Project Document

1. **Personal information:**

   - Project name: Fortress Fury

   - Topic: Tower Defense

   - Last name: Nguyen

   - First name: Vu Minh

   - Student number: 101676647

   - Degree program: Digital Systems and Design

   - Year of study: 1

2. **General description:**

   - Tower defense is a game where you have to place gun towers or use abilities to stop the attack of the enemy troops and protect the headquarter. Enemies will follow a specific path to the headquarter, and player has to place gun towers along the path to defense. The player can also use special ability to defense. The game progress can be saved to be continued playing later, and after the game end, the records will be saved for leaderboard statistics.

   - Difficulty: Demanding

3. User interface:

   a. Lobby Scene:

Figure 1: Lobby Scene

- **Click *Play*** to be diverted to *Tutorial Scene*.

- **Click *Continue*** to continue playing existing game (the button will not be visible if there are no existing game). Player will be diverted to *Game Scene*.

- **Click *Leaderboard*** to be diverted to *Leaderboard Scene*.

   b. Tutorial Scene:

Figure 2: *Tutorial Scene*

- **Click *Back arrow*** to be diverted to *Lobby Scene.*

- **Click *Next arrow*** to be diverted to *Map Selection Scene*
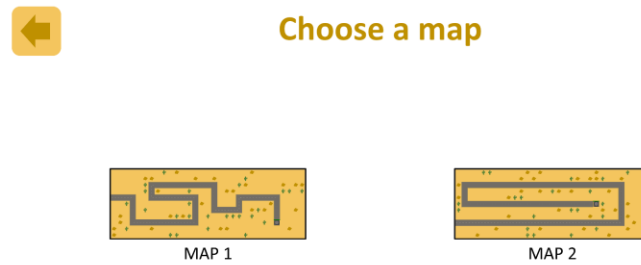
c. Map Selection Scene:

Figure 3: Map Selection Scene

- **Click either map** to be diverted *Game Scene* with chosen map.

- **Click *Back arrow*** to be diverted to *Lobby Scene*.
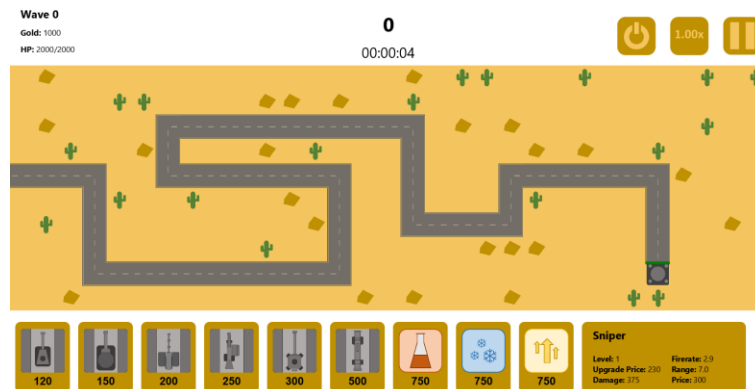
d. Game Scene:

Figure 4: Game Scene

i. Top:

- On the left are basic statistics.

- In the middle are *score* and *timer*.

- On the right are 3 buttons:

    o **Click *quit*** to quit the game, save the game and be diverted back to *Lobby Scene*.

- o **Click** *speedup* to switch among 5 speed levels (0.5, 0.75, 1, 1.5, 2).

- o **Click** *pause* to pause/resume the game.

ii. Bottom:

- • Six first buttons are gun towers that player can buy. **Click a gun tower** to select it, and the information of the selected gun will be display on the center bottom.

- • Three next buttons are 3 abilities player can use: poison, freeze, and rage.

- • The last one is the information of the selected gun tower.

- • **Press "y"** to deselect a gun.

iii. Middle:

- • After selecting a gun from the bottom right, **click an empty square** to place in at that square. The action will fail if the player's gold are not sufficient.

- • **Click an existing gun** to select in. The information of the selected gun will be display on the bottom right.

- • **Press "u"** to upgrade selected gun.

- • **Press "r"** to remove selected gun.

- • **Press "y"** to deselect a gun.

iv. When game is over:

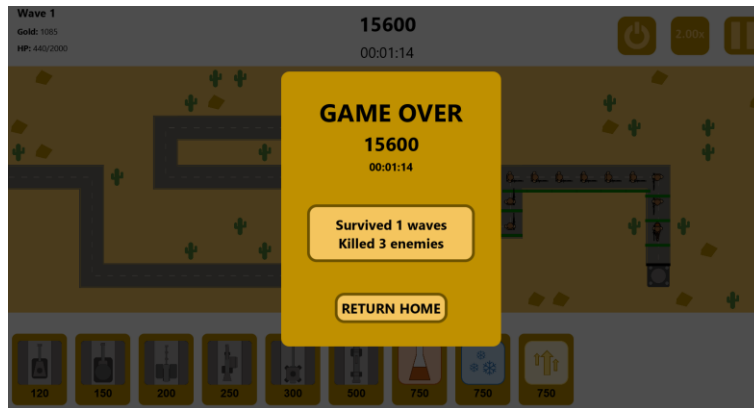*Figure 5: Game Scene when game over*

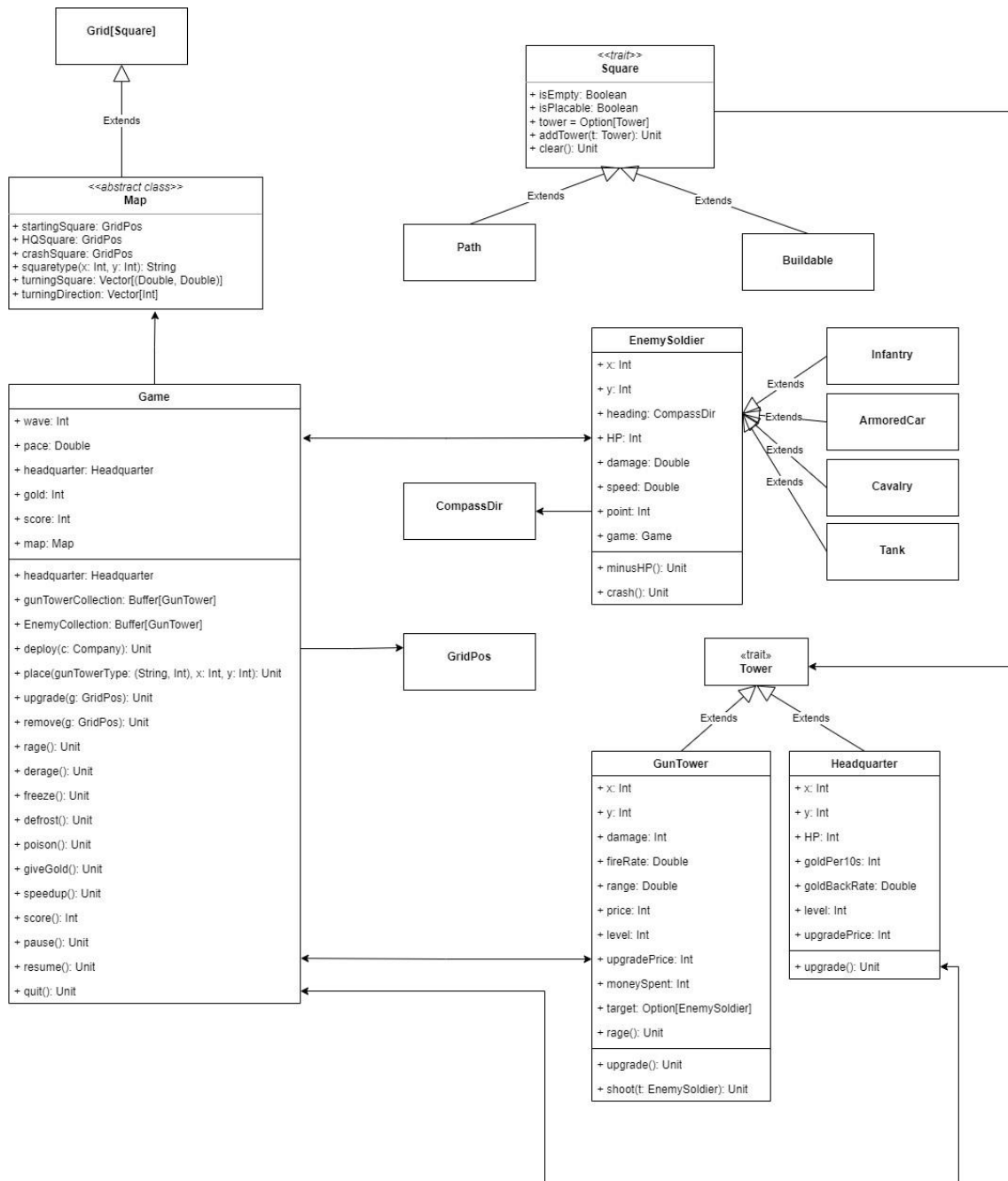- **Click** *Return home* **to be diverted to Lobby Scene.**

e. Leaderboard Scene:

Figure 6: Leaderboard Scene

- The records are sorted by score.

- **Click** *Back arrow* **to be diverted to** *Lobby Scene*.
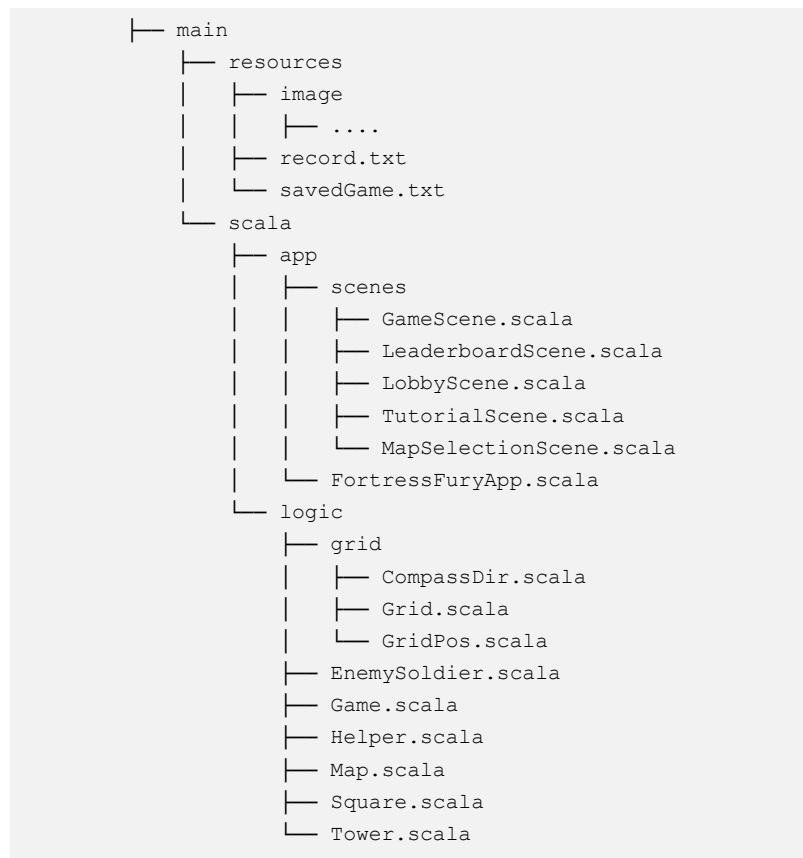
4. Program structure:

Grid[Square]

Extends

<>
Map
+ startingSquare: GridPos
+ HQSquare: GridPos
+ crashSquare: GridPos
+ squaretype(x: Int, y: Int): String
+ turningSquare: Vector[(Double, Double)]
+ turningDirection: Vector[Int]

Game
+ wave: Int
+ pace: Double
+ headquarter: Headquarter
+ gold: Int
+ score: Int
+ map: Map

+ headquarter: Headquarter
+ gunTowerCollection: Buffer[GunTower]
+ EnemyCollection: Buffer[GunTower]
+ deploy(c: Company): Unit
+ place(gunTowerType: (String, Int), x: Int, y: Int): Unit
+ upgrade(g: GridPos): Unit
+ remove(g: GridPos): Unit
+ rage(): Unit
+ derage(): Unit
+ freeze(): Unit
+ defrost(): Unit
+ poison(): Unit
+ giveGold(): Unit
+ speedup(): Unit
+ score(): Int
+ pause(): Unit
+ resume(): Unit
+ quit(): Unit

<<trait>>
Square
+ isEmpty: Boolean
+ isPlacable: Boolean
+ tower = Option[Tower]
+ addTower(t: Tower): Unit
+ clear(): Unit

Extends   Extends

Path

Buildable

EnemySoldier
+ x: Int
+ y: Int
+ heading: CompassDir
+ HP: Int
+ damage: Double
+ speed: Double
+ point: Int
+ game: Game
+ minusHP(): Unit
+ crash(): Unit

CompassDir

GridPos

Infantry

ArmoredCar

Cavalry

Tank

Extends
Extends
Extends
Extends

«trait»
Tower

Extends   Extends

GunTower
+ x: Int
+ y: Int
+ damage: Int
+ fireRate: Double
+ range: Double
+ price: Int
+ level: Int
+ upgradePrice: Int
+ moneySpent: Int
+ target: Option[EnemySoldier]
+ rage(): Unit

+ upgrade(): Unit
+ shoot(t: EnemySoldier): Unit

Headquarter
+ x: Int
+ y: Int
+ HP: Int
+ goldPer10s: Int
+ goldBackRate: Double
+ level: Int
+ upgradePrice: Int

+ upgrade(): Unit

```
├── main
    ├── resources
    │   ├── image
    │   │   ├── ....
    │   ├── record.txt
    │   └── savedGame.txt
    └── scala
        ├── app
        │   ├── scenes
        │   │   ├── GameScene.scala
        │   │   ├── LeaderboardScene.scala
        │   │   ├── LobbyScene.scala
        │   │   ├── TutorialScene.scala
        │   │   └── MapSelectionScene.scala
        │   └── FortressFuryApp.scala
        └── logic
            ├── grid
            │   ├── CompassDir.scala
            │   ├── Grid.scala
            │   └── GridPos.scala
            ├── EnemySoldier.scala
            ├── Game.scala
            ├── Helper.scala
            ├── Map.scala
            ├── Square.scala
            └── Tower.scala
```

Figure 7: Class structure

- All classes and sub-classes in package logic are used to support class Game. An instance of class Game is created in class GameScene, and class GameScene will control the flow of the game.

- All classes in package *app* are used to implement the user interface.

- There is a *Helper.scala* object which contain constants and helper methods.

- Other solutions:

  o Implement *Ability* as a trait with 3 sub-classes: This is unnecessary since 3 sub-classes can be implemented as 3 methods in class *Game*.

  o Implement *GameOver* as a scene: This is unnecessary since it can be done by adding a pane on top of *GameScene* when the game is over (see Figure 5).

5 . **Algorithms:**

- **Pythagoras theorem ($c^2 = a^2 + b^2$):** This is used to check if an enemy soldier is in the range of a gun tower.

- **2-argument arctangent (θ = atan2(x,y)):** This is used to calculate the angle of the line crossing the gun and its target.

- **Linear relationship:** Initially, the enemies and the square they are on are not aligned. The further the coordinate from (0, 0), the greater the misalignment. I find the misalignment of each coordinate and use Linear regression calculator to find the slope.

- **Find the target of a gun tower:** Iterate over enemies in the order of their deployment and return the first enemy that is in the range of a gun tower. In this way, the gun will prioritize shooting the enemy that is closer to the headquarter.

6. **Data structure:**

- **Buffer:** This is used to store enemies and gun towers on the map since they need to be mutated frequently.

- **Vector:** This is used to store constant information (pace, gunInfo, etc.) since they do not need to be mutated.

- **Seq[Seq[String]]:** This is used to extract information from .txt files. This is the most viable solution since the .txt files are in *tab-separated values* (TVS) format (see Figure 8, 9)

7. **Files:**

- There are 2 files in the project: *savedGame.txt* and *record.txt.*

- They are .txt file, but both of the are formatted in TVS format: Data in 1 line are separated by *tab character* ("\t") and lines are seperated by *new line character* ("\n")

a. *savedGame.txt*:

- **Line 1** includes 2 datas: the number of towers (n) and the number of enemies on the map (m).

- **Line 2** includes 8 datas: *score, gold, wave, time, nextDeployTime, cannotDeployUntil, enemyKilled,* and *mapType*.

- **Next n lines** each includes 4 datas (*name, x, y, level*) of n towers, including the headquarter.

- **Next m lines** each includes 3 datas (*name, stepTaken, HPpercentage*) of m enemy.

- **The last line** includes information about undeployed enemies.

- **Example:**

Figure 8: savedGame.txt

b. **record.txt:**

- Each line includes 6 datas: *date, mapType, score, survivedTime, survivedWave,* and *enemyKilled*

- **Example:**

Figure 9: record.txt

8. **Testing:**

- **Unit testing (using test file):** *giveGold(), deploy(enemy), enemy.advance(), place(gun), gun.shoot().*

- **System testing (using GUI):**

    o   Try to place a gun on occupied square or on the road.

    o   Try to buy a gun or ability when having not enough money.

    o   Try to perform actions when the game is paused.

    o   Try to pause, resume, speedup, quit, continue.

    o   Try to remove the headquarter.

    o   Try to crash the headquarter.

9. **Known bugs and missing features:**

- When changing the scene, there is a glitch, which makes panes unaligned. This can be fixed by slightly adjust the window size.

- On Mac devices, the enemies are off the center of the path (but the scene changing glitch does not occur).

- Potential crash.

10. **3 best sides and 3 weaknesses:**

- **3 best sides:**

    o   Clearly structured, user-friendly user interface.

    o   Has many interesting features: speed up, quit, save game, etc.

    o   Relatively clean code, with comments to divide it into different sections.

- **3 weaknesses:**

    o   Potential performance issues.

- o Require keyboard to play the game (most tower defense game does not require)

- o No usernames to identify whose record it is.

11. **Deviations from the plan, realized process and schedule:**

Week	Plan	Actual progress	Progress evaluation
11.2 – 24.2	Finish basic class implementation.	Initialize classes.	Behind schedule
25.2 – 9.3	Finish complex methods and finalize enemy soldiers' and guntowers' attributes.	GUI: worked on the basic of *GameScene*. Logic: finished *Map*, *Towers*, a part of *Game*.	GUI is far beyond schedule, but logic is behind schedule.
10.3 – 23.3	Start working on UI.	GUI: had *LobbyScene* and the basic of *GameScene* done. Logic: finished the basic of all classes.	On schedule
24.3 – 6.4	Keep working on UI and test the functions via UI.	GUI: finished other scenes. Logic: finish some I/O functions.	On schedule
7.4 – 17.4	Finish UI and finalize documents.	GUI: Reformat the layout of all scene.	On schedule

- I decided to start UI earlier than in the plan, because it made testing functions easier.

- The biggest difference is I worked with logic and GUI concurrently, instead of finishing logic then move to GUI.

- The order of implementation of the project is to implement small classes first (*Towers, EnemySoldiers*) first, the implement bigger one (*Game, Game Scene*). After finising the core features, I moved on to other scenes and minor features (*saveGame, LobbyScene, LeaderboardScene*).

- During the process, I got to know many new features of the language Scala. It is also the first time I got to use the I/O features (apart from A+ exercises).

Besides, when trying to make the layout and color of the scenes harmonious, I learned something about arranging and coloring in a design.

12. **Final evaluation:**

- **Good and weak aspects:** The game layout is familiar, users can easily navigate through different scenes. There is a *TutorialScene* to help players to use the *GameScene*. However, there are no usernames to identify whose record it is, but for an offline game, it is not necessary. Last but not least, for a tower defense game, it would be smoother to navigate through the game using only mouse instead of having to use keyboard.

- **Shortcoming and explanation:** The program has potential performance issues. This is due to the method *updateTimerText()* in GameScene being called once every 0.2 second in normal speed or 0.1 second in x2 speed.

- **Shortcoming and explanation:** The movements of the enemies are always on track on Window devices, but are off track on Mac devices. This is due to there are 2 layer of pane, one is a *GridPane* for the squares on the map, the other is a *StackPane* for the enemies. On Window devices, these 2 panes align perfectly, but on Mac devices, there is a slight unalignment, leading to the unalignment of the enemies.

- **Possible future improvement:**

    o   Make the game keyboardless

    o   Find another way to implement enemies movement

    o   Have enemies or guns to operate on their own, instead of using *foreach* to control each of them.

- **Different choices of solution/data structure/class structure:** I think the general structure of my solution is relatively good. The program is easy to make changes or extensions. By adding a few more lines, you can add/remove guns, enemies, map, etc.

- If I started the project again from the beginning, I would research the tower defense game pattern and the scalafx documentation from A+ more carefully. This would lead to fewer refactoring.

13. References:

- How to write a file: https://alvinalexander.com/scala/how-to-write-text-files-in-scala-printwriter-filewriter/
- How to use Thread: https://alvinalexander.com/scala/how-to-create-java-thread-runnable-in-scala/
- How to update UI in a thread: https://stackoverflow.com/questions/32640697/updating-ui-from-a-background-thread-in-scalafx
- O1 Library grid: https://gitmanager.cs.aalto.fi/static/O1_2023/modules/given/O1Library/doc/o1/grid.html
- Gitlab project link: https://version.aalto.fi/gitlab/nguyenv23/fortress-fury

14. Appendices:

Gun towers	Cannon	Grenade Launcher	Turret	Rocket Launcher	Sharpshooter	Sniper
Damage	90	270	90	525	45	375
Fire rate	0.6	1	0.6	1.4	0.8	3
Range	3	3	5	5	7	7
Cost	150	250	200	500	120	300
Upgrade price	100	160	130	330	80	200
Upgrade constant	105%					
Damage/s	150.00	270.00	150.00	375.00	56.25	125.00
Damage/s/gold	1.00	1.08	0.75	0.75	0.47	0.42

Table 1: Gun towers statistics

Enemy soldiers	Infantry	Calvary	Armored car	Tank
Base damage (bd)	200	400	600	800
Damage (dmg)	$bd * 1.05^{wave}$			
HP	dmg			
Gold	dmg/100			
Point	dmg*10			

Table 2: Enemy soldiers statistics

	Headquarter
HP	20000
Gold/10s	10
Gold back	60%
Upgrade price	500+200*level
Upgrade constant	105%

Table 3: Headquarter statistics