Vili Virtanen, 896191, Tietotekniikka, First year student, 28.4.2021

Project documentation

Tower Defence game

General Description

Tower defence game, in which a group of enemies aim for a target through a predetermined route. The player must stop the enemies by building towers next to the path that shoot or in other way defend against the enemy. In the game there is three different towers with different types of strengths/abilities. For example, different range where the tower can hit the enemy and different damage. Enemies have three different levels of difficulty and they attack in waves. waves have different levels of difficulty too. Player gets money by destroying the enemies and that money can be spent on new towers. Player has a health, and one enemy does certain amount of damage if they get to the end.

There were some implementations left out. For example, road spikes and poison/fire towers just because I did not have enough time for them, but I think 3 different towers are enough for a small game like this. The project Itself did not have different difficulty levels.

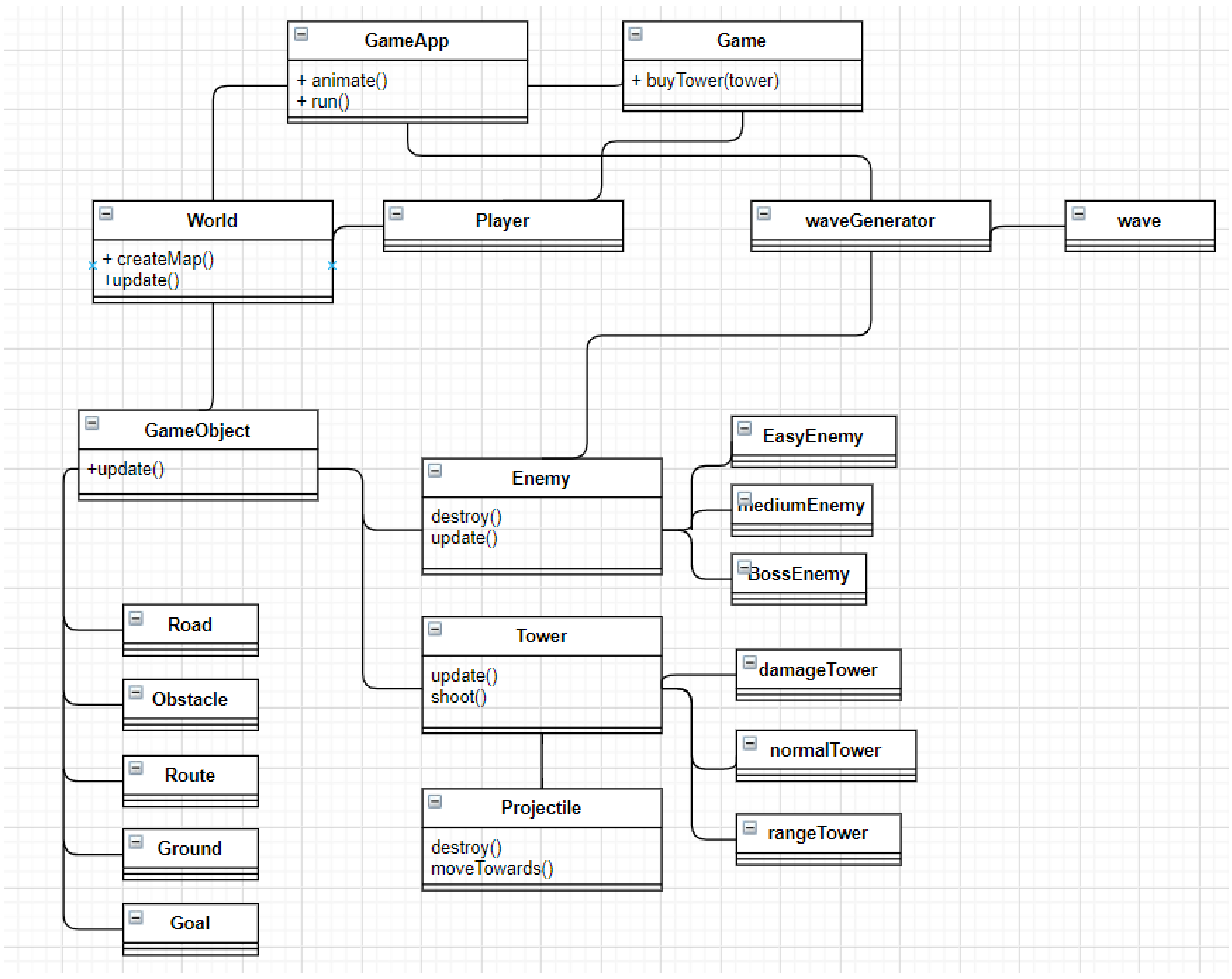
User Interface

You can start the game by going to the GameApp class and by pressing the green play button on the side or the same green run button on the top row of IntelliJ. Then the game starts automatically, and enemies start to come after 10 seconds so you have time to check the GUI buttons and so on.

There are three buttons on the left side of the interface for adding different towers and there is a label for your health and for your coins. You can decide which tower you want to buy first. When you click the button, a popup shows up with information where to put the tower. The grid is 100x100 so it is difficult to place a tower right where you want first but there is automatically a great spot for a tower in the text box at first if you want to be safe. If you place the tower in the wrong place, you get a popup warning. You should keep track of your coins, so you know if you have enough for a specific tower. If enemies get to the other end of the road, you lose damage. Easy enemies are colour yellow, normal enemies are orange and boss enemies are violet. Towers are big circles and enemies are small circles. Gray circles are just obstacles/rocks. Everything else is a square with different colours.

You should also keep in mind that the game does not scale well for full screen. It is for the best if you keep the original size of the window.

Program structure



The most important classes here are World and GameApp. World class connects most of the smaller pieces together and GameApp class uses most of the classes too. There are some changes to the plan, For example there is road and Route now and the difference is that route is the real road for the enemies and road is just for visuals so the road is not just one square wide.

GameObject is a simple abstract class that connect and updates all of the different objects in the game. Road, obstacle, route, ground and goal are all simple subclasses of gameobject for representing the real map of the game. Enemy has different methods like update which it uses to check the next direction and moves there. It also has a destroy method which is used for destroying dead enemies. Tower class has update method too but uses it differently. It checks for the nearest enemies and shoots them if they are in range of the specific tower. Projectile on the other hand is more a visual class and it moves toward the targeted enemy.

WaveGenerator is a simple file reading class that creates the enemy waves adding them to a buffer of waves. Waves consists of buffer of enemies. The game class itself does not have a lot of use because I thought that this way it is easier to implement the game. It still controls the buying of the towers ingame. Player is just a simple class with players health and coins.

Algorithms

The most important algorithms are Enemies movement, projectiles movement and towers shooting algorithm. The gameApp has some simple algorithms for drawing enemies and projectiles too.

Enemy movement algorithm checks all four directions (Up, left, down, right) from the worlds map for gameobjects called Route. If it finds it, enemy moves to that location and updates its location. It also adds a route to the last location so next enemies can find their way to the end. Algorithm also checks that the enemy in question has not been at the next location. This prevents enemies from “running in circles” or any other exceptions.

Projectiles movement is a bit different. First it counts the distance between target and projectile with mathematical algorithm. (sqrt(dx^2+dy^2)). If the projectile is not close enough, it moves toward the target. When moving, it checks if the difference of x is bigger than the difference of y. If x is bigger, it checks if it must move left or right. Then it moves 4 blocks in x axis and 2 blocks on y axis. And of course, if y is bigger, it moves move up/down than left/right. This was simple and easy way of making projectiles move towards their target. It is a bit rough because they move in kind of zigzag motion.

Towers shoot algorithm uses the same distance algorithm (sqrt(dx^2+dy^2)) for checking if there are any enemies inside its range. If there is it shoots projectiles towards it and counts if the enemy health is 0 or not. If its zero, the enemy is destroyed, and the player earns 10 coins.

The gameApp uses few algorithms for different things. It checks if the waveGenerator has any enemies inside and spawns the first one. Then it checks again if there are left any enemies. If not, it takes the next wave in waveGenerator and does the same. If there are no waves left, it just spawns easy enemies every second. It also uses some algorithms to draw the game and for updating it. It checks all the enemies and projectiles currently in the world and draws them in their location. Then it draws back the correct object to the last position of the enemy or projectile. Placing towers also uses a simple algorithm checking the correct location and if it is possible to place there. It checks which tower is being added and places it if there are coins and the place is correct.

When the game is first started the gameApp checks for the world classes map and places every object to the right place.

Data Structures

I tried to use immutables as much as possible, but still, most of my collections are mutable because the game needs mutable collection to change the state of the game. So, for collections I chose buffers because they were the easiest ones to play around to me and I have the most experience with them. Buffers like the 2d map of the game needs to be updated constantly so it must be mutable. For different variables I tried to use Val as much as possible, but some variables needed to be Var so they can be updated. There were many options for collections like lists and arrays, but I chose buffer because I just felt most comfortable with it.

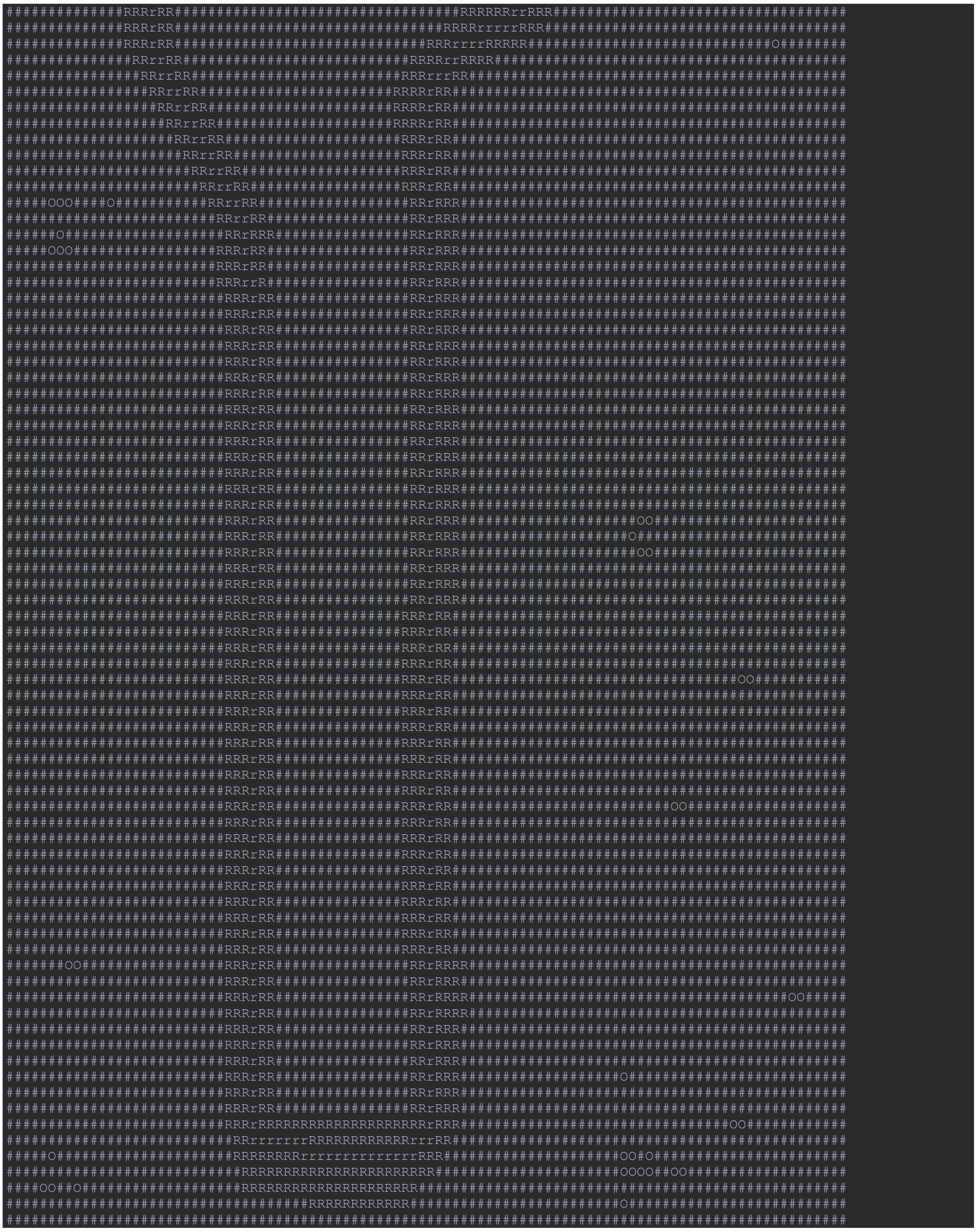
Files

I used external files for creating the base map and for creating the waves of enemies. For both I used simple text files that I my self wrote.

For the map I used 100x100 "grid" of characters. In the file there is 6 different characters with different meanings. "#" means ground object, "O" means obstacle, "R" means Road, "r" means predefined route, "g" means goal and "s" means start. The map is not the easiest to create but because its so large it is better for creating more creative maps in the future if I want.

The first map is written like this:

```
##############RRRgRR###########################OOOO#################################################
##############RRRrRR############################OO##################################################
##############RRRrRR#####################################################################OOO########
##############RRRrRR######################################################################OO########
##############RRRrRR###############OOO##############################################################
##############RRRrRR################O###############################################################
##############RRRrRR#########################################################################RRRRRRR
##############RRRrRR#############################################################RRRRRRRRRRRRRRRRRRR
##############RRRrRR#########################################################RRRRRRRRRRRRRRRRRRRRRRR
##############RRRrRR##################################################RRRRRRRRRRRRRRRRRRRrrrrrrrrrrs
##############RRRrRR####################O###########################RRRRRRRRRRrrrrrrrrrrrrRRRRRRRRRR
##############RRRrRR#############################################RRRRRRRRRRRrrrRRRRRRRRRRRRRRRRRRRRR
##############RRRrRR############################################RRRRRRRRRRrrrRRRRRRRRRRRRRRRR########
##############RRRrRR###########################################RRRRRrrrrrrrRRR######################
##############RRRrRR##########################################RRRRRrrRRRRRRRRR######################
##############RRRrRR##########################################RRRRrrRRR#############################
##############RRRrRR##########################################RRRRrRR###############################
##############RRRrRR##########################################RRRrrRR###############################
##############RRRrRR##########################################RRRrRR################################
##############RRRrRR#########################################RRRrrRR################################
##############RRRrRR########################################RRRRrRRR################################
##############RRRrRR######################################RRRRrrrRRR################################
##############RRRrRR####################################RRRRRrrRRRR#################################
```



```
##############RRRrRR##################################RRRRRRrrRRR###################################
##############RRRrRR################################RRRRrrrrRRR#####################################
##############RRRrRR##############################RRRrrrRRRRR##############################O########
###############RRrrRR###########################RRRRrrRRRR##########################################
################RRrrRR#########################RRRrrrRR#############################################
#################RRrrRR#######################RRRRrRR###############################################
##################RRrrRR######################RRRRrRR###############################################
###################RRrrRR#####################RRRRrRR###############################################
####################RRrrRR#####################RRRrRR###############################################
#####################RRrrRR####################RRRrRR###############################################
######################RRrrRR###################RRRrRR###############################################
#######################RRrrRR##################RRRrRR###############################################
#####OOO####O###########RRrrRR##################RRrRRR##############################################
#########################RRrrRR#################RRrRRR##############################################
######O###################RRrRRR################RRrRRR##############################################
#####OOO#################RRRrRR#################RRrRRR##############################################
#########################RRRrRR#################RRrRRR##############################################
#########################RRRrrR#################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR#####################OO#######################
##########################RRRrRR################RRrRRR####################O#########################
##########################RRRrRR################RRrRRR#####################OO#######################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR##################################OO###########
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR##########################OO###################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
##########################RRRrRR###############RRRrRR###############################################
#######OO#################RRRrRR################RRrRRRR#############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRRR######################################OO#####
##########################RRRrRR################RRrRRRR#############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR###################O##########################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRR################RRrRRR##############################################
##########################RRRrRRRRRRRRRRRRRRRRRRRRrRRR#################################OO###########
###########################RRrrrrrrRRRRRRRRRRRRrrrRR###############################################
#####O#####################RRRRRRRRrrrrrrrrrrrrrrRRR#####################OO#O#######################
############################RRRRRRRRRRRRRRRRRRRRRRR######################OOOO##OO###################
####OO##O###################RRRRRRRRRRRRRRRRRRRRR###################################################
####################################RRRRRRRRRRRR#########################O##########################
####################################################################################################
```

For the wave file I did the same kind of format with different character meaning different enemies, but it does not have any constrains in size. I can make any kind of waves with different sizes if I want. In the file there is three different characters. “e” means easy enemy, “m” means medium enemy, and “b” means boss enemy. Every row is one wave of enemies. My first test file for the game is written like this:

```
eeemeebeeeeeeeemmmmm
eeeeeeeeeeemmmmmmbmm
eeeeemmmmmmmmmmmmmbb
eemmemmmemmmmmmbmmbb
mmmmmemmmmmmmmmmmbbb
eeebbeeebbmmmbeeeeee
mmeebbmmmmmmbbbmmmmb
mmmmmbbbmmmmbmbmeeee
eebbmmmmeeeemmmmeemm
bbbbmmmeeemmmmbbmmmm
bbbbbbmmmmbbbmmmmmmm
```

I chose this format because it is easy to create new maps and waves however you like if you just put a bit of time to it. Also, it was the easiest way for me to read from file with buffered readers. I just read one row at a time and create the correct pieces for the game.

Testing

The testing of my game differs from my plan a bit because I really did not know a lot about testing beforehand. I had to learn a lot about unit testing and stuff while working on this game. Most of my testing was done with unit tests. I created a SetSuite that extends AnyFunSuite. I found this scalatest class from the internet and because it worked well, I chose to go with it. I am not sure if I was supposed to save all of my tests, but I just erased the old ones and created new ones in the place so my GameTests class does not have a lot of different tests. For example, destroying the enemies and projectiles were tested with unit tests. If the enemy disappears from the current enemies’ buffer, it worked an so on. Most of my methods were tested this way and along the way I had to make a lot of improvements in the tests and the methods.

Other testing was done by running the game itself. I tried to run it with incorrect and different values and methods that does not work and that way I found a lot of mistakes and bugs. I used try catch structure for the file readers and that way I found some mistakes and bugs too.

The GUI was a big part of my testing as well. I know it is not the best testing method, but this way I could see a lot of the mistakes with my eyes, and it was easier to understand them. With GUI, I tested and improved the enemy movement and the projectile movement a lot. If I did not like the way the projectiles moved towards the enemy, I tried with different type of values and implementations. I could also improve the towers range and placement with GUI testing.

This is just a quick rundown of all the testing that was done, and it is not all the testing that I did. I still do not know a lot about testing, but these testing methods helped me a lot with the game development.

Known bugs and missing features

There are some bugs in the game. Making the GUI was completely new thing for me so there are probably more bugs that I do not know about but at least the drawing of the enemies and the projectiles are sometimes little buggy. When a tower shoots an enemy and it destroys it, the GUI still draws a projectile to the map that does not sometimes disappear but stops at the place it was at the moment. Enemies when they die might not disappears at first because it was hard for me to implement something that draws the road tile back to the place where the enemy died. This is because every time I tried to fix this, it just created a bug where enemies are not drawn to the GUI at all. This is all because the way I implemented the drawing of the objects, but I tried many ways, but I never got anything else to work other than this way. Sometimes the placement of the towers bugs out too when close to the road or to the edges of the map. There is a bug where you can accidentally place tower on top of each other and way it is possible to sometimes place towers on top of the road because towers are few tiles wide. This does not always happen, and I have no idea why it happened. This was just one time it happened but that means it is a bug. I am still not sure why it happens sometimes and sometimes it works just right.

For missing features/ partial features there are few. The wave generator works perfectly but when the game spawns enemies, there is not a significant pause between different waves. I tried to work with threads and making them wait some seconds, but it did not work with my implementations. One other bad /“missing” feature is how the projectiles work. There is no real connection with projectiles hitting the enemy. They are simply for visual effect. This might be the cause of some other bugs in the game, but I did not find why this could not be done like this. It is a bit simple and stupid way to make projectiles in the game, but I almost threw away the whole concept of projectiles because I did not get it to work better. There is no pause button or a starting menu of some kind for the game which would be nice, but I simply just did not have the time to do them because I had so much other schoolwork to do.

There are some ways to fix these problems. If I had more time, I probably could change the whole GUI to work with real coordinates and not a simple Grid. This would make it easier to implement better movement and working projectiles to the game. My original idea was to make the game like this but when I tried to do it and read from the internet about it, I just could not get it to work so I had to throw it away and do the grid type implementation.

3 best parts and weaknesses

Best parts of my code are in my opinion the file readers, the way enemies and towers are implemented from abstact classes and the easiness of adding new towers, enemies, maps and waves.

The file readers work well and by using try catch method, it also checks for mistakes in the files and other exceptions. I like this because this was still a bit new concept for me, but I got them to work just the way I wanted.

The implementation of the enemies and towers are pretty good for my level of coding too. I like how they are all game objects but have completely different use. It is so easy to change different enemy’s health or damage because the enemy class is abstract too. All enemies get the same methods from the abstract class enemy but still can easily be changed to completely different enemies. Same with towers.

The easiness of adding different things and changing this are also in my opinion well done. Map is easy to write if you have time and you can be creative with it. New waves can be added just by writing for example (eeemmmmbbbee) to the waves file. Enemies can be added just by creating new enemy subclass and adding it to the wavegenerator. Towers might seem pretty tricky, but it is easy to just add a new button and add the tower to the game apps button action method.

Biggest weaknesses are GUI, projectiles and the fact that GUI is not independent but relies on other classes a lot. The GUI like I said was new thing for me, so its implementation is rough. The animating of projectiles and enemies is blocky, and the GUI lags a lot when animation tics. I have two different tickers for spawning enemies and for drawing them which is not the worst but could be implemented better. The Grid does not scale to Fullscreen and is not the best way of making the game.

The projectiles like I said before are just for visuals and they bug out sometimes. The movement could be a lot better if I had better coordinate system so they would move straight and not by blocks of the grid.

The GUIs dependency on other classes was necessary evil for making the game because I had no experience on GUIs and it was already hard to make the game to work like this. Ways to fix these problems is to implement real vector type coordinates and making the Game class better so it is reliant on the classes not the GameApp.

Deviations from the plan, realized process and schedule

The schedule in hours was close to correct and I went with it. But the order of implementing things weekly was not the same. In the first weeks I had more time to do stuff, so I was able to create a lot more than just the outline of the game. I started working on the update methods for enemies and towers and made the world class to read from the file already. I also made the GUI already in the first weeks to some extent.

Next weeks I also made more progress even though I had a lot of learning and reading to do. I started working on the enemy spawner and shooting of the towers. I also added new towers and enemies to the game. I also made buttons for the towers to be added. I was bit ahead of the schedule and implemented this in different order because it just felt right time for them.

Next weeks I almost completed the game. I added the wave generator and tried new movement styles for projectiles. I had a lot of time for testing and optimizing the game.

And the last weeks I have been optimizing and testing the game a lot more. I made new popups for error situations and game ending. My schedule was rough and by doing the game I deviated from the plan a lot because sometimes I wanted to do different thing and sometimes it just felt like the right moment to implement this and that. I learned that I can always try to make a good plan but sometimes you have to just go with the flow to get to the end.

Final evaluation

I am happy with the outcome of the project. I got to learn a lot of things about making a project and learned a lot from my mistakes. The game itself works relatively well in my eyes and I was really scared about my coding skills. The GUI is the biggest shortcoming of the project with the projectiles, but this was pretty much expected, because I did not know anything about GUIs when I started.

In the future this could be improved by making better maps and optimizing the game more. There is a lot to be improved in the game, so it works better overall and looks better. Data structures were chosen pretty well because the game need mutable structure so it can be changed. The whole structure of the project is pretty suitable for future improvements except maybe the GUI would have a lot of work. But adding new maps and enemies and towers is easy in it, so I think that is a big win for me. Overall, I did good for myself and I did not expect even this level of game to be finished. I know there is a lot of simple and dumb solutions for problems in my project, but I do not have the necessary skills or time to make a better game. I hope I learned a lot from this project and maybe next time I will know make better decisions for problems in hand.

If I had to start over this project, I would spend a lot more time working on the GUI and making it less dependant on other things. I would want to make a real coordinate system with direction vectors and all so it would run better and smoother. With more time I would like to make it a real and fun game that could be played a lot.

References

https://stackoverflow.com/ (Helped me a lot when I had questions about something. Usually, I took reference from java code and made it better for my project. I did not copy anything from here I just needed some direction to go to with my problems and this helped.)

https://www.scalafx.org/docs/home/ (This helped me a lot too with the implementation of the GUI. I took reference for example for popups and button mechanics.)

https://www.youtube.com/channel/UCEvjiWkK2BoIH819T-buioQ

(These videos about scalafx helped me with GUI too.)

https://gameprogrammingpatterns.com/singleton.html

(this book helped too with some simple problems.)

https://plus.cs.aalto.fi/studio_2/k2021/modelproject/

(The model project helped me to get started and I took some reference from it)

Also, the A+ project tutorials like scalaFx tutorial helped me a lot when starting the project.

https://www.scala-lang.org/api/2.13.5/ (scala documentation helped a lot too with collections and more)

https://www.scalatest.org/user_guide/using_scalatest_with_intellij

https://www.scalatest.org/user_guide/writing_your_first_test

(for unit testing)

There is probably a lot that I do not remember because I used google a lot with simple problems like getting the 2d buffer to work and other small things.