Pipes in the Desert
Project, Requirements & Functionality

TEAM NR5 – main(String)

Supervisor:
Dr.Katalin Balla

Members:

Klevis Imeri	T4XGKO	klevisimeri11@gmail.com
Mean Diamand	KSG25Z	klevisimeri11@gmail.com
Mousavi Hossein	XQ9SE7	mousavi.hn@gmail.com
Taher Mured Abdulghani	B6RXU1	murad.abdalghani7@gmail.com
Ibrakhim Tolobekov	GQ1UJJ	ibrakhim1908@gmail.com

26/02/2024

2. Requirements, project, functionality

2.1 Introduction

2.1.1 Goal

The goal of the document is to provide a detailed description of the game application showing how the game works on different levels of descriptions, the requirements of how the application will be developed and processed, the planning, and use cases.

2.1.2 Application domain

A game application that can be used for entertainment and educational purposes which allows you to compete with the other players and learn more about resource management, network flows, and strategy.

2.1.3 References

https://www.iit.bme.hu/oktatas/tanszeki_targyak/BMEVIIIAB02
https://www.iit.bme.hu/targyak/BMEVIIIAB02/problem-definition
https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
https://code.visualstudio.com/docs/java/java-linting
https://maven.apache.org/plugins/maven-checkstyle-plugin/

2.1.4 Document content

- ★ Section 2 - Overview
  - General description of the Game application
- ★ Section 3 - Requirement
  - Specific functional that detailed the expected behaviors and non-functional requirements that detailed the technical specification of the application.
- ★ Section 4 - Essential use-cases
  - Use-case diagram with descriptions for each possible case that can occur in the game, showing the purpose of the application.
- ★ Section 5 - Glossary
  - explanation of the keywords used in the documentation which makes it more understandable.
- ★ Section 6 - Project Plan
  - Summary of the entire project with clear details on project structure, goal, timetable, workflow, and roles of each member.

2.2 Overview

2.2.1 General overview

The application consists of subsystems such as a
- Graphical User Interface(GUI)
- User interface (UI)
- Grid(pipe system):
    - Pipes
    - Active elements:
        - Spring: the water source of the game,
        - Cistern: water storage of the cities,
        - Pump: connectors for pumps
- Players

Pumps have multiple features concerning the connectivity and flow of the water.

Depending on the behavior of the subsystem we can specify interfaces such as Flowable, Movable, and Modifiable that act as a common base. E.g the flow of water throughout the pipes, and pumps can be called a Flowable interface, the movement of the actors like plumber and saboteur can be called Movable, also the modification of the system by the same actors can be called Modifiable.

The basis of communication is the User Interface where the user can interact with the game application with the action of the keypress, mouseclick, button-click,...etc. Graphical User Interface is also the basis of communication where the user can interact with the game using graphical elements such as windows, icons, and menus.

For data storage, this game application will store different types of data such as the amount of water they collected(score), and the names/information of the team players.

2.2.2 Functions

This game simulates a strategic battle between two teams in which the game consists of a complex pipe system that can be looked at similar to a network flow, where the system is responsible for transferring water from the source to the destination, the system is built of the following components such as pipes (the primary conduit for the water flow)and some active elements like pumps, spring, and cisterns which each of them has specific limited functionalities, such as the Spring acts as the source of the water and the cistern acts as the destination, In addition to that multiple pipes can be connected to a pump. However, each pump has a specific number of pipes that can be connected to it. Each pump can have only a single input and output while other connected pipes to the pump are closed(do not transfer water). At random times a pump may stop working which means that water will not flow out of the pump, each pump contains a water tank which will act as a temporary reservoir when transferring water from the incoming pipe to the outgoing pipe, in addition to that pumps will only be able to transfer water if the outgoing pipe had some free capacity.

One of the features of the pipe system is the ability to be extended and modified, the pipe component will have the ability to disconnect from the active element that is located at either end and the ability to connect it to another active component, additionally, these pipes, as well as the pumps, will be regularly manufactured at the cisterns (I will finish this part at a later time because of uncertainty), however, the pipes in the system have a certain capacity which means that the flow of the water throughout the pipes can not exceed the capacity.

Two other Main Players play a crucial role in the system behavior which are the plumbers and the saboteurs where the role of the plumbers is to maintain the system as well as fix the broken pumps, set the direction of the pumps, fix the leaking pipes which in the case of leaking the water will not be able to reach the other end of the pipe (spills out), in other words, the plumbers try to transfer as much water as possible (from spring to destination), they can extend the system with pipes and pumps which both are manufactured at the cisterns in effort to transfer as much water as possible, additionally, plumbers will be able to add a pump in the middle of a pipe which is only possible but cutting the pipe and connecting the free ends to pump.

Moreover, the role of saboteurs is to counter the plumbers aiming to reduce the water transfer, creating as much water leak as possible from the pipe system, that goal can be reached with the abilities saboteurs will have which mainly consist of the ability to change the direction of the pumps and the ability to puncture the pipes which results in the leaking of the pipes, hence water spills out.

The whole pipe system will exist on a dessert, which is not considered to be a safe place and although these two players can manipulate the system and the flow of the water through it, there are some limitations to their abilities which mainly involve the following: Movement limitation which means players will only be able to move on pipes and pumps. More specifically, only one player can go along a pipe at a time. However, players will be able to go around each other on the pump, in other words, multiple players will be able to be at a pump at the same time.

The game is played by two teams who are as mentioned the plumbers and the saboteurs, both have their tasks with respect to their abilities, and when the game comes to an end, the winner is decided depending on the amount of water the team has collected.

2.2.3 Users

This game application is suitable for different ages like adults or kids who are interested in the strategy and teamwork game. It will be a two-team game consisting of at least two players on each team

- Plumber Team:
    - can set the directions of the pumps.
    - can fix the leaking pipes.
    - can manufacture the pipes.
    - can pick up new pumps.
- Saboteur Team
    - can change the directions of the pumps.
    - can puncture the pipes.

The players can pick which team they want to play as and add the plumbers and saboteurs as much as they want but it has to be at least two players in each team.

2.3 Requirements

2.3.1 Functional requirements

ID	Description	Check	Priority	Source	Use-case	Comment
FR1.1	Allow plumbers to connect several pipes to a pump such that the limit number of pipes the pump can have is not exceeded.	We are going to demonstrate that plumbers can connect several pipes to pumps without exceeding the limit of the pump by using the GUI interface. The grid will have several pipes and the plumber will connect several pipes to the pump. We check that we can not connect more pipes than the limit.	Must have	Problem Description Paragraph [1,3]	Connect Pipe	-
FR1.2	Enable plumbers to select only one ingoing and one outgoing pipe for each pump, while the other pumps are closed.	We are going to demonstrate that plumbers can select only one ingoing and one outgoing pipe for each pump by using the GUI interface. The pump will have several pipes and the plumber will select one ingoing and outgoing pipe. We will check visually the water flow and check that the water is only going in these pipes and that the other pipes are closed.	Must Have	Problem Description Paragraph [2,3]	Change In and Out Pipe	-
FR1.3	Pumps can go out of order randomly, halting water flow through them	This feature is a randomly activated attribute of the system that must be checked by the GUI interface. The pumps should turn randomly out of order, as indicated by them. becoming red on the	Must have	Problem Description Paragraph [1]	Pump Breaks	-

		screen. The water should visibly stop flowing through the pump.				
FR1.4	Pumps should transfer water from the incoming pipe to the outgoing pipe, utilizing a temporary water tank as a reservoir, and water transfer by pumps is only possible if the outgoing pipe has available capacity.	This feature must be checked through the GUI by connecting a lower-capacity outgoing pipe than the input pipe to the pump. We check visually that the water flowing through the outgoing pipe does not exceed the pipe capacity and second, the leftover water will be reserved in a reservoir tank connected to the pump.	Must have	Problem Description Paragraph [1]	Change In and Out Pipe	-
FR1.5	Pipes should have a capacity which allows specific amounts of water in.	This feature must be checked through the GUI by having an already established grid and measuring that the pipe capacity (optionally displayed) is not smaller than the amount of water flowing through the pipe.	Must have	Indirect requirement Problem Description Paragraph [1]	Change In and Out Pipe	-
FR2.1	Allow plumbers to connect pipes to active elements (spring, cistern, solar water pump).	We are going to demonstrate that plumbers can connect pipes to active elements by using the GUI interface. The grid will have several pipes and the plumber Must have Problem Description Paragraph [2,3] will connect	Must have	Problem Description Paragraph [2,3]	Connect Pipe	-

		them to the active elements. We connect a pipe for each type of active element so we can check that we can connect a pipe to each.				
FR2.2	Cisterns can produce new pipes regularly. The new pipes should be connected from one end to the cistern while the other end is free (not connected to any active element).	We are going to demonstrate that the cistern regularly produces new pipes by demonstrating visually, using the GUI, that new pipes will show up at the cistern. These new pipes should have one end at the cistern and the other end should be free.	Must have	Problem Description Paragraph [2]	Produce Pump/ Pipe	-
FR2.3	If a pipe has a free end the water transferred into the pipe should flow out to the desert.	This will be tested by demonstrating in the GUI. We will set the end of one pipe to be free and the other end to be connected to an active element which water is flowing into. The direction of the pipe should be toward the free end. Then we make this pipe the outgoing pipe. The water flowing through this pipe should flow in the desert, in other words, the water through the pipe should not flow towards the cistern (back in the grid, but outside it). This is checked by checking the water flow visually.	Must have	Problem Description Paragraph [2]	Produce Pump/ Pipe	-

FR3.1	Plumbers should be able to fix broken pumps.	We are going to demonstrate that the plumber can fix broken pumps by using the GUI interface. The grid is going to have broken pumps (red colored) where water is not flowing through them. The plumber will go to the pump and fix it (turn the pump blue). The water visually should start flowing through the pump.	Must have	Problem Description Paragraph [3]	Fix Broken Pump	-
FR3.2	Plumbers should be able to fix damaged (can be leaking or there can be a crack but because there is no water flowing through the pipe there is no leak) pipes.	We are going to demonstrate that a plumber can fix damaged pipes by using the GUI interface. The grid is going to have damaged pipes (pipes with a crack in them) where water is not flowing through them. The plumber will go to the leaking pipe and fix it (the crack must disappear). The water visually should start flowing through the fixed pipe if the pipe was leaking before being fixed.	Must have	Problem Description Paragraph [3]	Repair Leaking Pipe	-
FR3.3	The plumber should be able to set the direction of the pump.	This will be checked using the GUI interface. The plumber will change the direction of the pipe where water doesn't flow through. The change in direction will be shown in the pipe by the change of the arrow (or another visual element that corresponds to the	Must have	Problem Description Paragraph [3]	Change Pump Direction	*-p*

		direction of the pipe). Then the plumber will use the pipe or water flow through the other direction. The structure of the grid should be set accordingly for this demonstration so that when the pipes change direction it can be set to flow water through this new direction. So the pipe will be tested to allow the flow in the new direction. This change will be shown visually by the water flow.				
FR3.4	The leaking pipes should not transfer water. The water should spill out.	This will be checked with the GUI interface. There will be a leaking pipe. We will flow water through it. The water should not reach the end but it should spill in the desert. This can visually be checked by the water flow.	Must have	Problem Description Paragraph [3]	Puncture Pipe	-
FR3.5	The cistern should regularly produce new pumps.	This will be checked using the GUI interface. New pumps should show up (visually) regularly at the cistern. They can be plates in the grid by the plumbers.	Must have	Problem Description Paragraph [3]	Produce Pump/ Pipe	-
FR3.6	The plumber can insert new pumps manufactured at cisterns into the middle of pipes by cutting and connecting	This will be checked using the GUI interface. There will be an already laid-out grid with pipes. The plumber using his player interface will insert a pump in the middle of the pipe that can be selected from his position in the grid. The new pump	Must have	Problem Description Paragraph [3]	Add Pump	-

	the free ends	will be visually inserted between the two pipes.				
FR4.1	Saboteurs can change pump directions.	This will be checked using the GUI interface. The saboteurs will change the direction of the pipe where water flows through. The change in direction will be shown in the pipe by the change of the arrow (or another visual element that corresponds to the direction of the pipe). Because the direction has changed the water should not flow anymore through the pipe. So the pipe will be tested for stopping the flow in the old direction. This change will be shown visually by the water flow.	Must have	Problem Description Paragraph [4]	Change Pump Direction	-
FR4.2	Saboteurs can puncture pipes.	This will be checked using the GUI interface. There will be an existing grid with unpunctured pipes. The saboteur player using his interface will puncture a pipe he can reach from his current position in the grid. The water should stop flowing through the punctured pipe if it was flowing in the first place.	Must have	Problem Description Paragraph [4]	Puncture Pipe	-
FR5.2	The players (plumbers and saboteurs) can only move on the pipes and	It will be checked using GUI. We already have a grid. We start moving the players and just check if the movement is only allowed from one	Must have	Problem Description Paragraph [5]	Move	Because the program will let each player take turns one by one we only

	the pumps. People can go around each other at the pumps, but they cannot go around each other on pipes: only a single player can go along a pipe and many players can stay in a pump.	pump to another pump. In other words, no player stops at a pipe. We move several players to one pump. We also visually check that several players can fit (go around) in one pump. Then we check if all the players in this pump can move out of it.				have to check that each player goes from a pump to another pump in one go.
FR6.1	There must be two teams Plumbers and Saboteurs.	We can easily be checked by the prototype allowing the formation of two teams.	Must have	Problem Description Paragraph [6]	Select Teams	-
FR6.2	You can add players to teams.	This will be checked by the prototype allowing the addition of new players to each team.	Must have	Indirect requirement from Problem Description Paragraph [6]	Select Teams	-
FR6.3	Each team must have at least two players.	We can test that by demonstrating with the prototype that we cannot continue the game until each team has at least two players.	Must have	Problem Description Paragraph [6]	Select Teams	-
FR6.4	The system must be able to measure the water that flows from the spring through the grid to the cistern. It	This will be checked using the testing and GUI with several grids. We will have the spring which will bring x amount of flow. The first thing we are checking is when we just have a pipe connecting the	Must have	Indirect requirement from Problem Description Paragraph [6]	Change In and out Pipe	-

	also measures the amount of water that flows out of the system.	spring to the cistern is the water added up to the total water following how much flow is flowing. If the spring produces x water per play in this simple grid then x water is added to the cistern. Then we will form more complicated grids. We will check using GUI through each pump if the water flow is in accordance with the water flowing out and with the capacities of the pipes. Then we will measure the amount of flow reaching the cistern. Using a predefined grid we know how much flow should reach the cistern if the spring produces x flow. This can be easily tested for several grids and several flows. The water flowing out of the system can easily be calculated if you have the flow rate of the spring and the volume of water accumulated at the cistern.				
FRI.1	Each pipe may have the same capacity	This can be tested using the GUI interface. Each pipe should have the same capacity. And the newly created pipes should also have the same capacity. This can be checked visually by how much	Could have	Group Meeting [Assumptions]	Produce Pump/ Pipe	-

		water can flow through the pipe.				
FRI.2	The game should last for a predefined time.	We can check using the GUI. After the timer reaches the predefined time the game should stop and the results should be shown on the screen.	Must have	Group Meeting [Implementation Ideas]	Change Settings	-
FRI.3	When the game is started the user should be met with a window where he can press the button to start a new game or he can press the settings button to change the settings.	This can be checked using the prototype. The prototype after starting should show the window with the button to start a new game or the button to change the setting.	Could have	Group Meeting [Implementation Ideas]	Start Game and Change Settings	-
FRI.4	The user can start a new game pressing the button corresponding to it. After this the game application should ask for the teams.	This can be tested by the prototype such that when the program is opened the user can press the new game button and the program should start a new game where it will ask for the users.	Should have	Group Meeting [Implementation Ideas]	Start Game	-
FRI.5	The users should be able to stop the current game.	This can be tested by the prototype such that when the players are playing the game they can choose the button to end the game.	Could have	Group Meeting [Implementation Ideas]	End Game	-
FRI.6	The spring has infinite amount of water	This can be tested by the prototype program where the grid never runs out of the incoming water from	Must have	Group Meeting [Assumptions]	Connect Pipe	-

		the spring while playing the game.				
FRI.7	The cisterns and the desert can hold an infinite(not completely) amount of water. If the accumulated water is more than the precision of the numbers to measure its volume then the game should end.	This can be tested by the prototype program where the cistern and the desert never get filled completely with water from the spring while playing the game.	Must have	Group Meeting [Assumptions]	Connect Pipe	-
FRI.8	The player can only apply changes to pipes which connect to the active element he resides upon.	This can be tested using the prototype. There will be a grid and a player will reside in the active elements. He will perform changes to the pipes which connected to this active element; those pipes should change appearance according to the state they are set. Then the player will try to change pipes that do not connect to the active elements he resides upon. In this case, nothing should happen.	Must have	Group Meeting [Implementation Ideas]	Connect Pipe	-
FRI.9	Players will take turns one by one. The next player to play will be decided randomly.	This can be tested by doing a demonstration. The players should get their turn to play randomly during the game.	Must have	Group Meeting [Implementation Ideas]	Move	-

FRI.10	In one play the player can either move from one active element to another or change the state of elements he is close to in appliance with FRI.8. If he can not perform any move then his turn will skip automatically.	This can be tested by doing a demonstration. The players should be able to move from one active element to the other or apply changes to the pipes accordingly during the game.	Must have	Group Meeting [Implementation Ideas]	Move	-
FRI.11	If it is the player's turn which can not perform any move then his turn should be skipped.	This can be tested by doing a demonstration. The grid is set up such that there is a player that has no moves but it is his turn. The program should skip this player's turn.	Could have	Group Meeting [Implementation Ideas]	Move	-
FRI.12	When a player has his turn. He only has a specific amount of time to play, otherwise his turn will pass.	This can be tested by doing a demonstration. If the player turns and he does not make a move till his timer runs out. The game should skip his turn.	Must have	Group Meeting [Implementation Ideas]	Move	-

2.3.2 Resource requirements

ID	Description	Check	Priority	Source	Comment
RR0	The game must compile and run on the virtual machine found on the smallville cloud (https://smallville.cloud.bme.hu/). The machines have Windows 10 with Java 11.	This can be checked by creating a virtual machine and compiling and running the program in the Windows 10 with Java 11 virtual machine.	Must have	Course Webpage	Depending on how good these machines are the program may run a bit slower.
RR1	The game should be supported and run smoothly for the Windows 11 operating system with at least 16GB RAM, SSD storage, COREi7 8th Gen Intel® CPU and GEFORCE® GTX 1050ti graphics card.	The check is done by running the program on a Windows 11 machine with the described specs.	Must have	Group Meeting.	Here the program should run flawlessly.
RR2	The developers will collaborate using GitHub with the git version control system.	The app is on GitHub and the developers are committing to it.	Should have	Group Meeting	-
RR2	The java version used should be 11 or higher.	This can be checked through the maven compilation and the pom.xml file.	Must have	Group Meeting	
RR3	The environment used by the team will be Visual Studio Code installed with the Extension Pack for Java or any other IDE which supports development with maven.The developing	This is checked by analyzing IDE specific files. For example .vscode folder for Visual Studio Code.	Should have	Group Meeting	-
RR4	Typical project structure of a maven project will be used.	This can be checked by analyzing the project structure and realizing it is a structure of a	Must have	Group Meeting	-

		typical maven project with the pom.xml file.			
RR5	JUnit will be used for testing.	This can be checked by analyzing the pom.xml file.	Must have	Group Meeting	-
RR6	The Google Java Style will be used.	This can be checked using the maven-checkstyle-plugin by applying the google_checks.xml and the fail on violation set to true.	Must have	Group Meeting	-

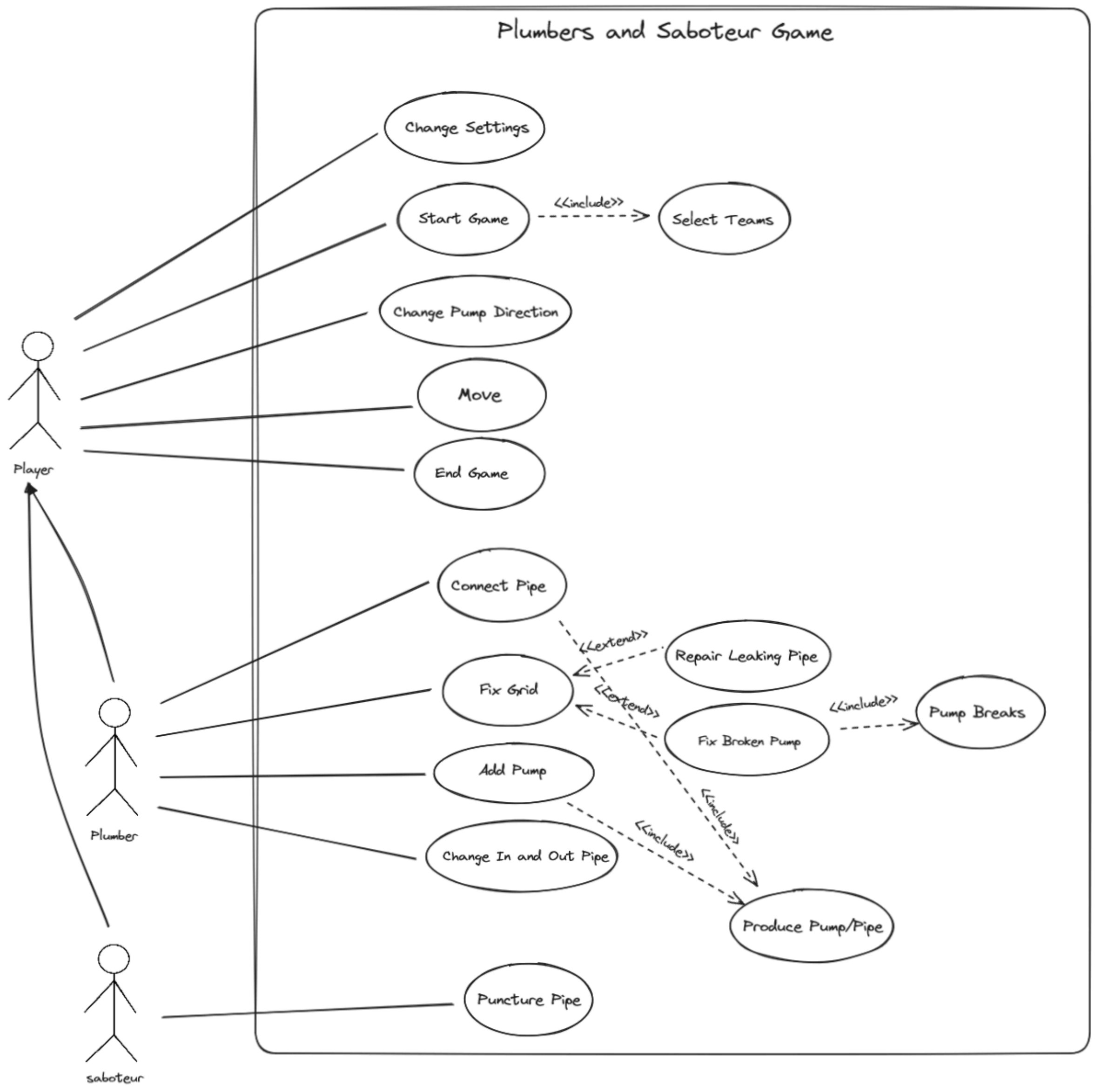
2.3.3 Non- functional requirements, Restrictions

ID	Description	Check	Priority	Source	Comment
NFR1	The application must be very easy testable visually using the GUI. GUI should have many options to display information of the system that are hidden for normal gameplay in an easy to understand way.	The application should be run and different information about the system should be displayed. It can easily be checked if the information is laid out in an easy to understand format by the time it takes to find the information.	Must have	Group Meeting	-
NRF2	The application should have a high level of modularity to make it easily testable.	This can be checked by loading new players easily. Loading new grids easily. Changing GUI for specific objects easily.	Must have	Group Meeting	-
NFR3	The system should be able to accommodate the addition of new pipes and pumps without significant performance degradation.	This feature must be checked by multiple additional attempts and checking the overall game response visually.	Must have	Group Meeting	It's hard to give specific numbers because it's hard to know how capable the devices are used to run the program are.
NRF4	The GUI and UI for the user should be easily usable and easy to understand. It should be very learnable.	This can be checked with a prototype by analyzing how a group of people play the game for the first time.	Should have	Group Meeting	-
NFR5	The application should be scalable to include more types of players or different kinds of active and passive elements.	This can be checked by analyzing the code for the application of OOP design patterns with interfaces and base classes. This can be checked by analyzing the number of times the code repeats or how easy it	Must have	Group Meeting	-

		is to add new functionality in the system.			
NFR6	The software should be easy to maintain.	This can be checked by analyzing the quality of the source code. Is the code easy to understand? Are the Coding guidelines enforced?	Should have	Group Meeting	-
NRF7	The software should be able to run in the span of 40 minutes without significant performance degradation.	This can be checked by playing the game for one or several games in 40 minutes.	Must Have	Group Meeting	-
NRF8	The software should be portable.	This can be checked and enforced easily because we are using maven and github. Maven is platform independent and makes compiling and running the code platform portable, while github makes it portable for downloading it from everywhere.	Should have	Group Meeting	You only have to use "mvn compile" or "mvn exec:java" in every platform after you have the code downloaded from git.
FN9	The software should reliably calculate the water flow through the grid to the cisterns.	This is checked together with the check that it can flow water from the spring to the cistern in FR6.4	Must have	Group Meeting	-
FN10	The software should be fair. It should randomly choose who plays the next move.	This can be checked by playing the game and measuring that the distribution of plays is equal in both the teams.	Must have	Group Meeting	-

2.4 Essential use-cases

2.4.1 Use-case diagram



2.4.2 Use-case descriptions

Use-case name	Connect Pipe
Short textual description	This use case involves the actions taken by plumbers to connect a pipe to an active element if it has a free end or disconnect from an active element and connect it to another one.
Actors	Plumber
Dialog, scenario	1. The actor identifies a pipe that he can connect from his position. 2. the actor proceeds to connect the pipe to the active elements he can reach from his position. 3. The pipe is connected to the new active element.

Use-case name	Fix Grid
Short textual description	This use case involves the actions taken by plumbers to maintain the integrity and functionality of the pipe system, including fixing broken pumps, and repairing leaking pipes.
Actors	Plumber
Dialog, scenario	1. The actor identifies a broken pipe/pump in the system that he can fix from his position. 2. the actor proceeds to fix the broken pipe/pump 3. The pipe/pump is repaired.

Use-case name	Fix Broken Pump
Short textual description	This use case involves the actions taken by plumbers to fix the broken pump.
Actors	Plumber
Dialog, scenario	1. The actor identifies a broken pump in the system that he can fix from his position. 2. the actor proceeds to fix the broken pump 3. The pump is repaired.

Use-case name	Repair Leaking Pipe
Short textual description	This use case involves the actions taken by plumbers to maintain the integrity and functionality of the pipe system by repairing leaking pipes.
Actors	Plumber
Dialog, scenario	1. The actor identifies a leaking pipe in the system that he can fix from his position. 2. the actor proceeds to fix the leaking pipe 3. The pipe is repaired.

Use-case name	Change Pump Direction
Short textual description	This use case involves the actions taken by both plumbers and saboteurs to change the direction of pipes in the pipe system, thereby controlling the flow of water.
Actors	Player
Dialog, scenario	1. The actor identifies a pipe whose direction is to be changed. 2. The actor selects the pipe and specifies the new direction for water flow.

Use-case name	Add Pump
Short textual description	This use case involves the actions taken by plumbers to add a new pump to the middle of a pipe by cutting the pipe
Actors	Plumber
Dialog, scenario	1. The actor identifies a pipe which is needed to be integrated with a new pump from his position. 2. The actor adds the pump to the proper section of the pipe.

Use-case name	Change in and out pipe
Short textual description	This use case involves the actions taken by plumbers to change the direction of ingoing and outgoing pipe for a specific pump.
Actors	Plumber
Dialog, scenario	1. The actor identifies the need to transfer water from an ingoing pipe to an outgoing pipe in the pump located at his position. 2. The actor checks that the outgoing pipe connected to the selected pump has sufficient capacity to receive water. 3. The actor chooses the pipes for int and out flow. 4. The other pumps are closed. 5. The water starts flowing into the new choses pipes.

Use-case name	Puncture Pipe
Short textual description	This use case involves the actions taken by saboteurs to puncture a pipe
Actors	Plumber
Dialog, scenario	1. The actor identifies a pipe that can be punctured from his position. 2. The actor punctures the pipe. 3. The pipe stops transferring water.

Use-case name	Pump Breaks
Short textual description	This is an internal system scenario when the pump just break randomly
Actors	Plumber
Dialog, scenario	1. The pump breaks randomly. 2. The actor locates it (moves to it).

Use-case name	Produce Pump/Pipe
Short textual description	The cistern produces a pump/pipe. This happens regularly. The plumber moves to the new pump to eventually put in in the grid.
Actors	Plumber
Dialog, scenario	1. The cistern produces a pump/pipe. 2. The pump shows on the screen/pipe. 3. The actor locates it (moves to it).

Use-case name	Move
Short textual description	This is the most essential action a player can take. A move can be all the thighs the specific actors derived from players can do or the player can move through the grid to reach elements he wants to change.
Actors	Player
Dialog, scenario	1. Player starts at an active element. 2. Player chooses another active element.

	3. Player moves to the other active element. Or The other scenarios Connect Pipe, Fix Grid, …

Use-case name	Start Game
Short textual description	This is the scenario when the user presses the start game button.
Actors	Player
Dialog, scenario	1. Player runs the application. 2. Player presses the "Start Game" button. 3. The select teams dialog opens.

Use-case name	Select Teams
Short textual description	This is the place where players enter their name and which team they belong to.
Actors	Player
Dialog, scenario	1. The dialog box to enter the names opens 2. Player writes his name in the team he chooses. 3. Player presses enter to confirm his choosing. 4. The game begins.

Use-case name	Change Settings
Short textual description	Players can change the settings in the main page.
Actors	Player
Dialog, scenario	1. Player presses the "Settings" button. 2. Application opens the screen to change the settings. 3. Player enters the values and presses save. 4. The game saves the new setting.

Use-case name	End Game
Short textual description	Players can end the game while they are playing.
Actors	Player
Dialog, scenario	1. Player presses the "End Game" button. 2. Application asks if the player wants to end the game. 3. Player enters "yes" or "cancel". 4. If "yes" the game ends and the players can start a new game in the main menu else they resume the same game.

2.5 Glossary

A

Active Elements - elements that are used in the game such as Spring, Cistern, and Pump.

C

Cistern - An active element that represents the water storage of the city.

Closed pipes - Pipes connected to a pump that are not actively transferring water.

Connectable pipes - Pipes that can be connected to a pump, with the number of connectable pipes specific to each pump.

D

Dangerous environment - The harsh conditions of the desert restrict movement to areas other than areas around pipes and pumps for both plumbers and saboteurs.

F

Flowable - An interface that is responsible for the water flow throughout the pipe system.

G

Grid - Represents the region of the game which is the pipe system that contains all the elements such as pipes, pumps, springs, and cisterns.

I

Input pipe - The pipe through which water enters a pump.

L

Leaking pipes - Pipes with damage resulting in water spillage, leading to loss of water flow through the pipe

M

Manufactured Pipes - New pipes are regularly produced at cisterns, with one end connected to a cistern and the other end free for water flow.

Modifiable - An interface that is responsible for the modifications that have been done by both of the teams.

Movable - An interface that is responsible for the movement of both the Plumber and Saboteur teams.

N

Network Flow - A fundamental concept in computer science with graph theory and algorithm design that has the goal involves finding a maximum flow that can be sent from a given source node to a target sink.

O

Out of Order - The state of a pump when it is not functioning correctly, resulting in no water flowing through the pump.

Output Pipe - The pipe through which water exits a pump.

P

Pipe System - A network of pipes and active elements designed to transfer water from mountain springs to city cisterns.

Players - The users who are responsible for interacting and playing with the game application. They will be the person who controls the Plumbers/Saboteur teams.

Plumbers - Individuals responsible for maintaining the pipe system, including fixing broken pumps, setting pump directions, and repairing leaking pipes.

Pump - An active element responsible for transferring water from the incoming pipe to the outgoing pipe, with a temporary water tank used as a reservoir.

S

Saboteur - Individuals with the ability to change the directions of the pumps and puncture pipes

Spring - Water source on the mountain that stores the water for transferring to the city by crossing the desert.

T

Teams - Groups of players engaged in the game, including plumber teams tasked with transferring water and saboteur teams aiming to leak water from the system.

W

Winning criteria - The objective of the game where the team collecting the most amount of water wins.

Water Tank - A reservoir within each pump used as a temporary storage for water during transfer between pipes.

2.6 Project plan

Purpose and Objectives

The purpose of this project is to develop a simulation game called "Pipes in the Desert" where players control plumbers and saboteurs to transfer or disrupt water flow through a complex pipe system in a desert environment. The objectives include creating a functional and engaging game, implementing realistic mechanics for water transfer and sabotage, and providing an enjoyable experience for players.

Assumptions and Constraints
Budget**:** This software will be designed as a university project without any considered budget.
Staff**:** 5 members, all students in BSc. Computer Science Engineering program of Budapest University of Technology and Economics
Schedule**:** This is started on 02.22.2024 and is going to end by 05.21.2024 with the workflow specified in the next section
Skill Requirement**:** Java programming language in OOP with the usage of Java libraries such as Swing, JUnit.. etc. Git/Github(Git bash command) will be also needed for version control and CI.

Workflow
Week 1 - Forming the teams
Week 2 - Requirements, project, functionally – documentation
Week 3-4 - Analysis Model - first version – documentation
Week 5 - Planning the Skeleton – documentation
Week 6 - Skeleton program – documentation and source code
Week 7-8 - Concept of Prototype – documentation
Week 9 - Detailed plans – documentation
Week 10 - Creating the Prototype part
Week 11 - Prototype program – documentation source code
Week 12 - User interface specific – documentation
Week 13 - Creating the Complete part
Week 14 - Complete program – documentation and source code
Week 15 - Summary

Deliverables

Game Design Document	**TBD**
Software Prototype	**TBD**
Final Version Release	**TBD**

Supporting Processes
- Regular team meetings for progress updates and issue resolution.
- Continuous integration and version control for code management.

Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
02.23.2024 12:00	5 hours	Mean Diamand and Taher Mured Abdulghani	Working on: - Section 2.1 - Section 2.2
02.23.2024 19:10	5,5 hours	Klevis Imeri	- Reading Sections 2.1 and 2.2 edited by Mean Diamand and Taher Abdulghani. - Added comments and suggestions. - Added a proposed part for the subsystem division in the 2.2.1 section. - Started section 2.3.1. - Wrote an email to Dr. Katalin regarding the "Check" field of functional requirements. - Draw 2 diagrams regarding the visual understanding of the problem definition to help aid the communication of the idea through the team.
02.24.2024 7:00	4 hours	Hossein Mousavi	Writing “Functional Requirements”: - Sections 2.3.1/ 2.3.2/ 2.3.3 - Completing “Functional Requirements” - Completing “Non-Functional Requirements” Writing “Essential Use-Cases”: - Sections 2.4.1 / 2.4.2 - Adding a high-level use-case diagram - Writing detailed use cases

02.24.2024 8:00	6 hours	Ibrakhim Tolobekov	- Checking the document structure in a general overview - Working with Hossein Mousavi on different sections
02.24.2024 11:10	4,5 hours	Klevis Imeri	- Refactored the functional requirements 2.3.1 with new check methods. - Ordered the functional requirements 2.3.1 in a logical order. - Added several new functional requirements 2.3.1.
02.24.2024 14:00	2 hours	Mean Diamand	- Working with Hossein Mousavi in adding more information on Section 2.5 - Glossary and Section 2.6 - Project plan. - Fixing some parts of my previous work after consulting with my teammates.
02.24.2024 14:00	2 hours	Hossein Mousavi	- Working with Mean Diamand on sections 2.5 and 2.6
02.24.2024 17:00	5,5 hours	Klevis Imeri	- Fixed and added many more functional requirements 2.3.1. - Fixed and added all Resource requirements 2.3.2 - Fixed and added all Non-functional requirements, Restrictions 2.3.3 - Set up the whole Github with the project. This included the setting.json of VsCode and the pom.xml for maven. Made sure that they applied the formatting and the coding guidelines accordingly with the Google Java Style.

			- Wrote an email to professor Katalin regarding the hardware on which our software should run.
02.25.2024 11:30	3.5 hours	Klevis Imeri	- Fixed the Use Case Diagram - Added many scenarios. - Added specific scenarios to the functional requirements in section 2.3.3. - Added the Cover and discussed with the overall documentation
02.25.2024	2,5 hours	Taher Mured	- completed the description of the functions - discussed and reviewed section 2.1 and 2.2 with Mean Diamand