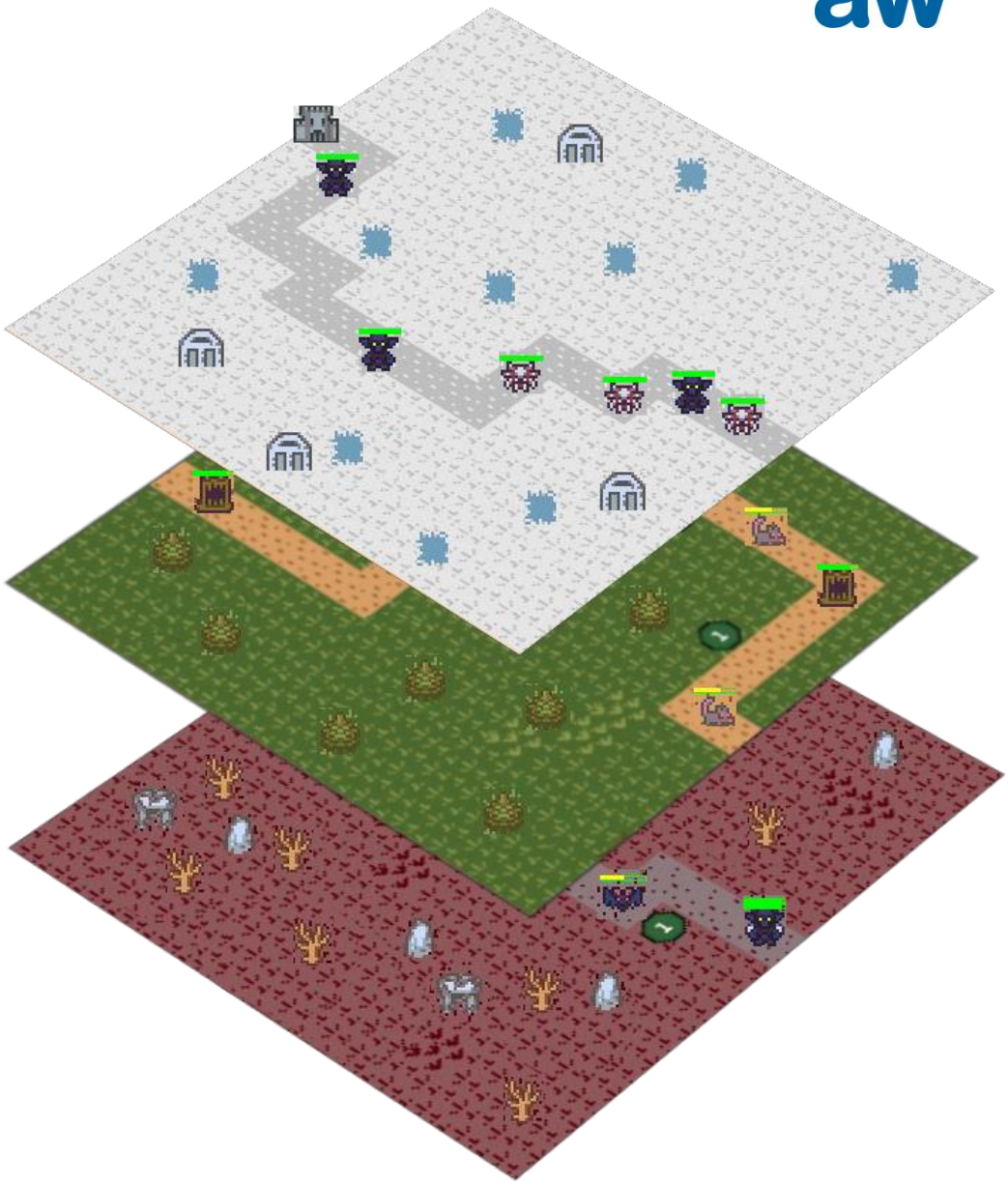
Zürcher Hochschule
für Angewandte Wissenschaften

zh aw School of Engineering



TOWER HOPSCOTCH

Final Report

S. Bösch, N. Eckhart, R. Emberger and P. Meier

Table of Contents

1. Project Outline

1.1. Starting Position

The market for video games is currently in a continuous state of growth with an estimated increase of almost eight percent in 2017. [1] While many start-ups and small businesses are entering the business to find success, the demand for original and entertaining games is still growing. While all this new content is being released, the strategy game subgenre and its sizeable and enthusiastic fan base finds itself somewhat neglected. We believe that this is something that can and should be changed.

1.2. Idea

The plan is to develop a tower defense strategy game for desktop PCs. The primary objective will be to defend a central structure over multiple landscapes against waves of enemy units. This is accomplished by building defensive structures to divert, impede and destroy the incoming foes. Each enemy that is stopped will provide the player with a set amount of currency that allows him to further improve his defenses.
In our game the player must manage multiple different maps at once, all of which have enemies that move towards the central structure. These maps can be either automatically generated or the player may design his own levels and play on them.

1.3. Customer Value

- The customer will be provided with an entertaining and unique strategy game with multiple original twists.
- The game will run all major operating systems that support a Java environment.
- It will feature increased challenge by having the player manage multiple maps at once.
- The customer will be able to choose to create his own unique maps for improved replay ability and enjoyment.

1.4. State of the Art – Competition Analysis

There are many tower defense games which gained much popularity over the years. The most successful game released 2009 and is still a name people know: Plants vs. Zombies. [2] It uses a small n by m grid where the enemies approach from one side and the “towers” are placed on the grid to fight against them. The enemies can destroy the towers when they’re near enough. Killed enemies give the player money to upgrade or place new towers.

1.5. Main process

The main process is the player who plays the game

- The player starts the game, which he already has installed.
- He selects the standard mode.
- The player receives a specific amount of game currency.
- The player spends the money to place towers.
- The enemies are coming in waves.
- The towers try to shoot down the enemies.
- A tower kills an enemy and the player receives money, which he can spend again on towers.
- An enemy reaches the players base and the base loses some health points.
- The player survives all waves and wins the game.

1.6. Economics

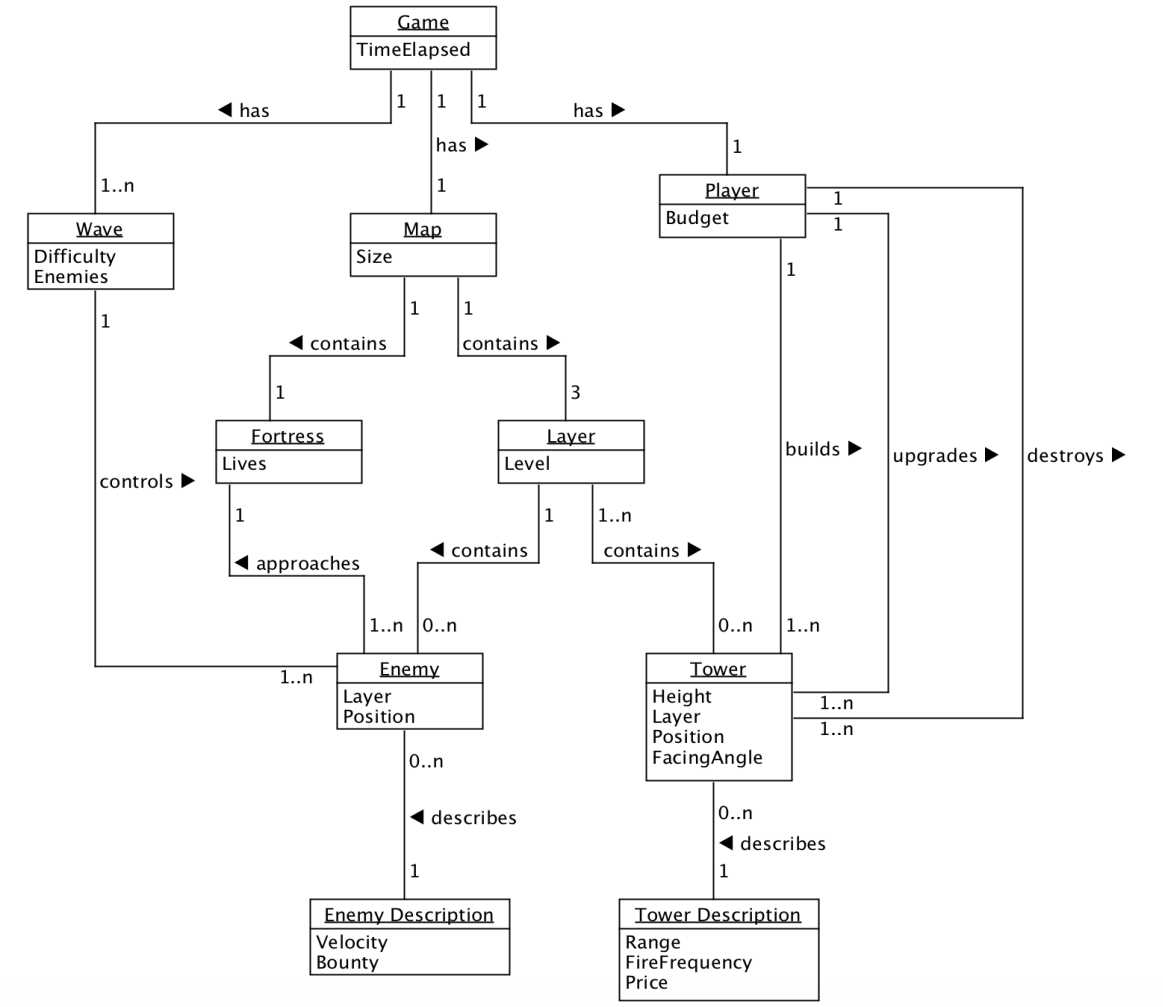
We'll each work for 120 hours, which accumulates to a 480 hours project. Our goal is to pay ourselves about 40 francs per working hour. Together with the marketing campaign and a reasonable time buffer for testing and unexpected bug issues or changes, the project will cost about 35’000 francs.
The game will be sold on Steam (biggest online video game store) for 10 francs. Which means that at least 3500 copies need to be sold to break even. Steam has a multi-million audience and is a great platform to sell games on. We estimate to break even after about 4 months.
The first year will probably be the most successful, because the typical gamer craves innovation and new experiences. A 50% decrease in sales per year is a realistic vantage point. After 5 years, the estimated income will be around 220’000 francs minus the first investment.

The economic efficiency calculation can now be carried out with the exact effort for the prototype!

2. Analysis

2.1. Domain Model Diagram

A short introduction to what follows would be useful.



The introduction is still a bit incomprehensible!

The central problem domain in our application is the game concept itself. It ties everything together as shown in the diagram. The individual parts are explained here:

- **Player:** The Player has a Budget, with which he can build or upgrade towers. He can also destroy towers to get a fractional amount of money back.
- **Map:** The map consists of three layers and a fortress.
- **Fortress:** The Fortress is the players object to defend. The player loses the game, if the fortress has not any health points anymore.
- **Layer:** On a layer are different paths which lead the enemies to the fortress.
- **Tower:** The Player can place towers. The towers attack the enemies and try to stop them before the reach the fortress.
- **Wave:** A Wave contains a specific number of enemies. The wave ends when all enemies are killed or if they are not killed in time and reached the Fortress.
- **Enemy:** The Enemies try to reach the Fortress. If this happens, the fortress will lose health points.

2.2. Use Cases

The following use cases are prioritized from top to bottom. The biggest use cases "Play Game" was split up into smaller use cases to keep it organized and clear, because it's the biggest and most time consuming one.

Name	Details	Priority
1. Play game		High
2. Place tower	UC 2 – 7 are all use cases extending UC 1. Play Game.	High
3. Upgrade tower		Medium
4. Tear down tower		Medium
5. Call next wave		High
6. Pause game		Low
7. Unpause game		Low
8. Create map		High
9. Edit map		Medium
10. Import map		Medium
11. Export map		Medium

UC 1 – Play game

Primary Actor: Player

Stakeholders and Interests:

- Player: Wants a stable framerate with short load times and no crashes to interrupt his experience.

Preconditions: The player has selected a map and started a new game on it.

Post conditions: The player has either defeated all enemy waves and won or his central structure has taken a critical amount of damage and has been destroyed resulting in the player losing the game.

Main success scenario: Ping-pong between actor and system!

1. The player has started a new game and the map is loaded.
2. The player spends his starting currency on building up his defenses.
3. The player clicks on begin, indicating they are done preparing and ready for the first enemy wave.
4. Incoming enemies are destroyed by the defensive structures and the player spends the money gained on new defenses.

Step four repeats itself so long as there are enemies remaining in the current wave and the main structure has not been destroyed.

5. When all enemies of the current wave have been destroyed, there is an indication that the next wave will be incoming soon.
6. The player has a set amount of time to improve his fortifications before the next wave begins automatically.

Steps four through six are repeated while the last wave has not been defeated and the main structure has not been destroyed.

7. The player has defeated the last wave and a message is displayed indicating that they have won the game.
8. The game automatically returns to the main menu after the message has disappeared.

Extensions:

*a. At any time, the game crashes:

The game shuts down and the player must restart the game if he wishes to continue playing. Any game progress will not be saved. Numbered steps!

*b. The player closes the game window:

The current game ends and no game progress is saved.

*c. The Player pauses the game:

The ongoing game is paused, and a menu is brought up allowing the player to leave to leave the game or to resume it.

Special Requirements:

- Windows or Mac computer with Java 8.
- Computer with mouse or a touch display.

Frequency of Occurrence: However often it is initiated by player.

UC 2 – Place tower

Main success scenario:

1. The player selects a tower type to build. The selection is valid if the chosen tower type is unlocked and the players budget is not lower than the costs of the chosen tower type. If the selection is not valid, then the scenario ends here and can be restarted at any time.

2. The player selects a tile on the map to build the tower. The selection is valid if the chosen tile is neither a path tile nor is already occupied by another tower and can fit the tower in if the tower is multilayered (Towers spanning two layers can only be placed on the lower 2 layers. Towers spanning all three layers can only be placed on the lowest layer). If there is no tile available which could be selected, the scenario ends here and can be restarted at any time.
3. The tower gets built on the selected tile and its cost gets subtracted from the player's budget.

UC 3 – Upgrade tower

Main success scenario:

1. The player can upgrade a tower so that the tower makes more damage to the enemies. To upgrade a tower the player needs enough amount of money.
2. By clicking on an existing tower, a menu shows up with different upgrade possibilities like double damage, shoot faster etc. If the player doesn't have enough money for a specific upgrade, the upgrade is still shown in the menu, but grayed out.
3. By clicking on the desired upgrade, the player loses money based on the cost of the upgrade and the tower gets upgraded.
4. The player closes the upgrade menu and is back in action.

UC 4 – Tear down tower

Main Success Scenario:

1. The player clicks on the tower he wants to tear down.
2. The tower menu opens in which he can upgrade or tear down the tower.
3. The player clicks on the tear down button.
4. The tower gets destroyed and the player gets a fractional amount of the money he invested in building the tower.
5. The tower menu gets closed.

UC 5 – Call next wave

Main success scenario:

1. When all enemies are defeated from the previous wave and the player has modified his defense, he can click on the button “Call Next Wave”. This results with the beginning of the next wave.

Alternate Scenario

When the player has defeated the last wave of enemies, the player can’t call a next wave because he has won the game.

UC 6 – Pause game

The player can pause an ongoing game to stop all ongoing actions until resumed.

UC 7 – Unpause game

The player can resume an ongoing game from a paused state to continue playing.

UC 8 – Create map

The player can create custom maps for him to play on.

UC 9 – Edit map

The player can edit his custom maps.

UC 10 – Import map

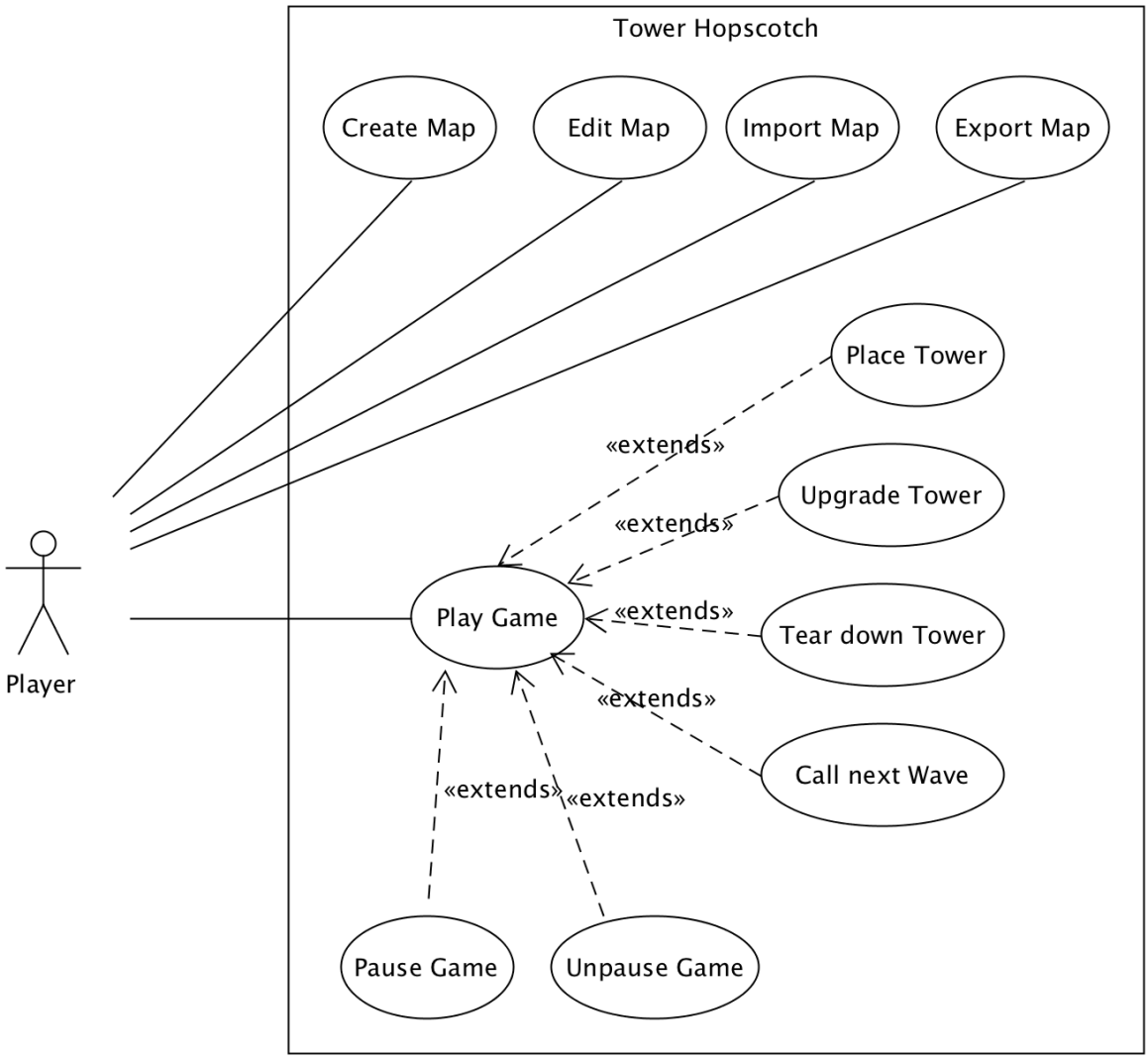
A map can be imported from a chosen directory to be used or edited in the game.

UC 11 – Export map

The player clicks on the “Export Map” button and the program exports a selected map to a chosen file location.

2.3. Use Case Diagram

As briefly mentioned above, most of the use cases are extensions of the primary “Play Game” use case. Everything related to the map on the other hand stands alone.

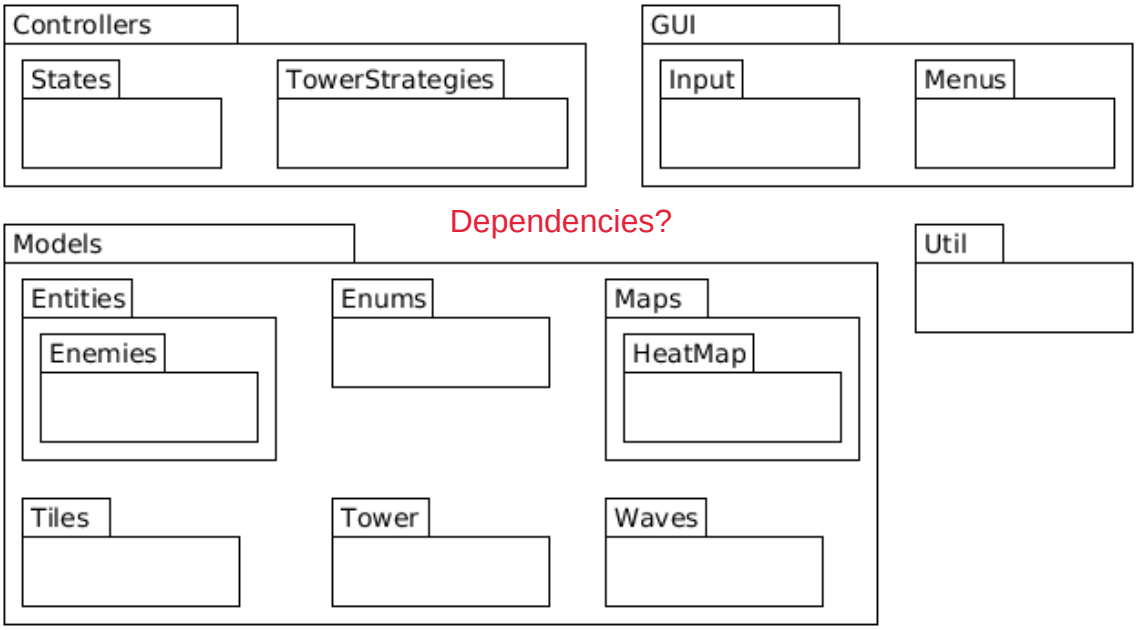


Add the ID to each use case for rerenceing them.

3. Design

3.1. Architecture

A short introduction to what follows would be useful.



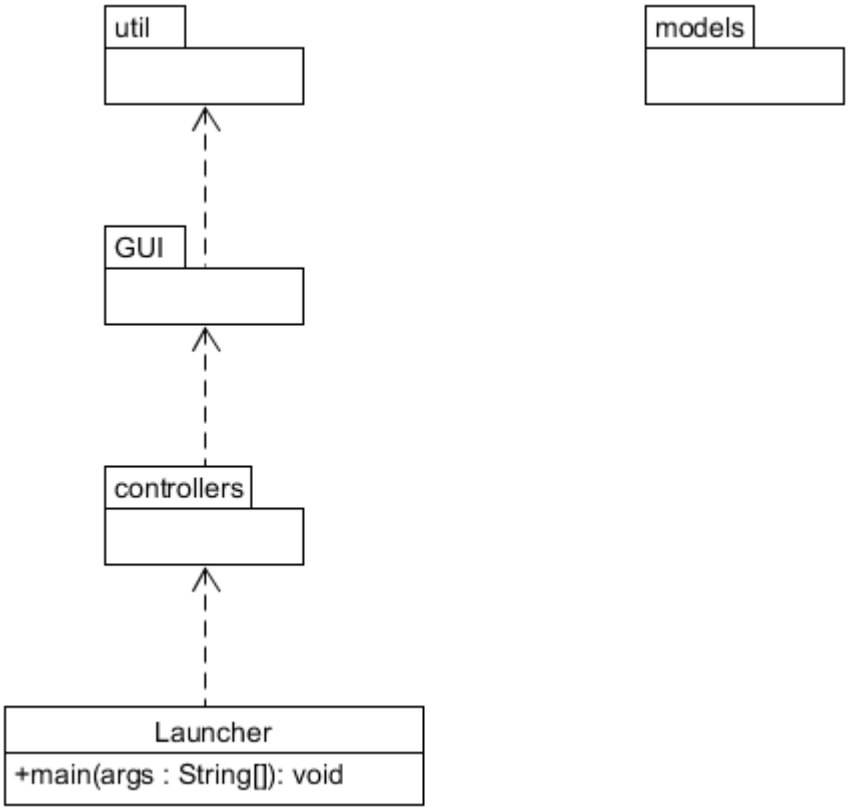
Add a short description for each package.

According to the code, individual model elements have a dependency on the GUI. Is that desired?

3.2. Design Class Diagram

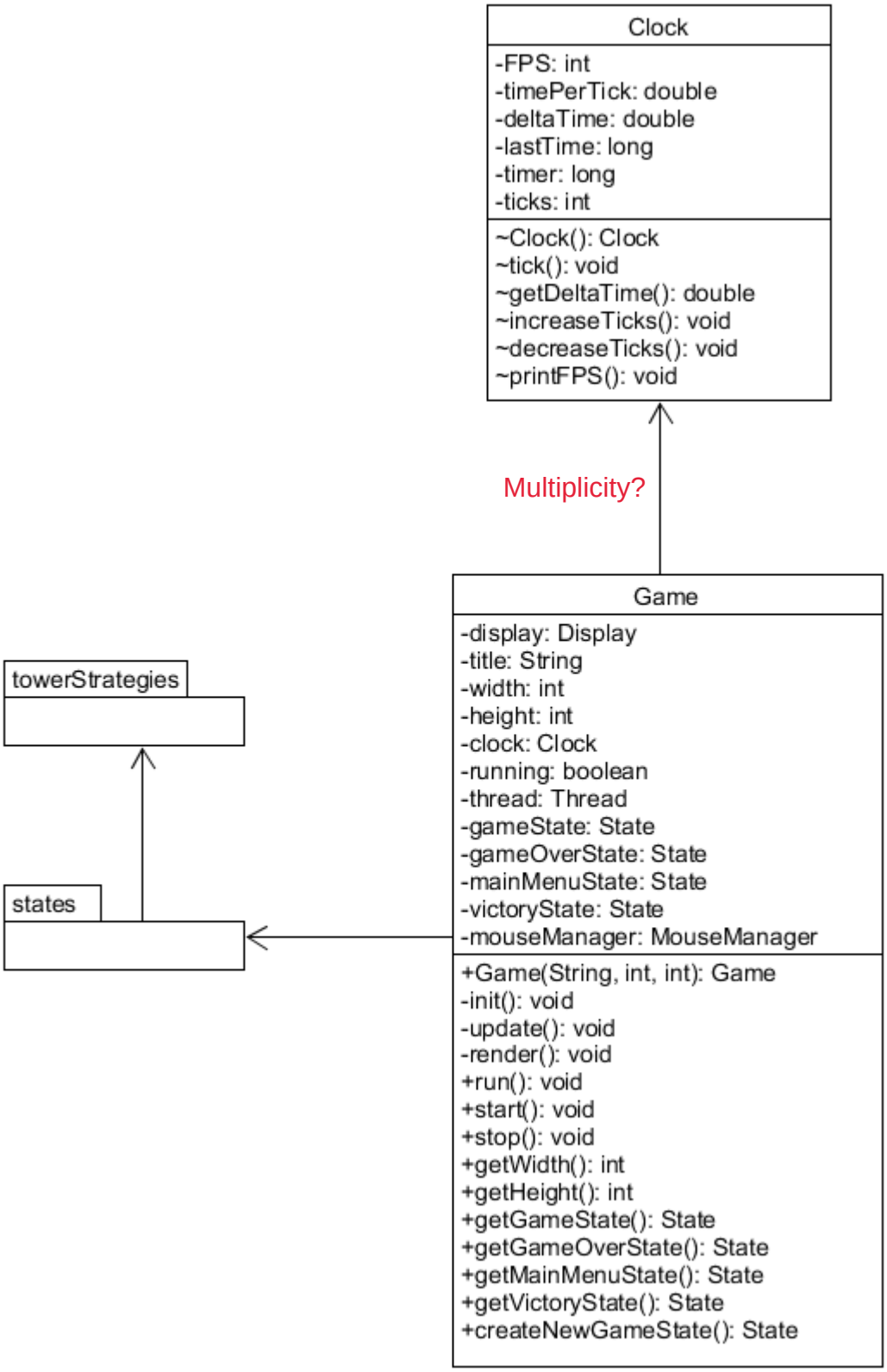
All diagrams show different packages of the program for a better visualization:

Root

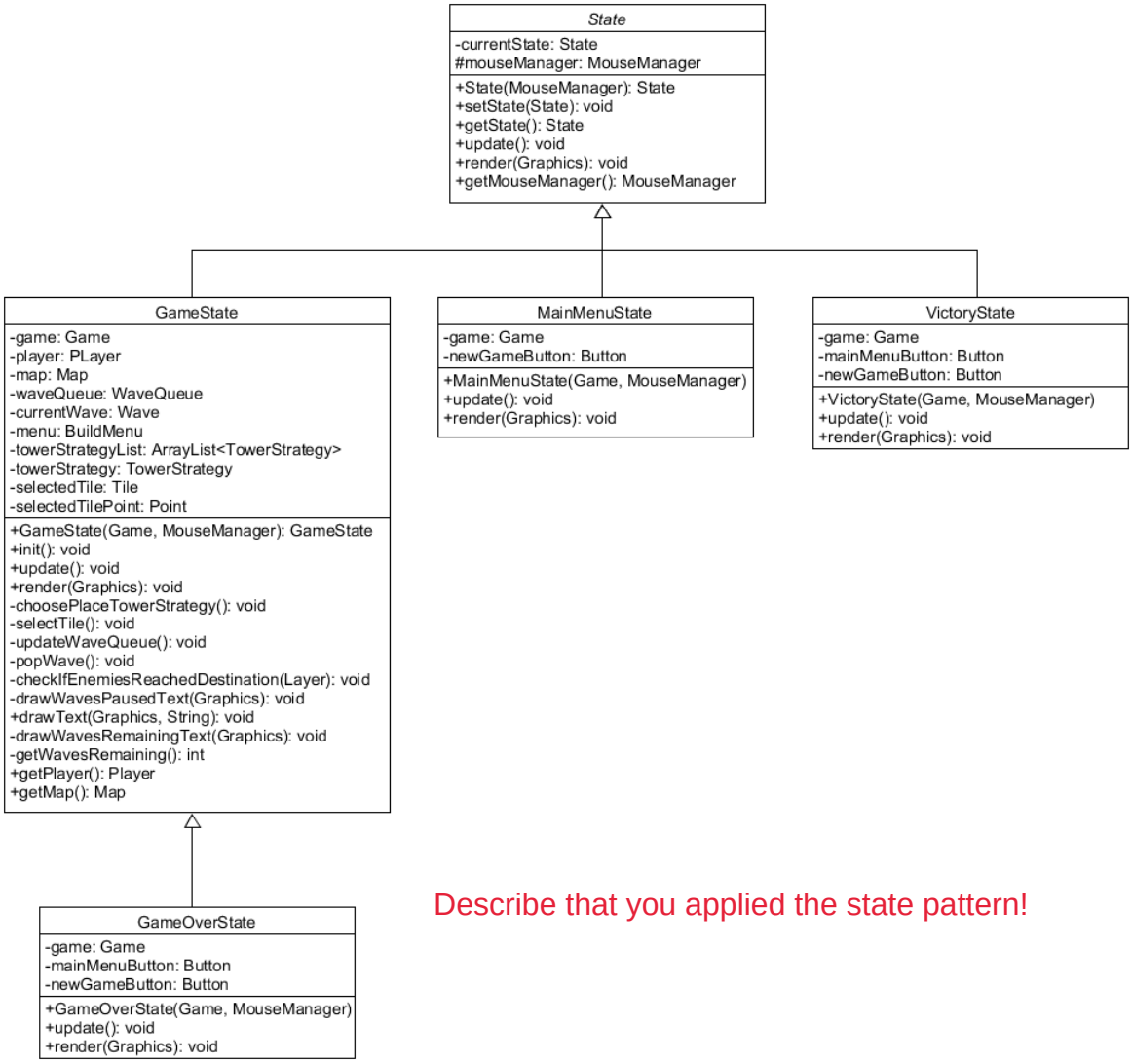


Each diagram should have a short description!

Controllers

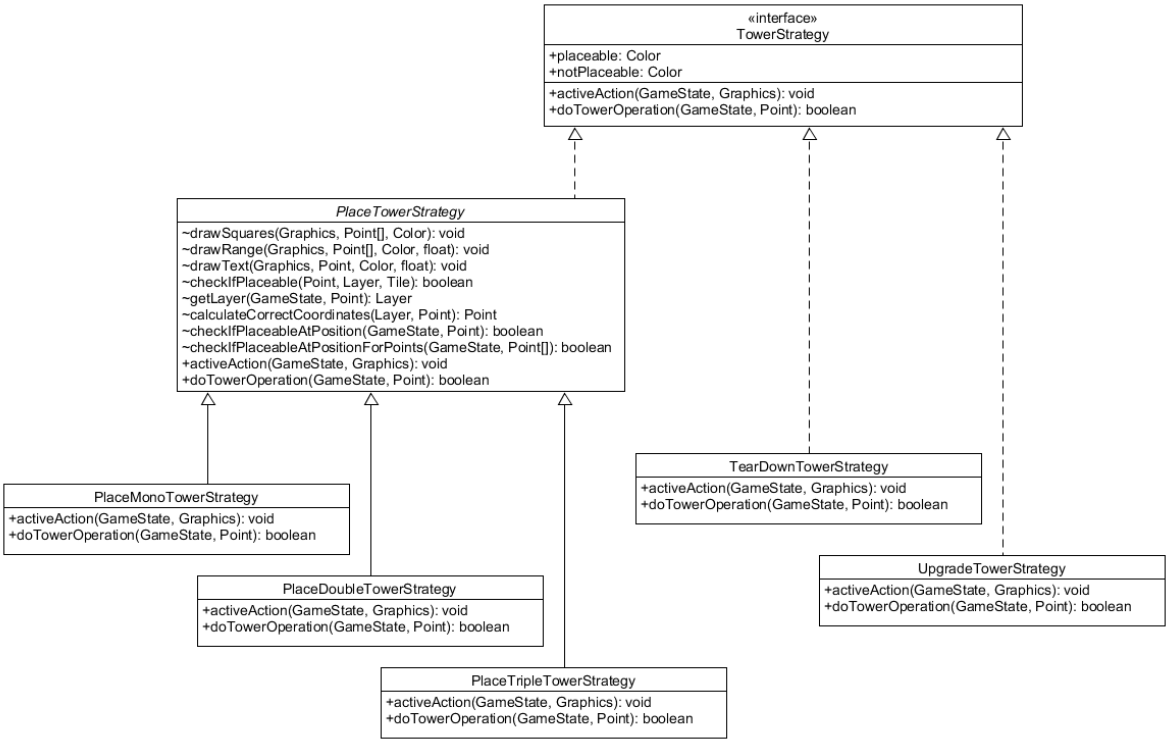


ControllersStates

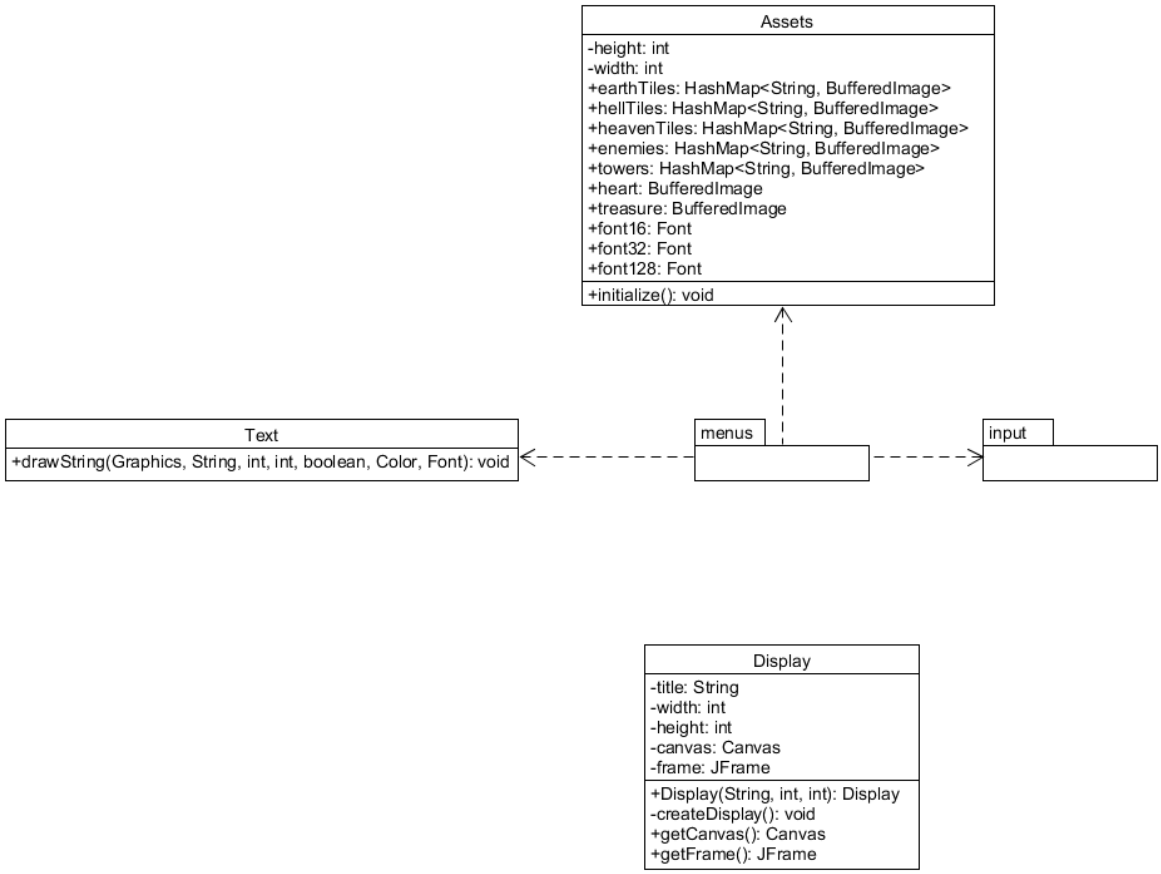


Describe that you applied the state pattern!

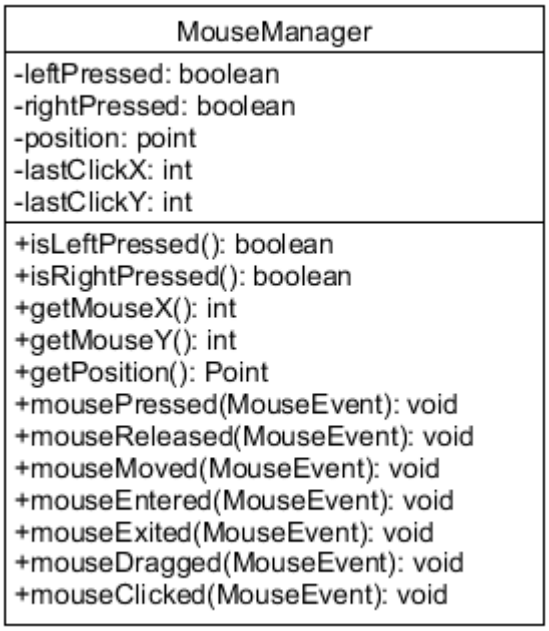
ControllersTowerStrategies



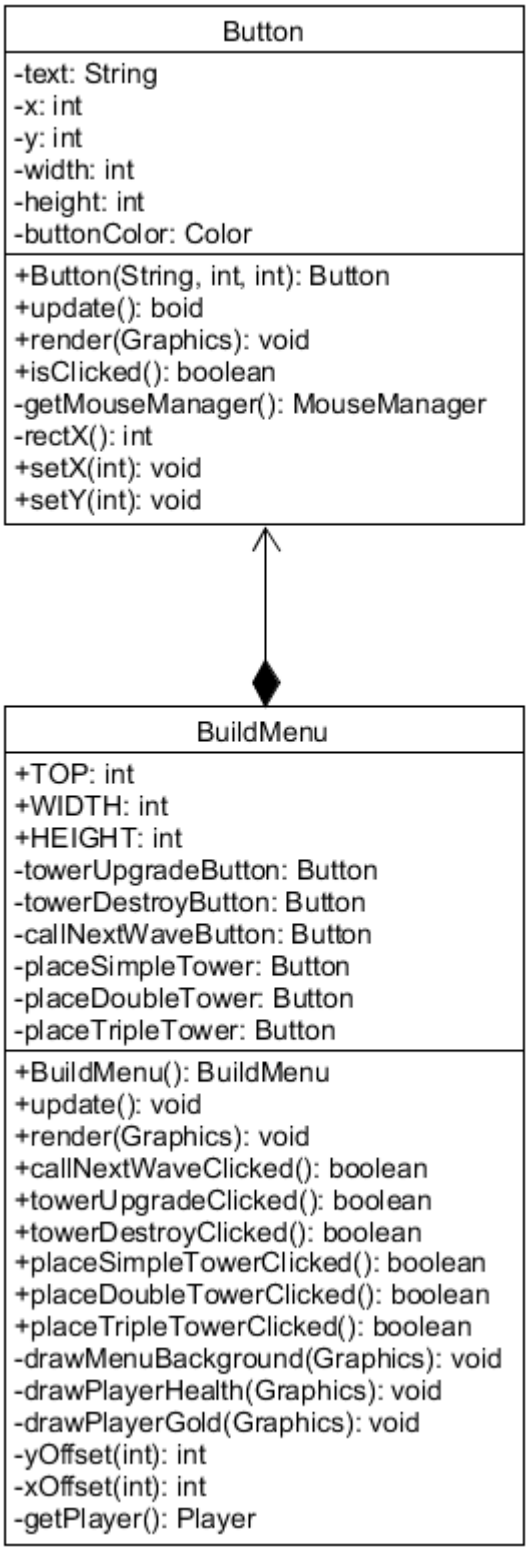
GUI



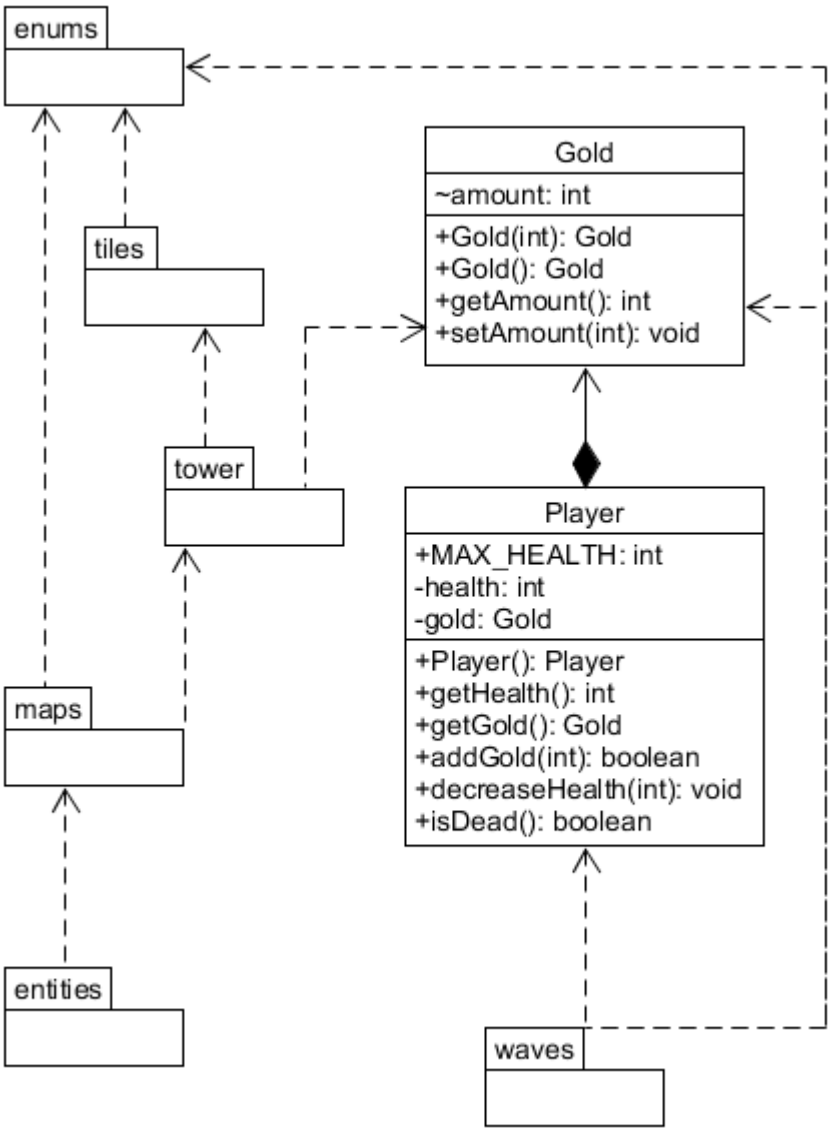
GUIInput



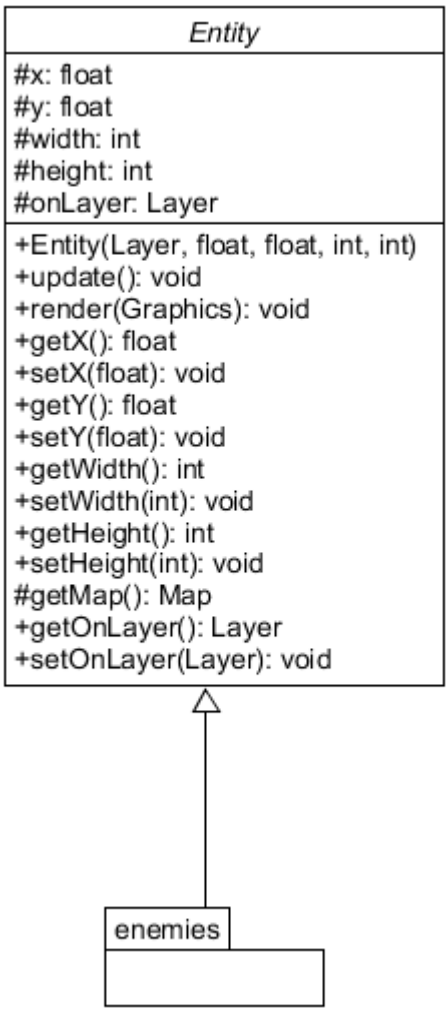
GUIMenus


Button
-text: String
-x: int
-y: int
-width: int
-height: int
-buttonColor: Color
+Button(String, int, int): Button
+update(): boid
+render(Graphics): void
+isClicked(): boolean
-getMouseManager(): MouseManager
-rectX(): int
+setX(int): void
+setY(int): void
BuildMenu
+TOP: int
+WIDTH: int
+HEIGHT: int
-towerUpgradeButton: Button
-towerDestroyButton: Button
-callNextWaveButton: Button
-placeSimpleTower: Button
-placeDoubleTower: Button
-placeTripleTower: Button
+BuildMenu(): BuildMenu
+update(): void
+render(Graphics): void
+callNextWaveClicked(): boolean
+towerUpgradeClicked(): boolean
+towerDestroyClicked(): boolean
+placeSimpleTowerClicked(): boolean
+placeDoubleTowerClicked(): boolean
+placeTripleTowerClicked(): boolean
-drawMenuBackground(Graphics): void
-drawPlayerHealth(Graphics): void
-drawPlayerGold(Graphics): void
-yOffset(int): int
-xOffset(int): int
-getPlayer(): Player

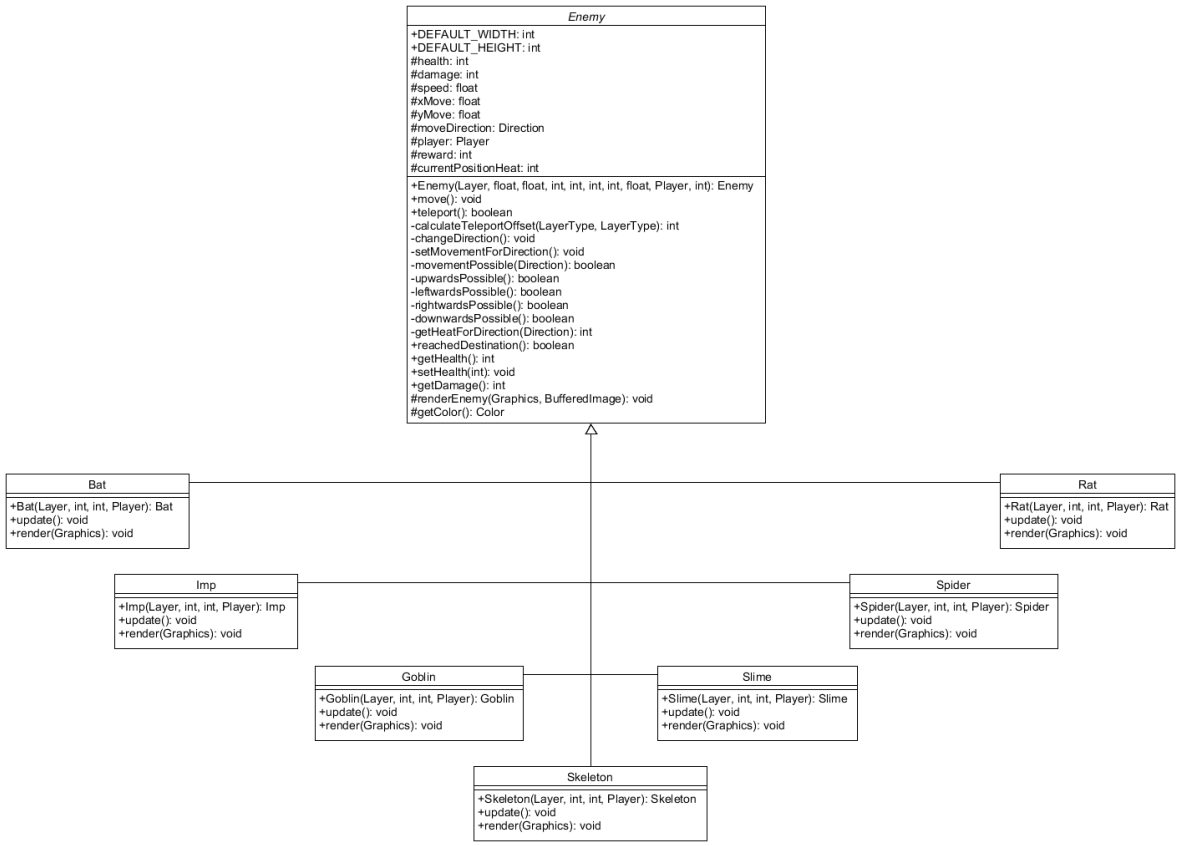
Models


enums
tiles
tower
maps
entities
waves
Gold
~amount: int
+Gold(int): Gold
+Gold(): Gold
+getAmount(): int
+setAmount(int): void
Player
+MAX_HEALTH: int
-health: int
-gold: Gold
+Player(): Player
+getHealth(): int
+getGold(): Gold
+addGold(int): boolean
+decreaseHealth(int): void
+isDead(): boolean

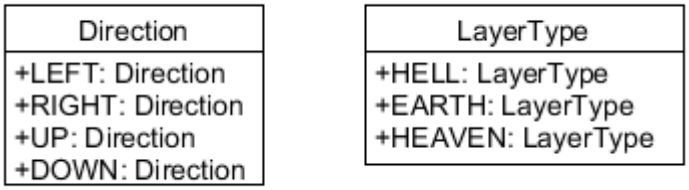
ModelsEntities



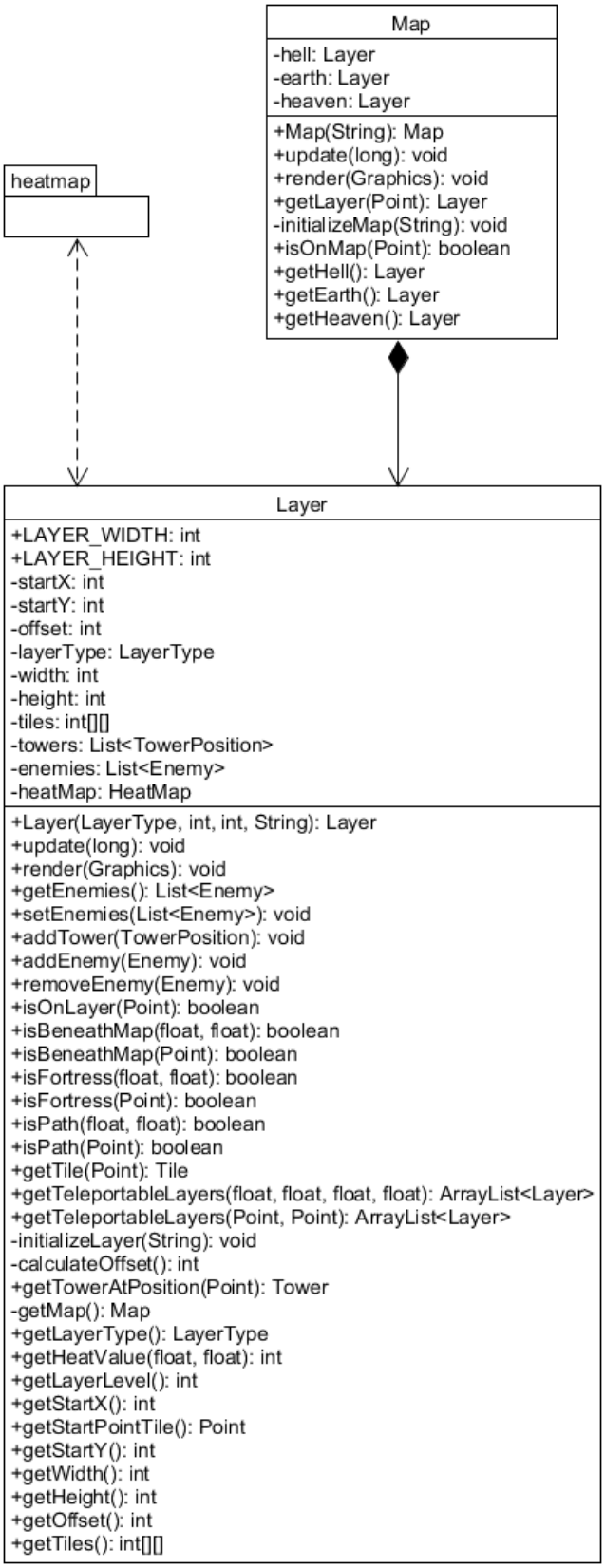
ModelsEntitiesEnemies



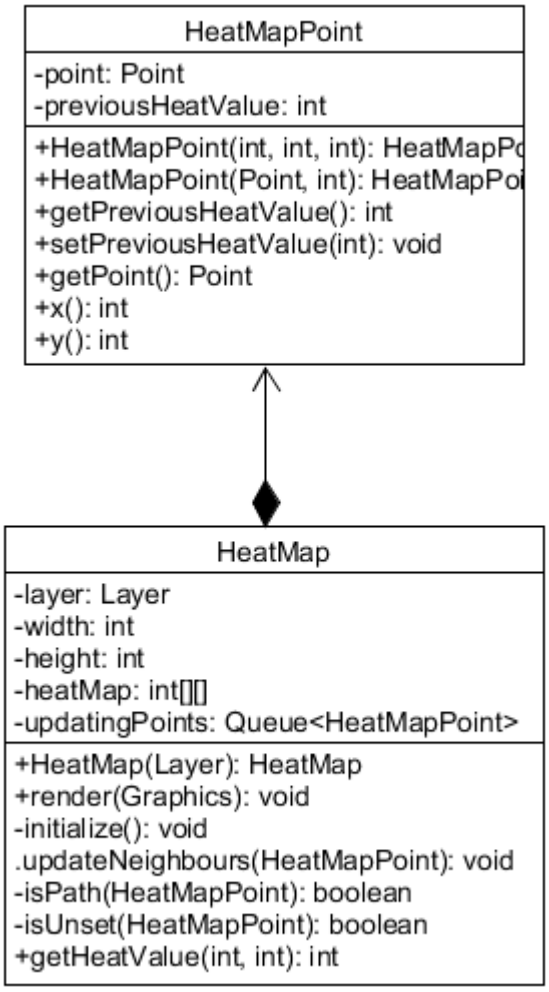
ModelsEnums



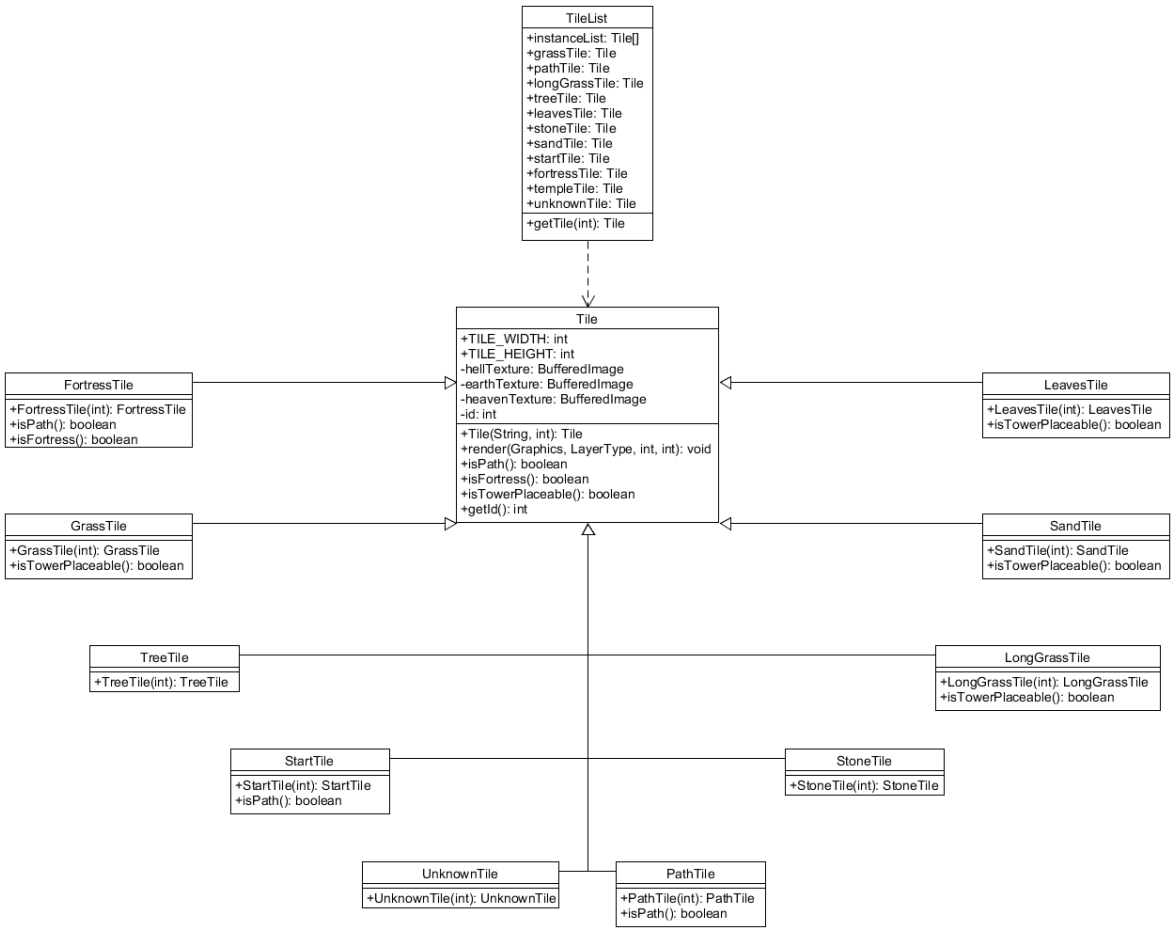
ModelsMaps


Map
-hell: Layer
-earth: Layer
-heaven: Layer
+Map(String): Map
+update(long): void
+render(Graphics): void
+getLayer(Point): Layer
-initializeMap(String): void
+isOnMap(Point): boolean
+getHell(): Layer
+getEarth(): Layer
+getHeaven(): Layer
heatmap
Layer
+LAYER_WIDTH: int
+LAYER_HEIGHT: int
-startX: int
-startY: int
-offset: int
-layerType: LayerType
-width: int
-height: int
-tiles: int[][]
-towers: List<TowerPosition>
-enemies: List<Enemy>
-heatMap: HeatMap
+Layer(LayerType, int, int, String): Layer
+update(long): void
+render(Graphics): void
+getEnemies(): List<Enemy>
+setEnemies(List<Enemy>): void
+addTower(TowerPosition): void
+addEnemy(Enemy): void
+removeEnemy(Enemy): void
+isOnLayer(Point): boolean
+isBeneathMap(float, float): boolean
+isBeneathMap(Point): boolean
+isFortress(float, float): boolean
+isFortress(Point): boolean
+isPath(float, float): boolean
+isPath(Point): boolean
+getTile(Point): Tile
+getTeleportableLayers(float, float, float, float): ArrayList<Layer>
+getTeleportableLayers(Point, Point): ArrayList<Layer>
-initializeLayer(String): void
-calculateOffset(): int
+getTowerAtPosition(Point): Tower
-getMap(): Map
+getLayerType(): LayerType
+getHeatValue(float, float): int
+getLayerLevel(): int
+getStartX(): int
+getStartPointTile(): Point
+getStartY(): int
+getWidth(): int
+getHeight(): int
+getOffset(): int
+getTiles(): int[][]

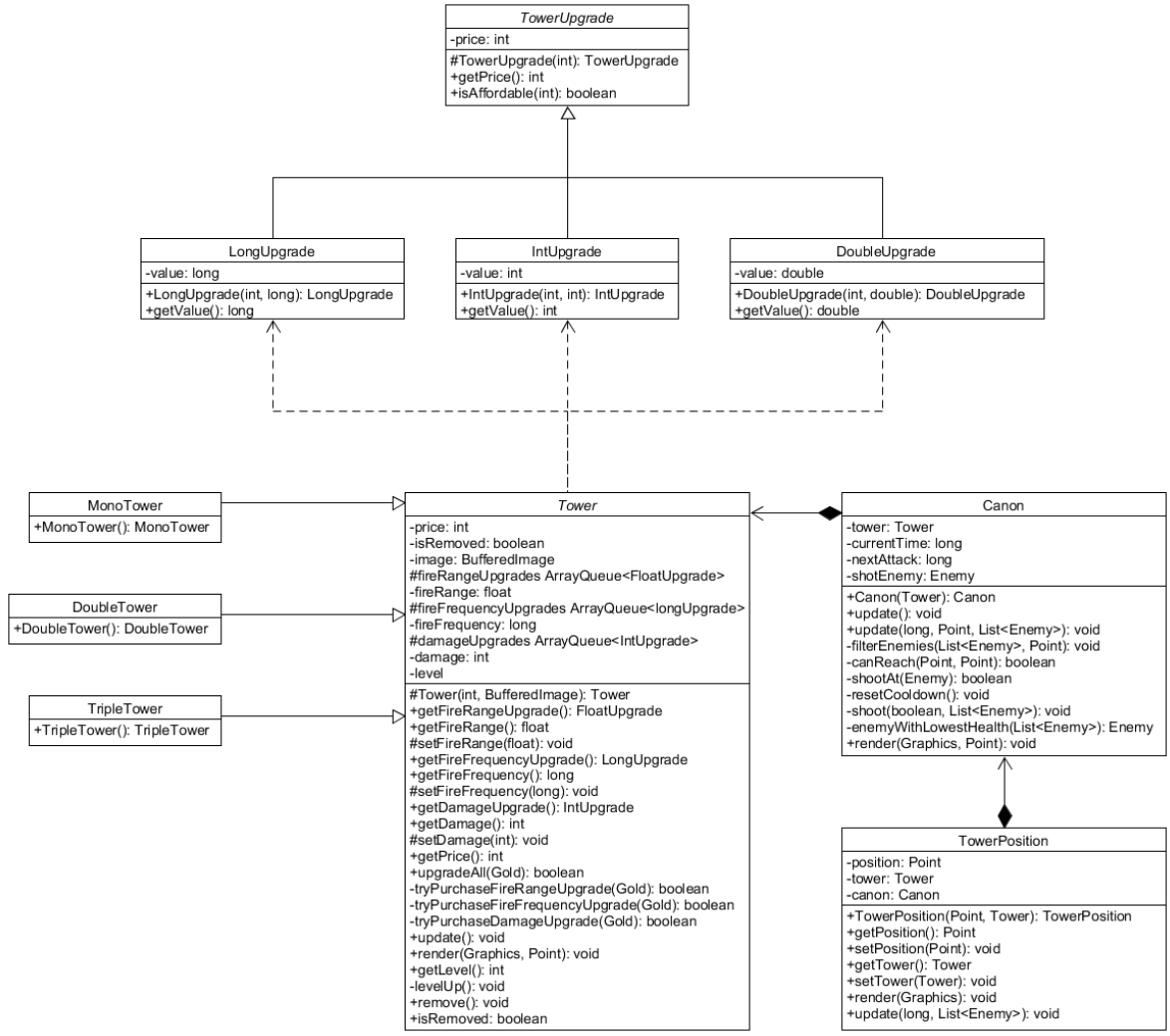
ModelsMapsHeatMap



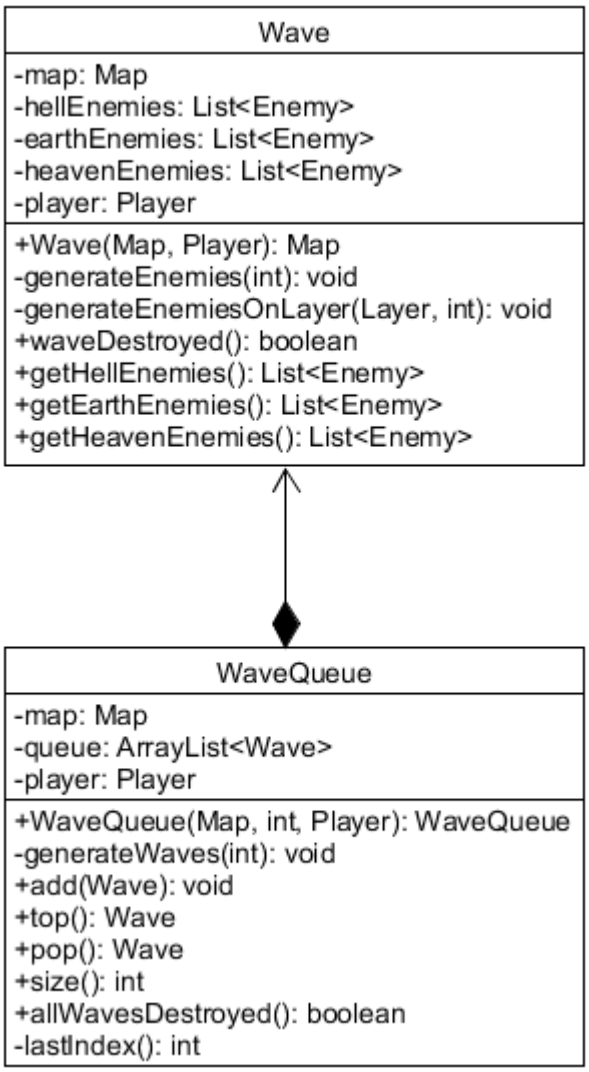
ModelsTiles



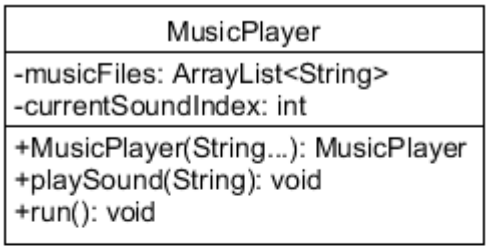
ModelsTower

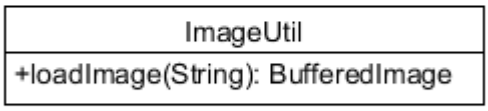


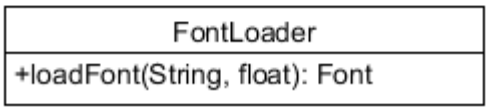
ModelsWaves



Util

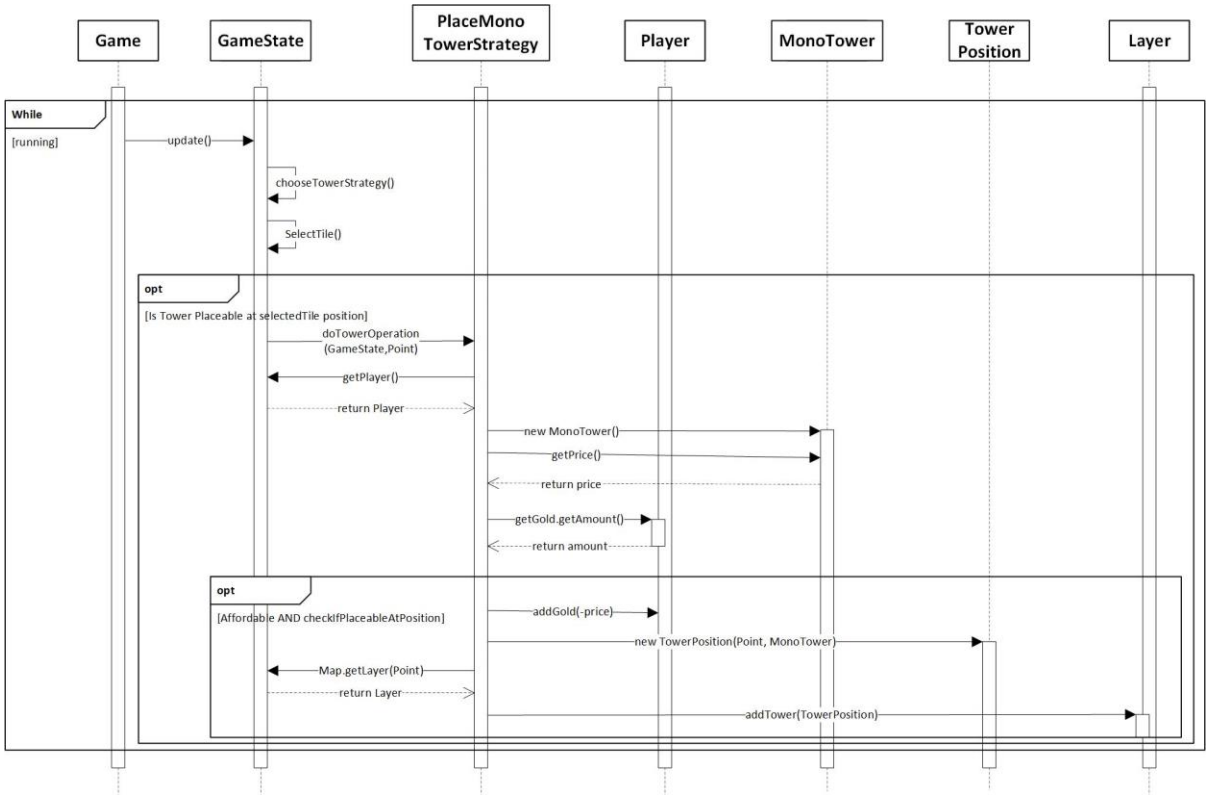

MusicPlayer
-musicFiles: ArrayList<String>
-currentSoundIndex: int
+MusicPlayer(String...): MusicPlayer
+playSound(String): void
+run(): void


ImageUtil
+loadImage(String): BufferedImage

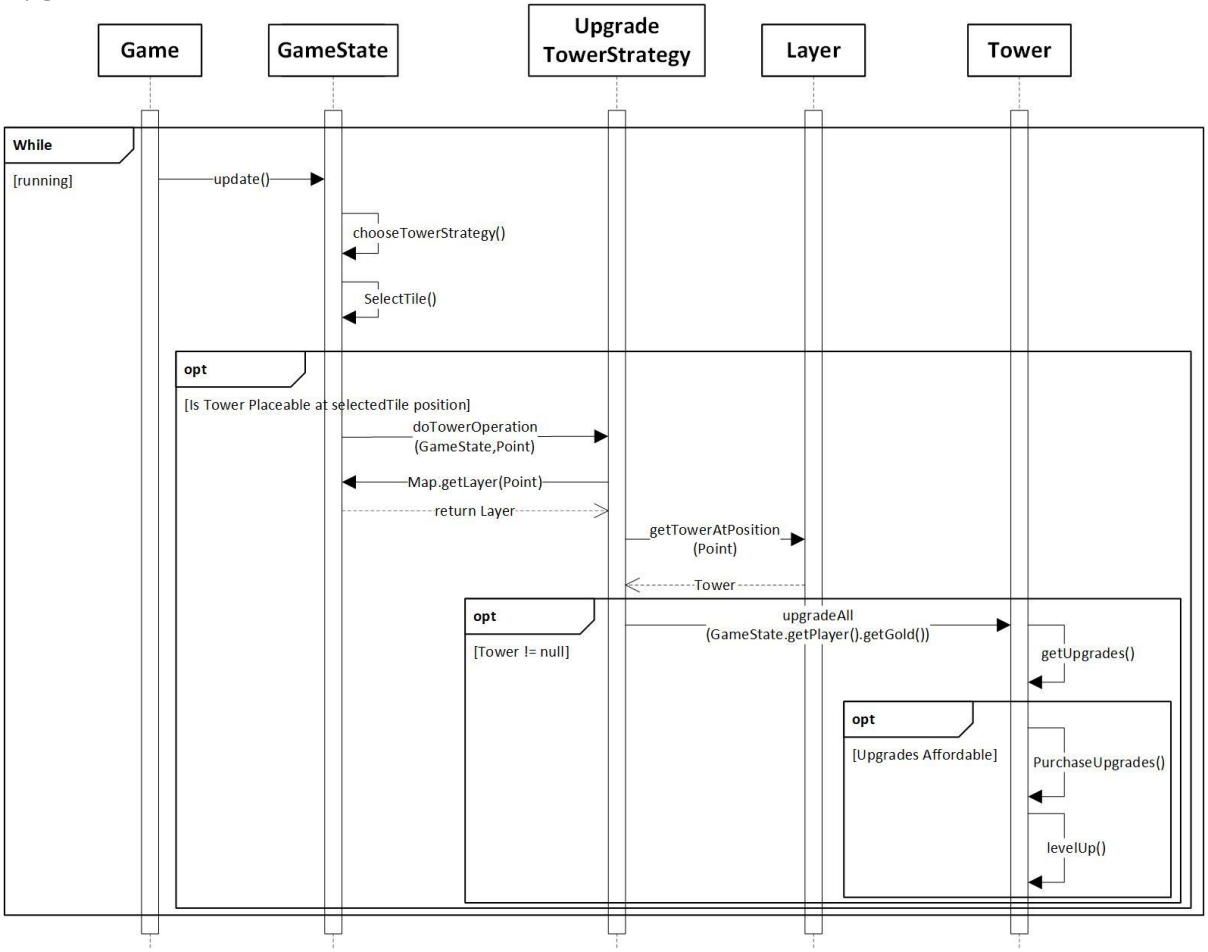

FontLoader
+loadFont(String, float): Font

3.3. Interaction Diagram

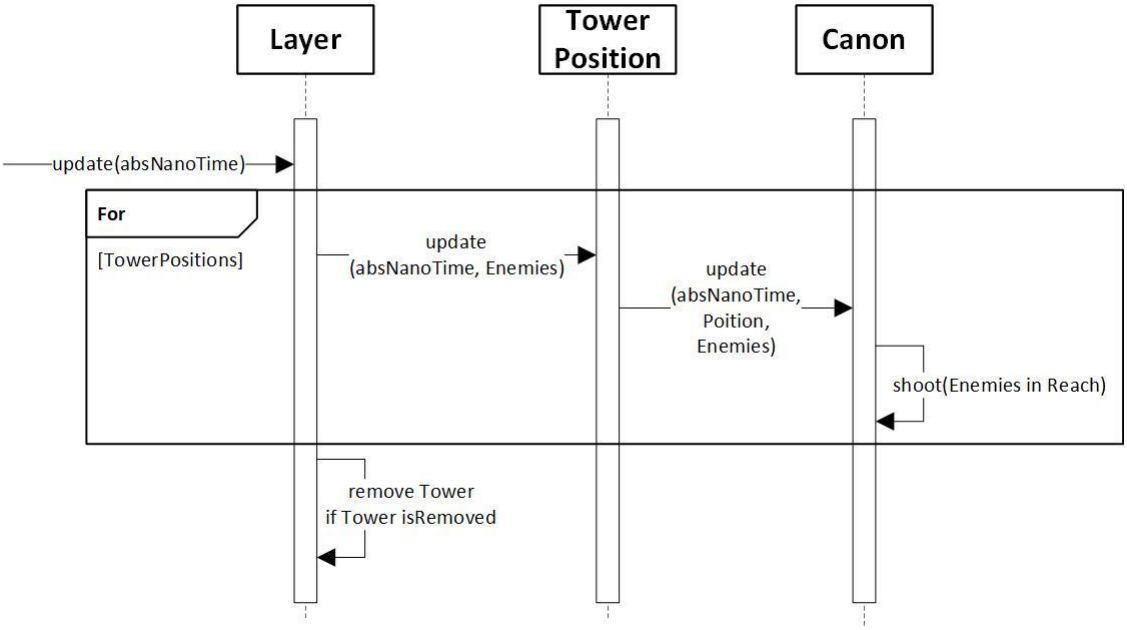
Place Mono Tower



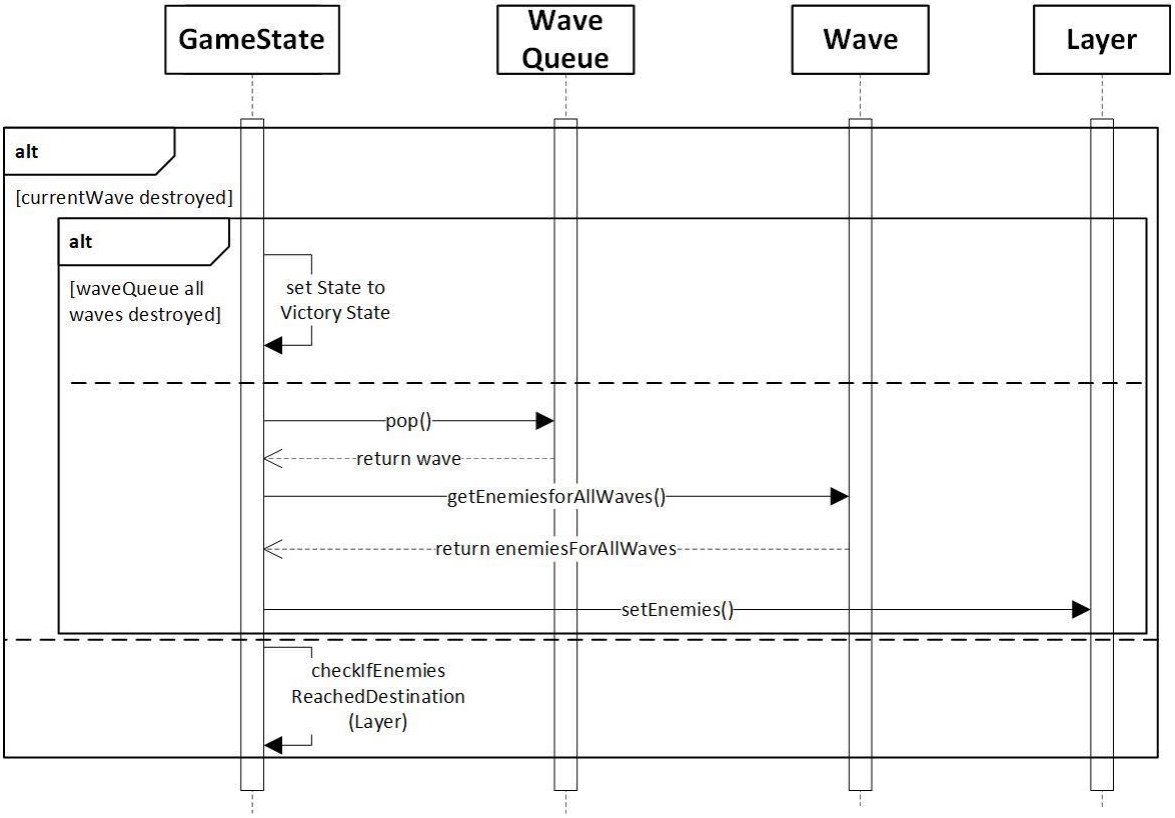
Upgrade Tower



Shoot Enemies



Call Next Wave



3.4. Design Decisions

We used especially two specific patterns excessively to our benefit, namely the state and the strategy pattern. Those patterns were mainly used to increase the readability of the code and of course to create low coupling and high cohesion.

Game States

Since there are different states in the game, it was only logical to break them down into different classes.

- State
- MainMenuState
- GameState
- GameOverState
- VictoryState

Tower Strategy

There are a lot of different interactions with towers in our game. It seemed natural to break them down into different classes to simplify the complexity.

- TowerStrategy(Interface)
- PlaceTowerStrategy
- PlaceMonoTowerStrategy
- PlaceDoubleTowerStrategy
- PlaceTripleTowerStrategy
- UpgradeTowerStrategy
- TearDownTowerStrategy

It would be better to explain the design patterns directly on the corresponding DCD.

4. Implementation

4.1. Tests

A short explanation of the test strategy in the project - when and which tests were done - would round off the test description.

Main Menu - System Tests				
#	**Initial State**	**Event**	**Expected Result**	**Result**
1	The player has just started up the game and is in the main menu.	The player clicks the “Start Game” button.	A new game starts and the player has full hit points while the starting amount of gold in the treasury.	
2	The player has either lost or won a game and has returned to the main menu from there.	The player clicks the “Start Game” button.	A new game starts and the player has full hit points while the starting amount of gold in the treasury.	
Victory Screen - System Tests				
#	**Initial State**	**Event**	**Expected Result**	**Result**
3	The player has won a game and is currently in the victory screen.	The player clicks the “Main Menu” button.	The screen changes to the main menu.	
4	The player has won a game and is currently in the victory screen.	The player clicks the “Start Game” button.	A new game starts and the player has full hit points while the starting amount of gold in the treasury.	
Game Over Screen - System Tests				
#	**Initial State**	**Event**	**Expected Result**	**Result**
5	The player has lost a game and is currently in the defeat screen.	The player clicks the “Main Menu” button.	The screen changes to the main menu.	
6	The player has lost a game and is currently in the defeat screen.	The player clicks the “Start Game” button.	A new game starts and the player has full hit points while the starting amount of gold in the treasury.	

In-Game - System Tests				
#	**Initial State**	**Event**	**Expected Result**	**Result**
7	The player has just started a new game from the main menu.	Clicks on any tower in the bottom menu bar.	A message is displayed that the player can now place the tower on the map and there are green or red indicator whether or not a tower can be built at the mouse's current position.	
8	Initial state and event from system test #7.	The player left clicks on the map at a valid build spot.	The tower is built and has level 1. The correct amount is deducted from the players treasury.	
9	The player has built a tower on the map.	The player clicks the "Upgrade Tower" button in the menu and left clicks the tower.	The tower now has level 2 and the upgrade cost is deducted from the players treasury, unless it would drop below zero.	
10	The player has built a tower on the map.	The player clicks the "Destroy Tower" button in the menu and left clicks the tower.	The tower is removed from the map and half of the build cost is restored to the players treasury.	
11	The player is performing any of the three tower actions (build, upgrade or destroy)	The player clicks his right mouse button anywhere in the screen.	The action is cancelled.	
12	The waves are paused.	The player clicks the "Call Next Wave" button.	The next wave of enemies' spawns and the walk towards the top end.	

13	An enemy wave is currently ongoing.	The player clicks the “Call Next Wave” button.	Nothing happens.	
14	An enemy wave is currently ongoing and there are towers on the map.	The first enemy walks into any towers range.	The tower opens fire on that enemy.	
15	An enemy wave is currently ongoing, there are towers on the map and there is only one enemy remaining.	Some tower delivers the killing blow to the last enemy.	The enemy is removed from the map, the “Waves remaining” counter at the top right is decreased by one and the waves are paused.	
16	There is only one enemy remaining of the last enemy wave and there are towers on the map.	Some tower delivers the killing blow to the last enemy.	The player enters the victory screen.	
17	An enemy wave is ongoing.	An enemy reaches the top of its layer.	The player hit points are reduced by whatever the enemy’s damage attribute is.	
18	An enemy wave is ongoing.	An enemy reaches the top of its layer and the players hit points drop to zero or below.	The player enters the game over screen.	
19	An enemy wave is ongoing.	An enemy reaches a point on its layer where there is also a path tile on another layer.	The enemy has a small chance of teleporting to the corresponding tile on the other layer in question.	

Balancing Tests

Several balancing tests and tweaks had to be done on different components of the game to create a meaningful game environment. Balancing tests were done with the following things in mind:

1. Is there enough gold for the player, without making the game too easy?
2. Are enemies walking too fast for the player to interact?
3. Are enemies walking to slowly, to the point where it becomes boring?
4. Are towers doing enough damage to the enemies?
5. Are towers shooting fast enough to keep the player visually entertained?
6. Do enemies have enough health in comparison to the damage output of the towers?
7. Are enemies doing a meaningful amount of damage to the castle to keep the player engaged?

4.2. Code

The complete source code and Java documentation of the current prototype can be found in a zip compressed file on OLAT as well as under the following link:
https://github.engineering.zhaw.ch/emberrap/PSIT3-HS17-IT16aWIN-5

4.3. Installation

A Java executable JAR file is also located in the zip compressed file on OLAT or can be generated in any Java development environment.

5. Results

5.1. Goal Summary

The current prototype is a fully functional and playable tower defense game. Enemies have a smart pathing logic implemented to traverse different maps and teleport between layers. There are different instances of towers that can operate on multiple layers at once and target enemies that are in range. Tower building, upgrading and destruction is easily achievable via the in-game user menu.

The entire game cycle stands too; from the main menu through the game all the way to the victory or game over screens depending on the players performance.

5.2. Remaining points for prototype goals

- While towers can be upgraded and its current level is always visible, it is hard to keep track of its exact statistics (range, firepower, etc.) from the players perspective. It would make sense to create a small, non-intrusive popup that appears whenever the player selects a tower on the map. Here the player can find all its current values.
- Currently pausing and unpausing the game at will is not possible. The game stops sending new enemies after a wave has been destroyed, but during waves the player cannot pause. It would make sense to have a small button for pausing and unpausing the game at will.
- Modification of the map currently requires the user to edit the map data file in the jar file or run the game through the development environment. It would make sense to have a file import functionality that allows any player to import new maps at any time.

5.3. Short-Term Future Improvements

- Compose a fitting soundtrack for the games art style and add sound effects for different actions.
- Add animations for enemy movement and teleportation.
- Add improved animations for towers attacking enemies.
- Add an in-game map editor to allow for easy map creating without the player having to deal with text files.
- Add more enemies with unique statistics and behaviors.
- Add more tower types to counter specific enemies.

5.4. Long-Term Future Improvements

- Integrate with Steam to allow for achievements, high score sharing and monetization.
- More game modes such as an endless mode where the player tries to survive as long as possible.
- Move to a semi 3D user interface as planned in the vision and have the layers stacked instead of aligned horizontally.

6. Appendix

6.1. Project Management

6.1.3. Summary project management processes

After having roughly outlined all iterations and generally assigned which tasks should be completed by when, the decision was met to relieve the team leader Raphael Emberger by assigning several management duties to Nicolas Eckhart. These included managing the time tracking document and planning each iteration specifically so the tasks could be assigned.

Each Tuesday the team would assembly to gather a list of all tasks that needed to be completed in the coming iteration. These tasks would then be estimated and assigned to a developer. After this was completed the developers would continue on whatever they were currently working on, unless there was another matter to be discussed.
This could be anything from having difficulty understanding someone else's code or wanting feedback on a newly created design class diagram proposal.

All smaller matters we discussed on the team's slack channel.

6.1.3. Time expenditures breakdown

Project Management Summary – Tower Hopscotch		
Inception Phase [E: 26hrs / S: 27.5hrs]	25.09.2017	06.10.2017
Iteration #1	**Estimation:** 26hrs	**Spent:** 27.5hrs
Elaboration Phase [E: 111hrs / S: 103hrs]	09.10.2017	17.11.2017
Iteration #2	**Estimation:** 34hrs	**Spent:** 31.5hrs
Iteration #3	**Estimation:** 34hrs	**Spent:** 27.5hrs
Iteration #4	**Estimation:** 43hrs	**Spent:** 43hrs
Construction Phase [E: 172hrs / S: 155.5hrs]	20.11.2017	15.12.2017
Iteration #5	**Estimation:** 74hrs	**Spent:** 72.5hrs
Iteration #6	**Estimation:** 98hrs	**Spent:** 83hrs
Iteration #7 **–** Buffer week 18.12.2017 – 22.12.2017		

Overall summary?

As becomes clear from the table above the team was very accurate when estimating the specific tasks. In some iterations we slightly overstepped our goal but in others we didn't need all the time set aside for the tasks. Only in the last iteration we saved a lot of time which is because we set high estimations for the manual and final report out of caution.

6.2. Bibliography

[1] E. McDonald, “NEWZOO,” 20 April 2017. [Online]. Available: https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/.

[2] EA. [Online]. Available: https://www.ea.com/en-gb/games/plants-vs-zombies/plants-vs-zombies-2. [Zugriff am 02 10 2017].

6.3. Glossary

Term	Definition
Fortress	The players central structure, the defense of which is the games main objective. Also referred to as Castle.
Tower	Any of a variety of defensive or offensive building created by the player to hinder or destroy incoming enemies.
Wave	A wave refers to a group of enemies. A game encompasses multiple waves that need to be defeated.
Layer	Each map has three layers that simultaneously spawn incoming enemies that may jump between these layers.
Treasury (Budget)	The amount of gold the player has at any given time. Also referred to as budget.
Gold	The currency used in Tower Hopscotch. Gold is obtained by destroying enemies and can be spent on towers and upgrades.
Hit points	Hit points refer to the amount of health an enemy or the players fortress has.