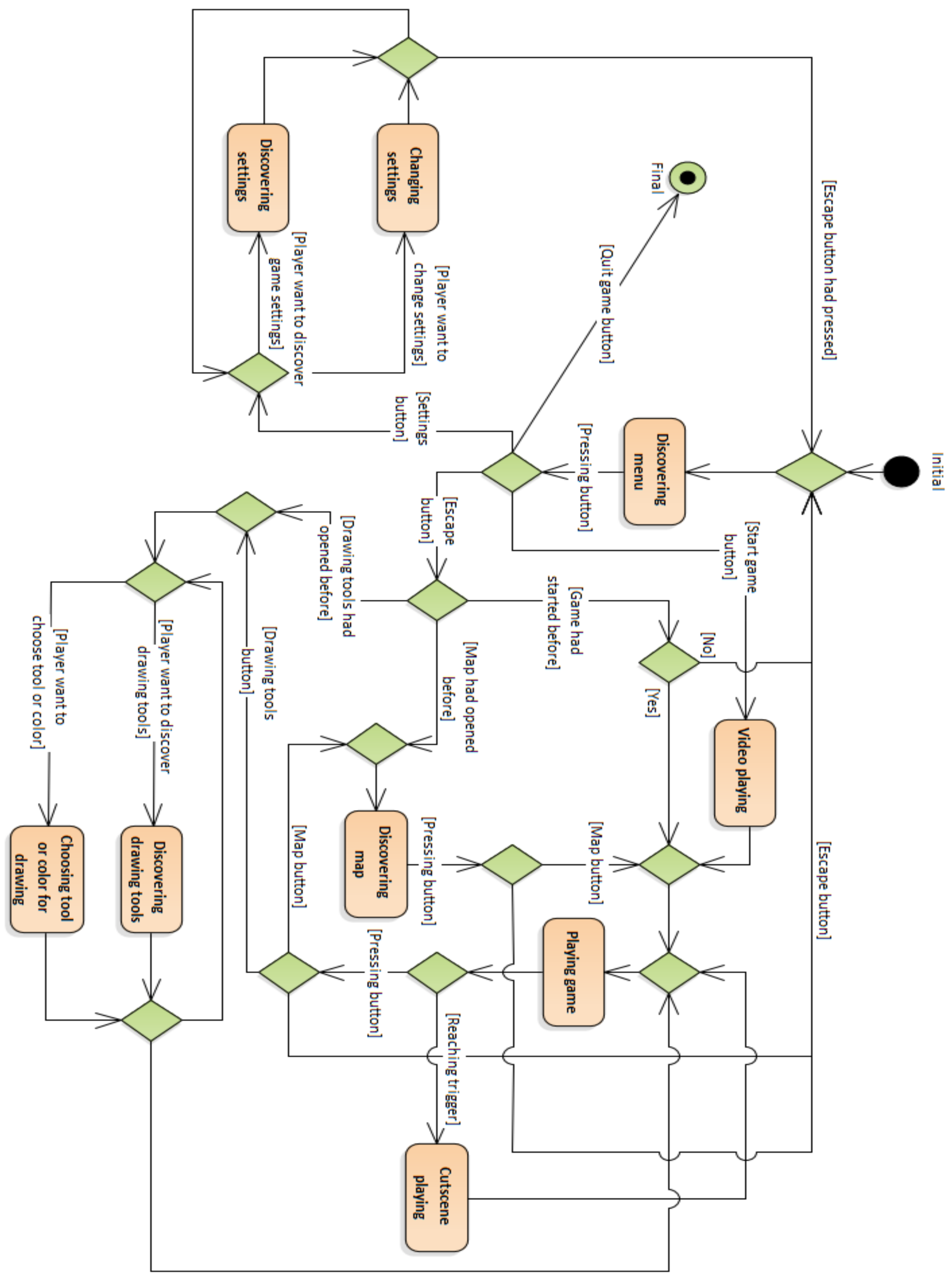


Figure 3.11: Game activity diagram

3.12 Conclusion

In this Chapter, the analysis of the game mechanics for the demo level was made. A list of requirements was compiled. The requirements were covered by the usecases. Next, the game demo level mechanics design was made. The game design document was created. Using this document, various diagrams, such as the class diagram, the components diagram, and the activity diagram were created and described. A GUI design was also built. All documentation created in this Chapter will be available in the folder containing this work. The results of this Chapter will be used for the further implementation of the demo level.

Chapter 4

Proof-of-concept implementation

This Chapter provides an implementation of the mechanics analyzed and designed in the previous Chapter. An implementation of the drawing system will be described. Within the framework of this system, the explanation of an implementation of brushes, as well as their import into the game, will be presented. An implementation of the basic mechanics of movement will be described. Also, there will be an implementation of the map. An implementation of taking colors from a color palette will be represented. An implementation of two cutscenes will be showed. The final part of this Chapter will be a conclusion summarizing the results of the implementation part.

4.1 Used assets

Before moving on to the implementation, it is necessary to list the assets that were used. All assets are taken from the Unity Asset store.

- Low Poly Cars[17]
- Cartoon Buildings[18]
- Epyphany textures and materials[19]
- Free stylized garden asset[20]
- HousePack[21]
- Free Hut Pack[22]
- Joystick Pack[23]

- Low poly playable vehicles[24]
- My 3D Town[25]
- Polygon desert pack[26]
- Simple City pack[27]
- Stylized vehicles pack[28]
- Tarbo - City 'Traffic Lights' Pack[29]
- The barn[30]
- Wires pack[31]

4.2 Drawing system

This section will discuss the implementation of the drawing system.

4.2.1 Brushes

Since the game brushes are simple and do not support resizing, it was decided to implement them using a bitmap. Each of the bit tells whether a given pixel is visible or not. The bitmap will be implemented using the two-dimensional array with the Boolean values. The set of the visible pixels forms the brush texture.

4.2.2 Brushes import

Brushes will be imported from files located in the file system (more specifically, in the game's resources folder, which is located in the folder with the game's assets).

Files with brushes will store information in the form of a string of bits and characters, for subsequent reading of this line and the conversion of the received values into a brush. The string will have the following format:
1) The first 16 bits of the string form a short number showing the size of the brush
2) The next 16 bits also form a short number and show the length of the brush name.
3) The next n characters, where n is the length of the brush name, form the name of the brush.
4) All further bits (the number of which should be equal to the size of the

brush squared) are bits for constructing the brush texture.
The construction of the brush texture from the bit string is carried out in the following way: the bit string is divided into the number of parts equal to the size of the brush (the length of each part is also equal to the size of the brush). Each of these parts is a line in a two-dimensional array of brush pixels. Therefore, we add values from the string to a two-dimensional array of pixels.

4.2.3 Drawing position detection

In order to be able to determine the position of a click on a texture, the following steps were taken:
1) The collider wass placed (since this mechanics will work in the 3D world, the collider must be 3D) on the sprite and resized so that it matched the size of the sprite.
2) An empty game object was placed in the lower left corner of the sprite (the coordinates of this object had to correspond with the coordinates of the lower left corner of the sprite in the game world). This object would be a kind of pivot (hereinafter we will call this object pivot).
3) Next, it was necessary to implement the OnMouseDrag method that was called while holding the left mouse button (or finger on the screen) in the collider area. Part of this method is shown in listing 4.1.
4) For the reason that the fact of holding the click on the collider did not carry any information about the position of the click, it was necessary to get the coordinates of the mouse on the game screen and create a ray that would shoot from the camera through a point having the received mouse coordinates.
5) Next, it was necessary to shoot this ray at the collider and get the object of contact of the ray with the collider (this object contains information about the point of contact).
6) The next step was to convert the coordinates of the point of contact into the 2D coordinate of the texture. Knowing the coordinates of the pivot, it was possible to calculate the coordinate of the click on the texture. The coordinates were calculated using the image 4.1. The red dot (hereinafter - R) is the pivot, blue (hereinafter - B) is the click point on the sprite. The distance between the two-dimensional points R (x, z) and B (x, z), which is indicated by the pink line, is the X coordinate of the click. To determine the Y coordinate, it is enough to subtract the Y value of the pivot from the Y value of the click point. This value is indicated by the orange line.
7) Since the sprite object itself can have a scale different from the standard, it was necessary to bring the obtained coordinates to a normal scale, thereby

obtaining the coordinates of the click on the texture.
8) The last step was to transfer the coordinates from the coordinates of the game world to pixel coordinates.

```
Vector3 clickPos = -Vector3.one;
// raycasting mouse click position
Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
RaycastHit hitData;
// if we've shot some collider
if (Physics.Raycast(ray, out hitData, Mathf.Infinity))
{
    // if it's not collider with layer Drawing (id = 8) or
       it's not trigger
    if (hitData.collider.gameObject.layer != 8 || !hitData.
       collider.isTrigger)
    {
        return;
    }
    // calculating coords of the pixel on which the player
       had clicked
    // calculating distance from left bottom point to the
       point where the player clicked (we're using x and z
       axes)
    clickPos.x = GetDistance(new Vector2(hitData.point.x,
       hitData.point.z), new Vector2(pivot.transform.
       position.x, pivot.transform.position.z));
    // calculating distance between left bottom point y
       axis and click position point y axis
    clickPos.y = Mathf.Abs(hitData.point.y - pivot.
       transform.position.y);
    // applying scale and translating it into pixels
    int x = (int)((clickPos.x / scaleX) * ppu);
    int y = (int)((clickPos.y / scaleY) * ppu);
    if (drawingPalette.colorPickerEnabled)
    {
        // if clicked pixel alpha is not zero
        if (renderer.sprite.texture.GetPixel(x, y).a >
           0.001)
            uIController.SetColor(renderer.sprite.texture.
               GetPixel(x, y));
    } else
    {
        DrawOnTexture(x, y, renderer.sprite.texture);
    }
}
```

Listing 4.1: Click position detection code listing

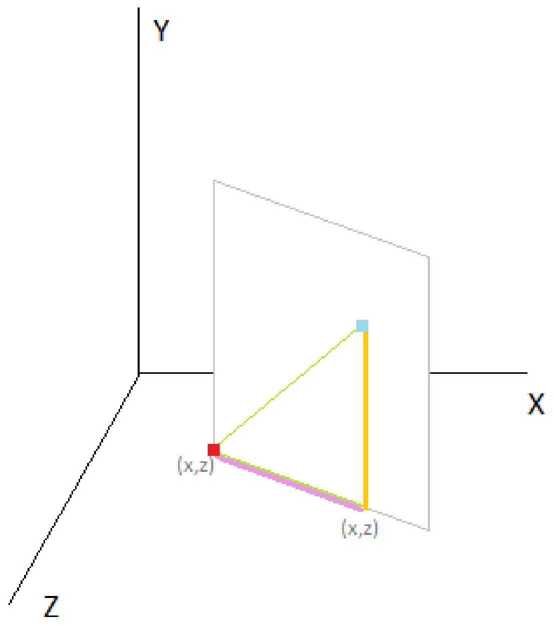


Figure 4.1: World coordinates system

4.2.4 Drawing/Erasing/Color picking

Paint/erase operations are implemented by setting the color/transparency of pixels that implement the brush texture at the player's click point.

The color picking operation is implemented by taking the color of the pixel that the player clicked on.

4.3 Map/minimap

To implement the map and minimap, we need two additional cameras (for the map and minimap) that will present a view of the game level from above. We have to set up the projection of cameras to orthographic. The image from the minimap camera will not be displayed on the screen, but to the texture. This texture will be displayed in the Image component of the minimap game object. Turning on (off) the map is a simple activation (deactivation) of the object with the camera.

4.4 Movement

To implement the movement, four empty objects were placed in front of, behind, to the left and to the right of the player. All objects are at the same distance from the player. To move the player, the player's position is changed to the position of the object located on the side to which the movement takes place. The player's speed is regulated by multiplying the x and z coordinates of the object to whose position the player is moving by a certain speed value.

The speeding up is implemented by multiplying the speed by some constant.

4.5 GUI

4.5.1 Joysticks

The implementation of joysticks was taken from the Unity Asset Store. [23]

4.5.2 Drawing palette

Based on information written in this section 3.6.4, the implementation process was divided into several steps:

1. The first thing to make was to calculate and preserve the scaling of the original image, relative to the size of the object (not considering the scale of the canvas).
2. Secondly, the real dimensions of the palette (the sizes of what we see on the screen) were calculated.
3. The next step was to find the coordinates of the lower left point of the palette.
4. The coordinates of the mouse click on the screen were obtained.
5. The coordinates of the lower left point of the palette were subtracted from the coordinates of the mouse click, thereby obtaining the coordinates of the click relative to the palette (in other words, we obtained the distances from the click point to the lower left point along the x and y axes).

6. The obtained coordinates were scaled back to the size of the original image, thus obtaining the coordinates of the pixel that the player clicked on.

The steps described above are shown in listing 4.2

```
// calculating image scale
imageScaleX = image.sprite.texture.width / image.
    rectTransform.rect.width;
imageScaleY = image.sprite.texture.height / image.
    rectTransform.rect.height;
// calculating actual visible image size
paletteWidth = image.rectTransform.rect.width * canvas.
    scaleFactor;
paletteHeight = image.rectTransform.rect.height * canvas.
    scaleFactor;
// calculating actual image position
paletteXPos = (image.transform.position.x - (paletteWidth /
     2));
paletteYPos = (image.transform.position.y - (paletteHeight
    / 2));
Vector2 clickPos = Input.mousePosition;
// cheching if the click position is inside the palette
if (clickPos.x > paletteXPos + paletteWidth || clickPos.x <
     paletteXPos || clickPos.y > paletteYPos + paletteHeight
     || clickPos.y < paletteYPos)
{
     return;
}
// getting click position point relative to image
clickPos.x -= paletteXPos;
clickPos.y -= paletteYPos;
// applying canvas scale on calculated positions
clickPos.x /= canvas.scaleFactor;
clickPos.y /= canvas.scaleFactor;
// applying image scale on calculated positions
clickPos.x *= imageScaleX;
clickPos.y *= imageScaleY;
int x = (int)clickPos.x;
int y = (int)clickPos.y;
// getting clicked point pixel color
Color color = image.sprite.texture.GetPixel(x, y);
uIController.SetColor(color);
```

Listing 4.2: Color palette code listing

4.6 Cutscene

The first cutscene is a pre-recorded video of the game world. This video was processed using EbSynth following the steps described in the section 2.3 of the EbSynth Chapter. An example frame from this video is shown in the image 4.2. The video itself can be viewed by running the game level. The implementation of the second cutscene consists of the trigger area, upon entering which the Play() method of the PlayableDirector component is called. After calling this method the cutscene starts playing.

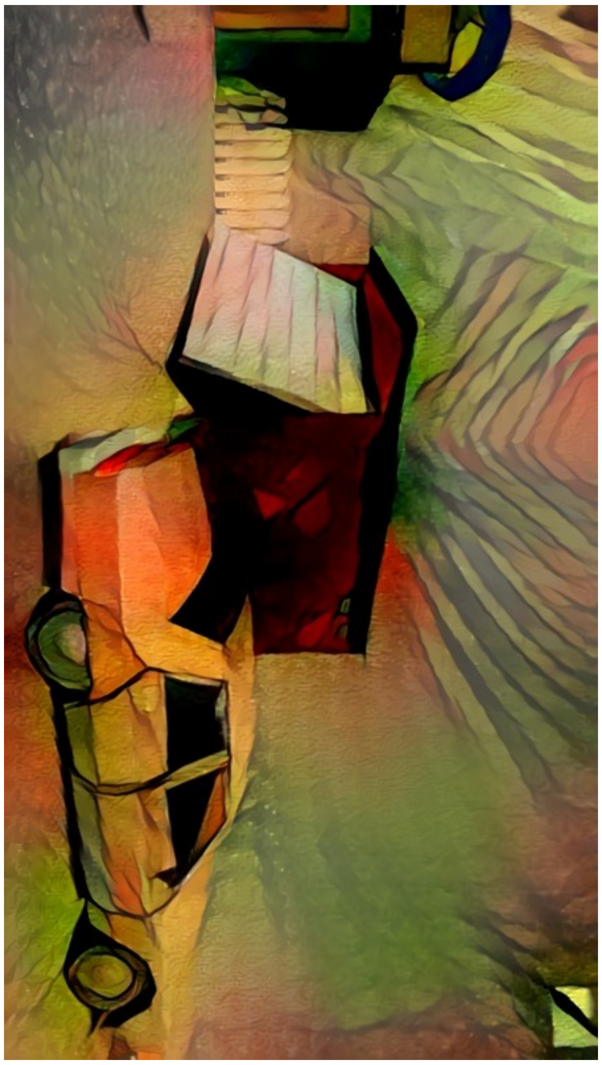


Figure 4.2: Synthesized video frame demonstration

4.7 Conclusion

In this Chapter, the proof-of-concept implementation has been described. All sources will be available on the GitHub. The results of this Chapter can be seen in the demo game level.

The results of this work can be applied in the art game called Sheepless or in any other game in a drawn style or having a theme of drawing.

The game implementation is fully compatible with the Android OS.

In future, the continuous drawing performance could be improved (for example, using multithreading). Another useful addition that will make the game developer's job easier would be a brush editor.

Conclusion

In this work, the possibilities of using EbSynth technology in creating a game on the Unity game engine were proposed. One of these features which is creating stylized cutscenes, was chosen and, subsequently, implemented.

The analysis and design of the primary game mechanics, such as movement, map, and playing cutscenes, were carried out. Also, the real-time drawing on the sprites mechanics was analyzed and designed. Problems that arose during the implementation of this mechanics were identified and solved. In addition to the painting mechanics itself, various painting tools such as the color picker, the eraser, and the color palette were also implemented.

A proof-of-concept game level was created with the implementation of all of the mentioned above. To demonstrate the EbSynth usage, a cutscene of the processed video was placed into the game level.

The implementation of this game is fully compatible with the Android OS.

This work could be useful for game developers having drawn stylistics or having a drawing theme. Sheepless could be one of these games.

All source codes made during this work will be available on GitHub. All documentation made while analyzing and designing the game mechanics will be available in this thesis folder.

One of the things that could improve the performance of the drawing system is the use of multithreading. A separate work might be devoted to the study of multithreading when implementing such a drawing system. Another improvement that might be implemented in future and will make it easier to make brushes can be the brush editor.

Bibliography

[1] Matt Peckham. Time. *See How Cuphead's Incredible Cartoon Graphics Are Made* [online] TIME USA, 2017 [visited on 2020-30-05]. [Cited 2020-29-05]. Available from: `https://time.com/4123150/cuphead-preview/`

[2] Omer Kaplan. Techcrunch - *Mobile gaming is a $68.5 billion global business, and investors are buying in* [online]. [Cited 2020-29-05]. Available from: `https://techcrunch.com/2019/08/22/mobile-gaming-mints-money/`

[3] Klicpera, Jan. Sheepless – *An Open-source 2D Adventure Game in Unity.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020. Supervisor: Marek Skotnica.

[4] Mustiats, Ian. *Sheepless – An Open-source 2D Adventure Game in Unity.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020. Supervisor: Marek Skotnica.

[5] Kevin Murphy. *Game sparks. Unity Game Engine Review* [online] Game Sparks Technologies Ltd 2019 [visited on 2020-30-05]. [Cited 2020-30-05]. Available from: `https://www.gamesparks.com/blog/unity-game-engine-review/`

[6] Unity Technologies. *Unity engine manual* [online]. [Cited 2020-10-04]. Available from: `https://docs.unity3d.com/Manual/`

[7] Romain Dillet. Techcrunch. *Unity CEO says half of all games are built on unity.* 2018 [online], Verizon Media, 2013-2020 [visited on 2020-30-05]. [Cited 2020-29-05]. Available from:

https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/

[8] Ashley Godbold, S. J. *Mastering Unity 2D Game Development. Second edition.* Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing Ltd, 2016, ISBN 978-1-78646-345-6.

[9] Unity Technologies. Unity engine manual. *Texture2D* [online]. [Cited 2020-30-05]. Available from: https://docs.unity3d.com/ScriptReference/Texture2D.html

[10] Unity Technologies. *Unity engine documentation* [online]. [Cited 2020-10-04]. Available from: https://docs.unity3d.com/ScriptReference/

[11] Calabrese, D. *Unity 2D game development.* Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing Ltd, 2014, ISBN 978-1-84969-256-4.

[12] SCRTWPNS. *EbSynth. Alpha version* [software] [visited on 2020-30-05]. [Cited 2020-30-05]. Available from: https://ebsynth.com/

[13] Jamriška, O.; Sochorová, v.; et al. *Stylizing Video by Example. ACM Trans. Graph.*, volume 38, no. 4, July 2019, ISSN 0730-0301, doi: 10.1145/3306346.3323006. Available from: https://doi.org/10.1145/3306346.3323006

[14] The Blender Foundation (2002). *Blender* [software] [visited on 2020-30-05]. [Cited 2020-30-05]. Available from: https://www.blender.org/

[15] GIMP. *Brushes.* [online] [visited on 2020-30-05]. [Cited 2020-30-05]. Available from: https://docs.gimp.org/2.10/en/gimp-concepts-brushes.html

[16] Common graphics. *Pixmaps in Common Graphics.* [online] [visited on 2020-30-05]. [Cited 2020-30-05]. Available from: https://franz.com/support/documentation/current/doc/cg/cg-pixmaps.htm

[17] Broken Vector. *Low Poly Cars* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-cars-101798

[18] IDALGAME. *Cartoon Buildings* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/urban/cartoon-buildings-161395

[19] Epyphany games. *24 PBR Materials for Unity 5* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/2d/textures-materials/24-pbr-materials-for-unity-5-51991

[20] Easy3D. *Free Stylized Garden Asset* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/props/exterior/free-stylized-garden-asset-145896

[21] Mehdi Rabiee. *House Pack* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/house-pack-35346

[22] Storm Bringer Studios. *Free Hut Pack* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/props/free-hut-pack-130776

[23] Fenerax Studios. *Joystick Pack* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631

[24] gameDev Mode. *Low Poly Playable Vehicles* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/vehicles/land/low-poly-playable-vehicles-154577

[25] Innovana Games. *My 3D Town* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/urban/my-3d-town-150535

[26] Runemark Studio. *POLYDesert* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/landscapes/polydesert-107196

[27] 255 pixel studios. *Simple City pack plain* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/urban/simple-city-pack-plain-100348

[28] ELRED. *Stylized Vehicles Pack - FREE* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/vehicles/land/stylized-vehicles-pack-free-150318

[29] Tarbo Studio. *Tarbo - City 'Traffic Lights' Pack [FREE]* [online]. [Cited 2020-27-05]. Available from: https://assetstore.unity.com/packages/3d/environments/urban/tarbo-city-traffic-lights-pack-free-154053

[30] AP3X_Models. *The Barns Free* [online]. [Cited 2020-27-05]. Available from: `https://assetstore.unity.com/packages/3d/environments/the-barns-free-155403`

[31] Mixaill. *Wire Fence* [online]. [Cited 2020-27-05]. Available from: `https://assetstore.unity.com/packages/3d/characters/wire-fence-67846`

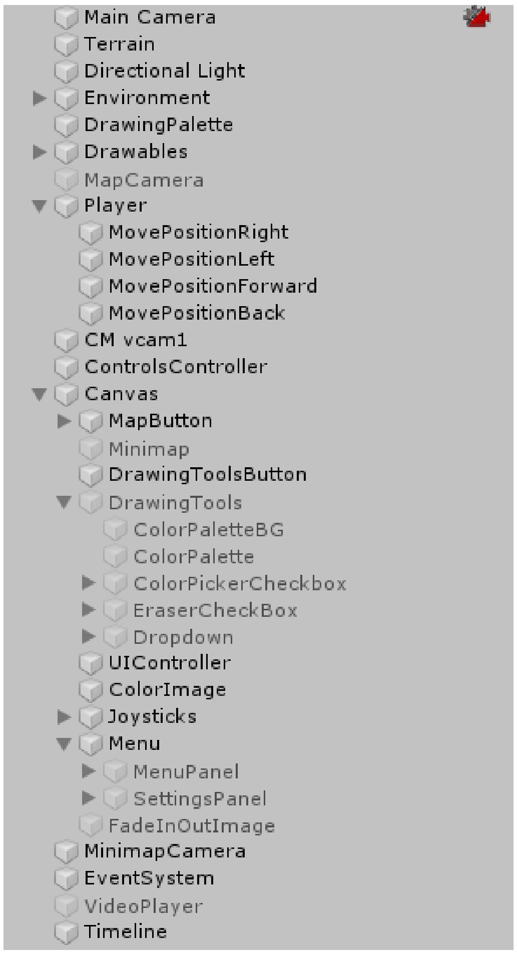
Appendix A

Acronyms

GDD	Game design document
GUI	Graphical user interface
OS	Operating system
UE	Unity Engine

Appendix B

Attachments



```
Main Camera
Terrain
Directional Light
▶ Environment
DrawingPalette
▶ Drawables
MapCamera
▼ Player
    MovePositionRight
    MovePositionLeft
    MovePositionForward
    MovePositionBack
CM vcam1
ControlsController
▼ Canvas
    ▶ MapButton
    Minimap
    DrawingToolsButton
    ▼ DrawingTools
        ColorPaletteBG
        ColorPalette
        ▶ ColorPickerCheckbox
        ▶ EraserCheckBox
        ▶ Dropdown
    UIController
    ColorImage
    ▶ Joysticks
    ▼ Menu
        ▶ MenuPanel
        ▶ SettingsPanel
    FadeInOutImage
MinimapCamera
EventSystem
VideoPlayer
Timeline
```

Figure B.1: Scene objects

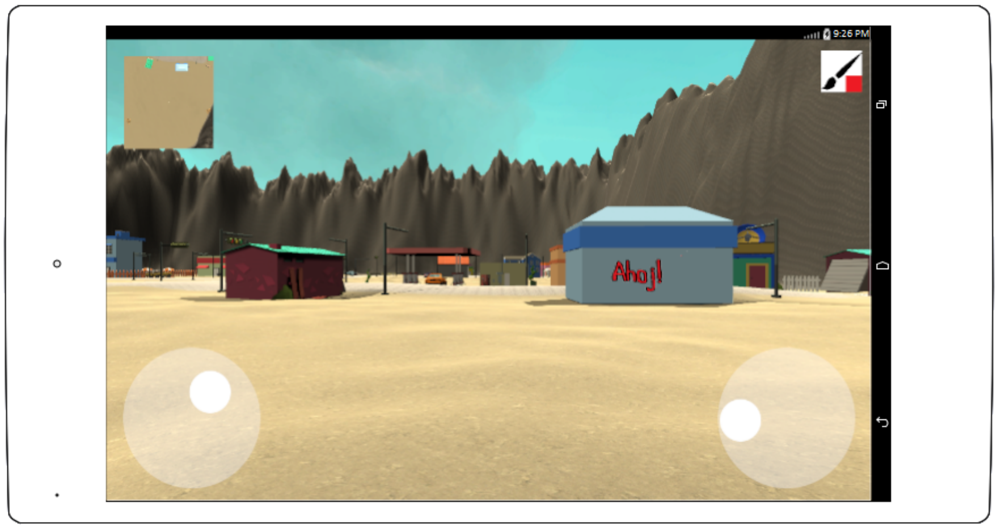


Figure B.2: Game view

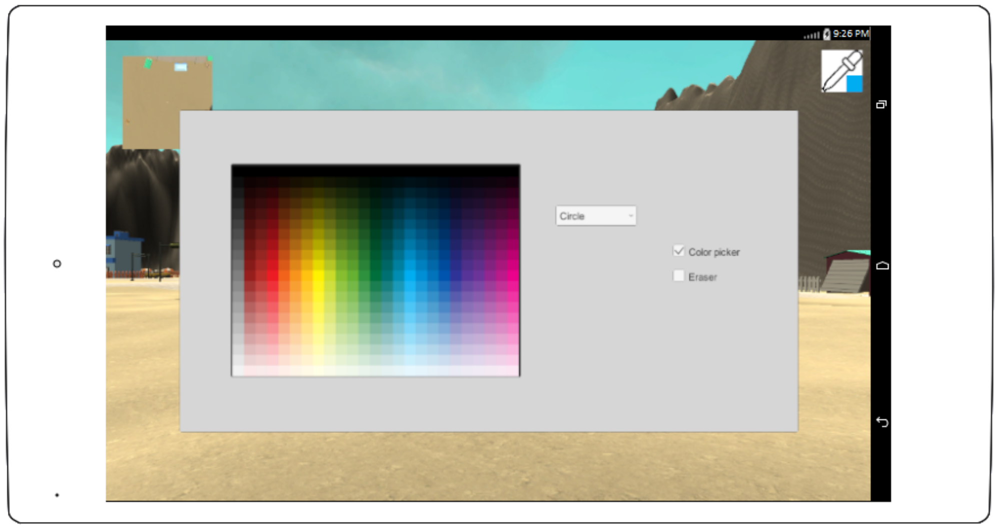


Figure B.3: Drawing tools window

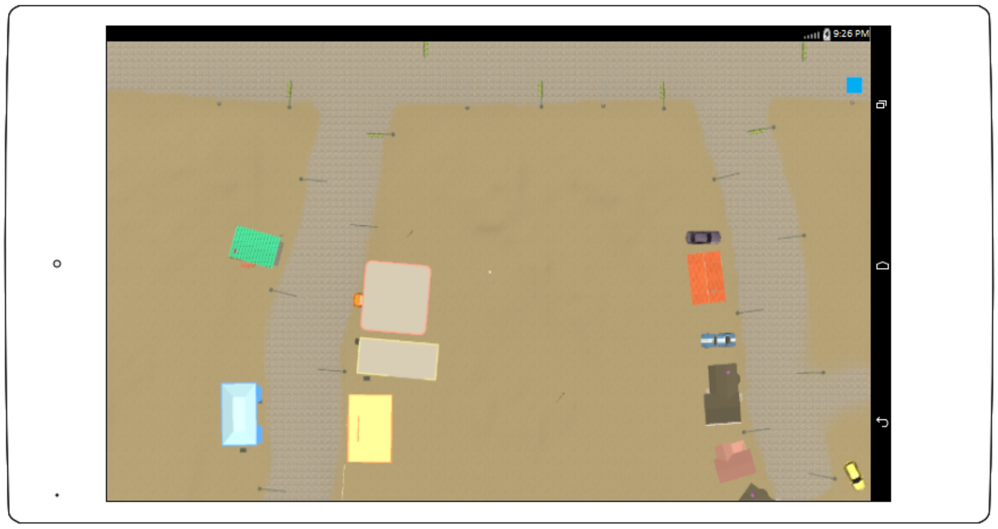


Figure B.4: Map view

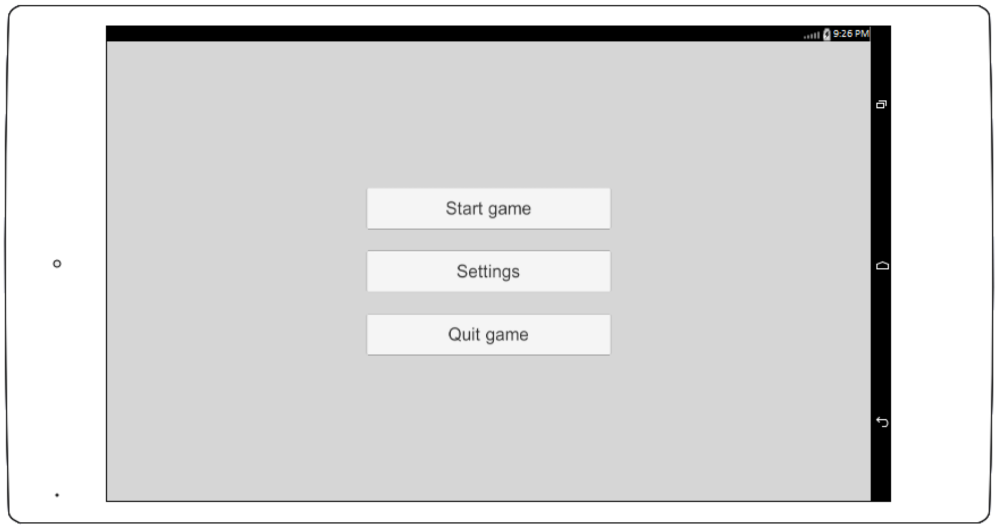
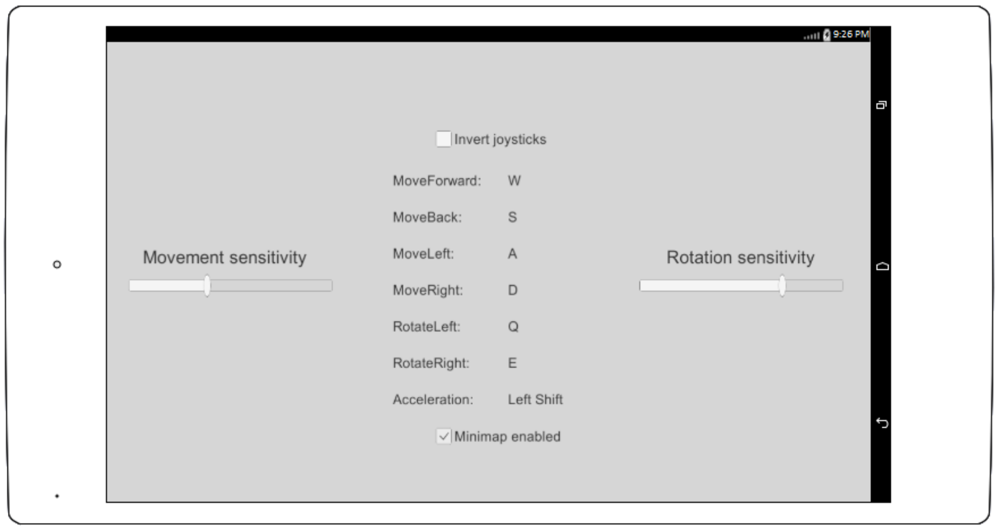


Figure B.5: Main menu



☐ Invert joysticks

MoveForward:	W
MoveBack:	S
MoveLeft:	A
MoveRight:	D
RotateLeft:	Q
RotateRight:	E
Acceleration:	Left Shift

✓ Minimap enabled

Figure B.6: Setting menu

Appendix C

Contents of enclosed USB disk

```
readme.txt ........................ the file with USB contents description
build ............................. the directory with the unity game builds
    android ....................... the directory with the android game version build
    PC ............................ the directory with the PC game version build
src ............................... the directory of source codes
    thesis ........................ the directory of LaTeX source codes of the thesis
    unity ......................... the directory with unity game sources
documentation ..................... the directory with the Enterprise Architect project
text .............................. the thesis text directory
    thesis.pdf .................... the thesis text in PDF format
```