CS 319 - Object-Oriented Software Engineering

Analysis Report

Medieval Tower Defense

<u>Group 1A</u>

H.Buğra Aydın

Abdullah Mahir Özer

Deniz Başaran

1. Introduction

Medieval Tower Defense is a basic tower defense type video game we decided to develop. There are lots of different tower defense games with different objectives. The main purpose of tower defense games is to prevent enemies from reaching a certain point or destroying a base at that point, by marching along a predetermined path. In order to achieve this, the player sets up a defense of their own design by using the tools provided to them by the system. This is usually done by building towers with resources, customizing them in a way of their choosing according to their strategy.

The game we were influenced by is following game mods from the game Warcraft III.
https://www.epicwar.com/maps/159097/
http://www.moddb.com/mods/element-tower-defense/news/element-td-40-released
Medieval Tower Defense (MTD) will be developed with different features. In MTD we will add new types of waves like boss fights and various difficulty settings.
Classic tower defense game rules will apply. The player will face fifty waves of enemies with the defense they built and prevent them from reaching to the exit point. The player will be able to face the next wave after cleaning wave that is coming. The game is designed to be reusable, since the players will want to defeat the game and obtain a spot in the high score list.
The game will be a desktop application and it will be controlled by a mouse.

2. Overview

MTD is a medieval themed tower defense game. All tools, waves and bosses will be from medieval times. Player will face a wave with towers built by spending resources provided to them at start. After killing each enemy, player will gain different amount of resources and player can choose to build more towers or upgrade towers they have already built. Goal of MTD is clearing all waves, killing bosses to face next wave and next boss until waves are finished.

2.1 Gameplay

Player will need a mouse to play the game. Keyboard will be available to perform some actions such as entering a nickname for high score screen when the game ends. Player will choose and place tower to the map with mouse. Towers can be placed on top of the areas which are made available to player. These areas will usually be near the path which enemies are walking on. After given time finishes, waves will start to come and player's towers will kill them or fail to kill them and enemies will pass exit. There are 50 waves. Wave difficulty will change with difficulty settings. Player is allowed to miss a certain amount of enemies but if that amount is reached, player will lose and start again from the beginning. Players will be able to customize tower according to their strategies and type of enemies.

Game mods are as follows.

- Normal mod : 50 waves, 5 bosses
- Survival mod : Infinite waves, infinite bosses

2.2 Waves

Game will progress one wave at a time. Meaning second wave will not start before all enemies from first wave are killed or missed. A brief waiting time will be given to player between waves. Player will be able to fast forward the game. A player who has established a fine defense which can hold on for several waves can use fast forward so that waiting time for those waves are shorter.

2.2.1 Boss waves

Bosses will come from one of two lanes in boss waves. Boss waves will be more challenging since they will require unique strategies to defeat.

2.3 Building and Customizing Towers

Player's defense will consist of towers they build. The player will place towers on areas they are allowed to and they will be able to customize them with game progress and resource gain.

2.3.1 Tower Types

There will be different tower types and each type will have their own specifications. Tower specs are as follows.

- Slow Rating
- Range
- Damage (siege, infantry)
- Fire Rate

Each tower type will specialize in different ways. Tower types are a follows.

Tower name	**Range/Attack Frequency**	**Attack Point/Damage Type**
Arrow Tower	Medium/High	Low/single target
Canon Tower	High/Low	Medium/splash damage
Ice Tower	Medium/Medium	low/slow debuff
Oil Tower	Very Low/Low	High/area damage
Poison Tower	Medium/High	Medium/Victim bleeds

Arcane Tower	High/Medium	Very low/Armor reducement
Ballista Tower	Very High/very low	Very High/single target
Mortar Tower	High/low	Medium/single target

2.4 Enemies

Each wave will have types of enemies with their own specs.

- Speed
- Armor
- Health

2.4.1 Enemy Types

Invader Name	**Speed**	**Health**	**Armor**
Footman	Medium	Low	Low
Catapult	Low	Medium	Medium
Knight	Low	Medium	High
Light Cavalry	High	Low	Low
Elephant Rider	Low	High	Medium
Trojan Horse	Low	Very High	High
Chengiz Khan & riders	High	Very High	medium
Battering Ram	Low	High	Very High
Jester	Very High	Medium	Medium

Saint John’s Knigths	Low	High	Very High
Pope	Medium	Very High	Low

3. Requirement Specificaton

3.1 Functional Requirements

3.1.1 Play Game

User will be able to control the game with mouse. The game will start after play game button is pushed.

Player will choose the game mode, either normal game mode or survival game mode. Survival game mode will consist unlimited amounts of waves, but still a limited amount of health points. Player will try to survive as long as possible, facing incoming enemies that gets stronger with each wave.

Game map will be constructed randomly. The isometric game map that includes a path for AI invaders and free sites to build towers will be loaded. The AI invaders will be on the path to the final location and this final location is the keep that player has to protect with the towers built. In addition, any invader who manages to trespass the keep will cost the player a health point and player will have 50 health points at total. There will be various types of towers with different characteristics.

Towers can be upgraded for better attack points, and better crowd control ability, also new towers can be purchased with sufficient amount of coin, which can be gathered by eliminating invaders with existing towers. Since Medieval-Tower-Defense is a wave-based game, there will be a number of 20 waves. At each 3 or 5 level player will face with a boss invader, which is considerably harder to beat, yet generates a lot of coins when defeated.

There will also be different types of invaders that have varying attributes of speed, health, and armor. There is another list available that shows the invader types. First 5 are minions and remainder 7 are boss units, which are more challenging special units.

Player will be able to place traps on the enemy road. Setting up these traps will damage enemies or give them debuffs.

3.1.2 Settings

Player will be able to change the game difficulty and music level. In each time when the game application is started, volume of the gameplay and music sounds, and visual quality are predefined in default settings. In order to change these default settings, user may enter the settings page from the main menu and adjust any two of the sound types with the two separate slide bars as desired.

3.1.3 High Scores

Game will store the highscores according to the game progression and time. Through High Scores bar user will be able to see the list of the highest scores. User's own score and other users with top scores will be visible in High Scores section. In this case, there will be a more competitive environment which means more game activity. The high score list will be implemented using a database. It is required to update and store this list.

3.1.4 Pause Button

The user can pause the game in progress any time desired by pressing pause button. Game time will be frozen as long as the pause button is pressed again. Button will be located at the top left corner during gameplay. If the player turns back to the main, game will be auto-paused, and player will have to click continue game in order to proceed more in the current game session. User will lose the achieved progress if application's closed during pause stance.

3.1.5 Help and Information

This section serves as a manual which represents the overall game concepts. The user can get information about: map pattern, game purpose, health, resource gathering, invader types/attributes, and tower types/attributes. Help document is accessed through the main menu, or anytime gameplay is in progress. There may also be a subsection in which game tips can be accessed.

Medieval Tower Defense game will have historical figures such as Chengiz Khan and St. John's Knights in it. Player's will be able to read information about historical characters and technologies from the information screen(tech tree).

3.1.6 Credits

The User may visualize the list of developers by pushing the credits bar of the main menu. Contact information for communicating the developers will be available as well. Suggestions, comments, and bug reports may be sent directly to the developers with information available in this section.

3.1.7 Exit

Exit button is pretty self-explanatory. The application will be closed completely when the button's pushed. Paused game session will be lost and settings will be set on the default mode in the next game launch of application.

3.2 Non-Functional Requirements

• Game graphics will be designed in order to be appealing. There will be no flickering and movement of the units and the projectiles will be smooth. Images of graphical objects will be chosen to give the players better gameplay experience.

• Control mechanisms of the game will have low response time, which enables users to play with minimal delay.

• The code will be efficient, allowing for no framerate drops during gameplay.

• The game will be readable, the player should understand what to do as long as they enter the game.

• The game art should not be too confusing to make it easier for the player to easily see what's going on on the map.

• Information screen (tech-tree) should be easy to understand and informative.

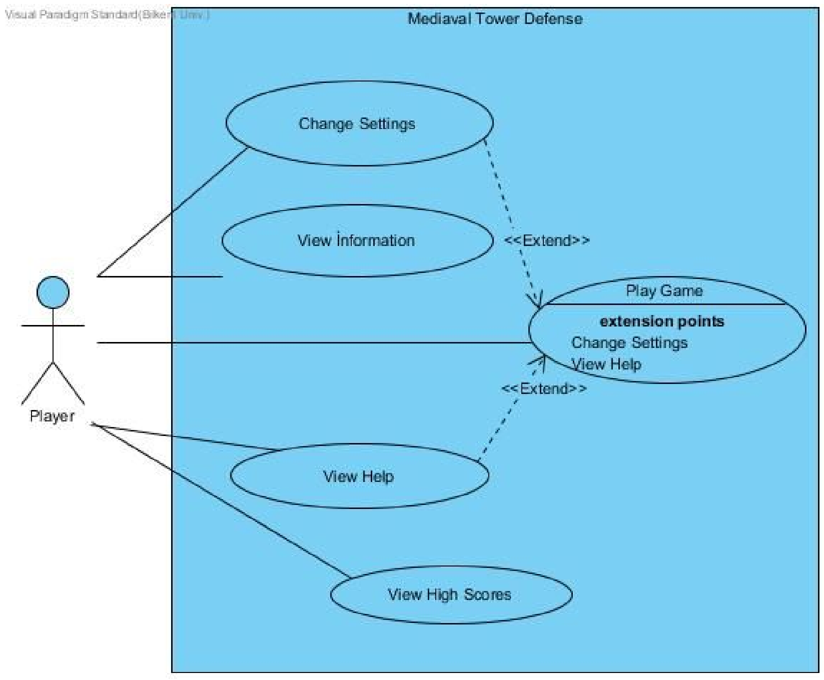
3.3 Pseudo Functional Requirements

• The game will be implemented in Java.

• Game art will be done using Adobe® Photoshop CC, Balzamiq and Photoscape.

4. System Models

This section provides information about the main use case model of MTD game, detailed use case explanations are included below.



4.1 Use Case Model

4.1.1 Play Game

Use case 1: Play Game

Participating actors: Player

Aim and System Response:

-Player aims to choose and place towers efficiently to not lose any lives. Losing a life means letting an enemy through, therefore also losing gold and reducing the chance of win.

-System keeps the score of the Player.

Entry condition: Player has opened the game and selected Play Game from the main menu.

Exit condition:

1. Player has won the game with or without achieving a high score, OR

2. Player has lost, i.e., letting through more enemies than his/her lives, OR

3. Player has exited the game from the in-game menu/pause game menu

Preconditions: Player defined settings are used when starting a game if the player hasn't visited & changed the settings tab from the main menu, pre-defined settings are used when Play Game is selected.

Post-conditions: If the player's score is high enough to be on the leaderboard, it is added to the highest scores list.

Main Flow of Events:

1. The player starts the game from the first level by clicking the play game button

2. Player constructs his/her tower system in the given one minute time before each wave appears. The player can also add towers when the wave is spawn.

Alternative Flows

Flow A:

3A. Player successfully finishes all the levels, i.e., not letting through more enemies than his given lives.

4A. The player's score is displayed on screen. If the player is among the top ten scorers, he enters his name to be added to the high scores list.

5A. Player returns to the main menu.

Flow B:

3B. Player lets through more enemies than his lives.

4B. The player's score is displayed on screen. If the player is among the top ten scorers, he enters his name to be added to the high scores list.

5B. Player returns to the main menu.

Flow C:

3C. Player pauses the game while playing, accessing the in-game menu.

4C. Player decides to quit from the in-game menu before finishing the game.

Flow D:

3D. Player decides to continue playing.(Back to step 2)

4.1.2 Change Settings

Use Case 2: Change Settings

Actors involved: Player

Aim and System Response:

-Players want to change the settings such as difficulty, sound volume, etc.

Pre-condition: Default game settings are given by the system each time the game is started.

Post-condition: Game settings are changed by the player.

Entry Condition: Player selects the "Settings" button from the main menu or from the in-game menu.

Exit Condition: Player selects "Back" to return to the menu

Main Flow of Events::

1. Player clicks on the "Settings" button from the menu.

2. Game settings are displayed with interactable buttons.

Alternative Flows:

Flow 1:

3A. Player changes settings.

4A. Player returns to the menu by pressing "Back." The changes are saved.

Flow 2:

3B. Player returns to the menu by pressing "Back." No new settings have been saved.

4.1.3 View Help

Use Case 3: View Help

Primary Actor: Player

Stakeholders and Interests:

- Player needs help in how to play the game.

- Player is given explanations on how certain aspects of the game work, the information is given in text format by the system.

Entry Condition: Player selects "View Help" from Main Menu or in-game Menu.

Exit Condition: Player selects "Back" to return to either main menu or the in-game menu.

Pre-conditions: Player is in the main menu or in-game menu.

Post-condition: -

Main Flow of Events::

1. Player click on "Help" on the menu.
2. A text explaining the controls and the aim of the game is shown.

4.1.4 View Credits

Use Case 4: View Credits

Primary Actor: Player

Aim and System Response:

- Player aims to get information about the developers.

- Name and contact information of the developers is displayed.

Pre-conditions: Player should be in the Main Menu .

Post-condition: -

Entry Condition: Player hits the "Credits" button from main menu.

Exit Condition: Player hits the "Back" button to return back to the main menu.

Main Flow of Events:

1.Player clicks on the "Credits" button from the menu.

2.Information regarding the developers is displayed.

3.Player returns back to the menu by hitting the "Back" button.

4.1.5 View High Scores

Use Case 5: View High Scores

Primary Actor: Player

Aims and System Response:
-Player aims to see the leaderboard.
-The system shows the top ten scorers with their saved nicknames.

Pre-conditions: System should save the high scorers after every game instance is ended.
Post-condition: -

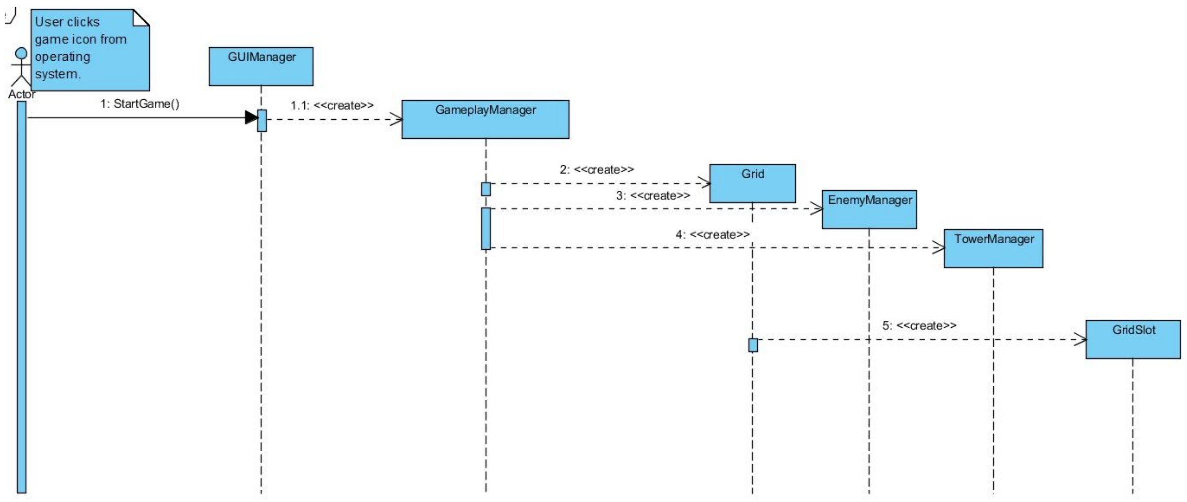
Entry Condition: Player hits the “High Scores” button from the main menu.
Exit Condition: Player hits the “Back” button to return back to the main menu.

Event Flow:

1. The player hits the “High Scores” button.
2. The system shows the nicknames of the top ten scorers.
3. The player hits the “Back” button when he is done examining the high scorers.

4.2 Dynamic Models

4.2.1 Start Game



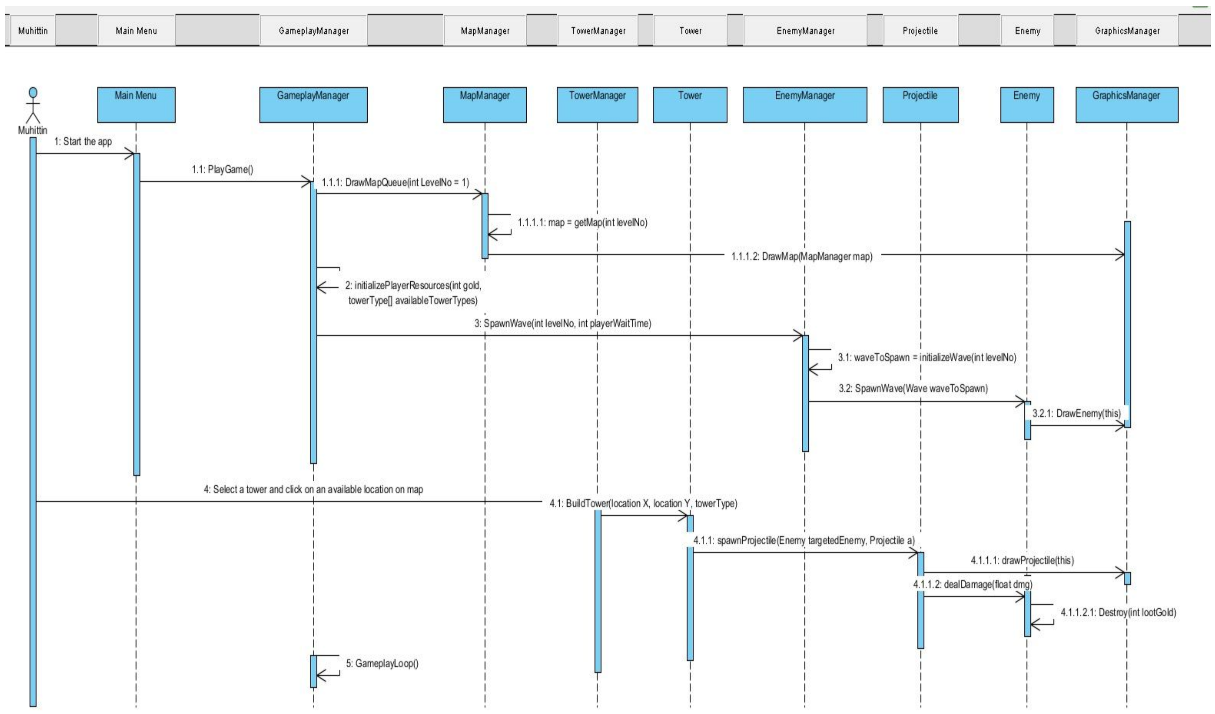

Muhittin
Main Menu
GameplayManager
MapManager
TowerManager
Tower
EnemyManager
Projectile
Enemy
GraphicsManager
1: Start the app
1.1: PlayGame()
1.1.1: DrawMapQueue(int LevelNo = 1)
1.1.1.1: map = getMap(int levelNo)
1.1.1.2: DrawMap(MapManager map)
2: initializePlayerResources(int gold, towerType[] availableTowerTypes)
3: SpawnWave(int levelNo, int playerWaitTime)
3.1: waveToSpawn = initializeWave(int levelNo)
3.2: SpawnWave(Wave waveToSpawn)
3.2.1: DrawEnemy(this)
4: Select a tower and click on an available location on map
4.1: BuildTower(location X, location Y, towerType)
4.1.1: spawnProjectile(Enemy targetedEnemy, Projectile a)
4.1.1.1: drawProjectile(this)
4.1.1.2: dealDamage(float dmg)
4.1.1.2.1: Destroy(int lootGold)
5: GameplayLoop()

Figure 4.2.1.1:Start Game Sequence Visualized
Scenario Name: Play Game
Event Flow:

Player Muhittin clicks the Play Game button to start playing. Then the MapManager system draws the pre-defined level and the pathway for enemies, then MapManager calls the GraphicsManager to display the graphics objects for these elements. Then the system gives the instructions and waits 30 seconds for the player to build towers. The player can then build the available towers with his resources.

EnemyManager creates the Enemy instances, each enemy instance calls the GraphicsManager to be drawn on the map. In the same manner, each tower instance calls the GraphicsManager to be drawn. For each enemy destroyed, the player receives resources to build/upgrade new towers. The amount of resources given to the player depends on the attributes of the enemy that is destroyed. Once all the enemies of a particular wave is either destroyed by the player or has crossed the end of the pathway, the system then goes into the gameplay loop for the next wave and again waits 30 seconds for the player to prepare new towers.Once the player wins or loses the game, the system asks for the nickname of the player if his score is in the top 10 among all scores so far.

4.2.2 Menu Operations

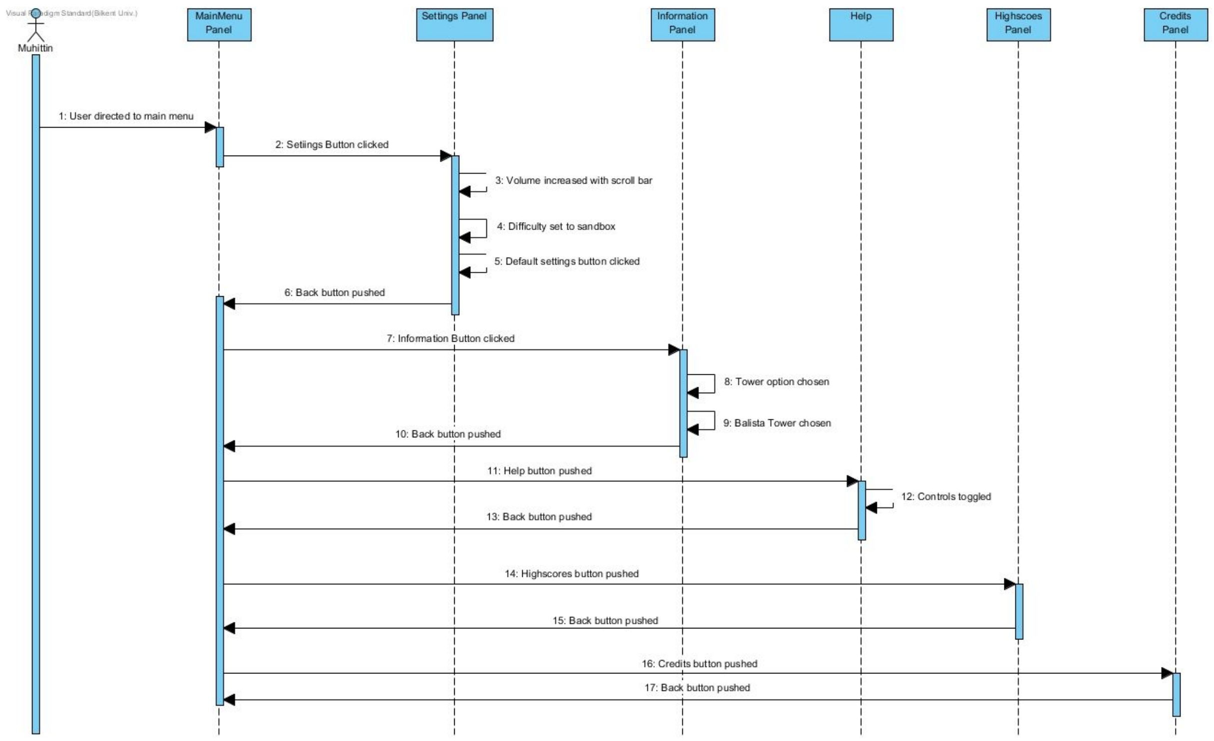


Figure 4.2.2.1:Main Menu Operations Visualized

Description:

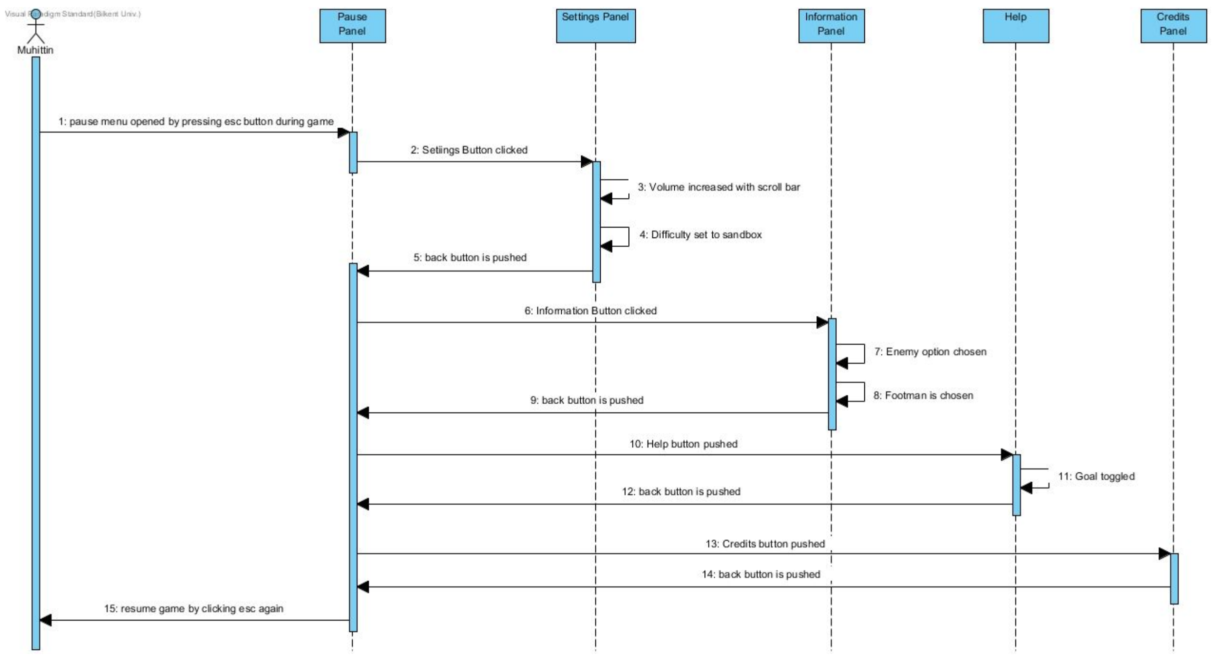
Through main menu, player is able to display the functionings and features of game. The menu buttons beside gameplay are as follows: settings, information, help, highscores, and credits. These buttons direct player to relevant panels. In settings, player can adjust sound via a slider. Difficulty is also chosen in settings through a box which contains several toggle buttons. Information panel includes a subpanel and basically shows information about unit and tower types. From help panel player can visualize the game controls and goal. In highscores, player can see which players have the highest score. And last, credits panel holds information about game developers.

Scenario Name: Menu Browsing

Event Flow:

Muhittin wants to browse on the main menu. First, he clicks on the settings from the main menu, and comes accross with the settings panel. Through there, he adjusts the sound with slider, and he sets the difficulty as sandbox. Then , he returns to main menu with back button, and he decides to view some information about tower types. Through Information button, he's directed to information subpanel. By choosing towers, and then choosing balista tower as next step, he manages to view the information of Balista Tower type. He turns back via back button and this time, and he wants to enter help section by clicking help button. In help panel, he toggles, control tab to view the game controls. After learning the controls, he returns main menu again. He also clicks high scores button in which high scores of players are displayed. After viewing, he's back to main menu once again via back button. Lastly, he decides to pay tribute to the game creators by clicking to the credits button and choosing to display credits panel. He turns back to the main menu again.

4.2.3 Pause-menu Operations



Visual Paradigm Standard(Bilkent Univ.)
Muhittin
Pause Panel
Settings Panel
Information Panel
Help
Credits Panel
1: pause menu opened by pressing esc button during game
2: Setings Button clicked
3: Volume increased with scroll bar
4: Difficulty set to sandbox
5: back button is pushed
6: Information Button clicked
7: Enemy option chosen
8: Footman is chosen
9: back button is pushed
10: Help button pushed
11: Goal toggled
12: back button is pushed
13: Credits button pushed
14: back button is pushed
15: resume game by clicking esc again

Figure 4.2.3.1:Pause Menu Operations Visualized
Scenario Name: Pause-menu Browsing

Event Flow:

Once he pauses the game, Muhittin gains access to the pause menu. Through there, he adjusts the game sound with slider, and sets difficulty to sandbox. He pushes the back button afterwards, and just before starting the game, he views some information about an enemy type that is footman. Also, he decides to look at the help section by clicking help button. In here, he toggles goal tab and checks the game goal. As last step, he returns to pause menu by clicking back button and checks out credits by clicking credits button. After visualizing the information of game develipers, he resumes the game with esc button and continues to play.

4.2.4 Buy Tower

Event Flow:

Muhittin selects a tower on the shop, when he has enough gold stored in the GameplayManager, the tower buy mode is enabled. At this point, Muhittin clicks on a grid slot and the tower in that slot is converted from null to the bought tower. Also hasTower boolean turned true for the grid slot instance, so Muhittin can't place any more towers on that slot. When the tower is placed, the gold requirement specified in the tower class is subtracted from Muhittin's gold.

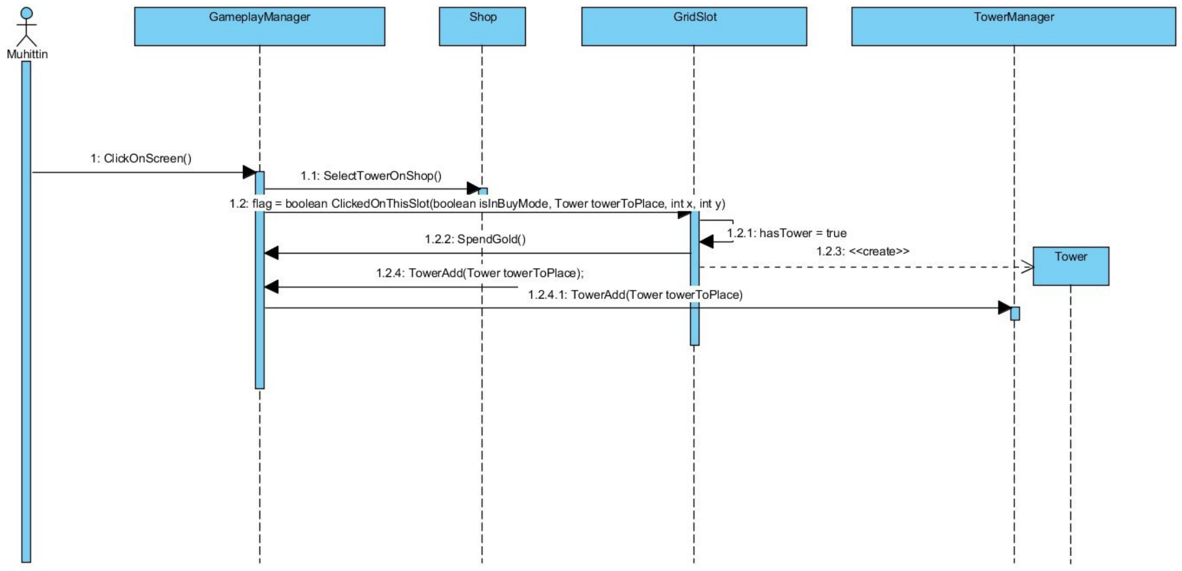

Muhittin
GameplayManager
Shop
GridSlot
TowerManager
1: ClickOnScreen()
1.1: SelectTowerOnShop()
1.2: flag = boolean ClickedOnThisSlot(boolean isInBuyMode, Tower towerToPlace, int x, int y)
1.2.1: hasTower = true
1.2.2: SpendGold()
1.2.3: <<create>>
Tower
1.2.4: TowerAdd(Tower towerToPlace);
1.2.4.1: TowerAdd(Tower towerToPlace)

Figure 4.2.4.1 Buy Tower Visualized

Scenario Name: Buy Tower

Description: Player can buy towers by interacting with shop menu via mouse. Clicking on the desired tower, and placing it into chosen grid section. Tower management will add the tower to the list and grid section will reserved and act as a reserved collision field. This operation is enabled by the game manager. The game manager handles also the money operations.

4.2.5 Upgrade Tower

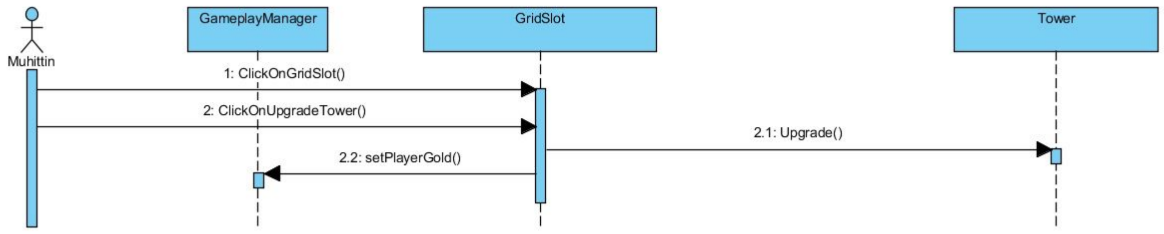


Figure 4.2.5.1 Upgrade Tower Visualized

Scenario Name: Upgrade Tower

Description: Once bought, towers can be upgraded if mouse click corresponds to the grid of a placed tower. There will be a sub section that includes the upgrade tower icon. Once clicked, player gold is reduced as much as upgrade price and tower is upgraded.

4.2.6 Play Game

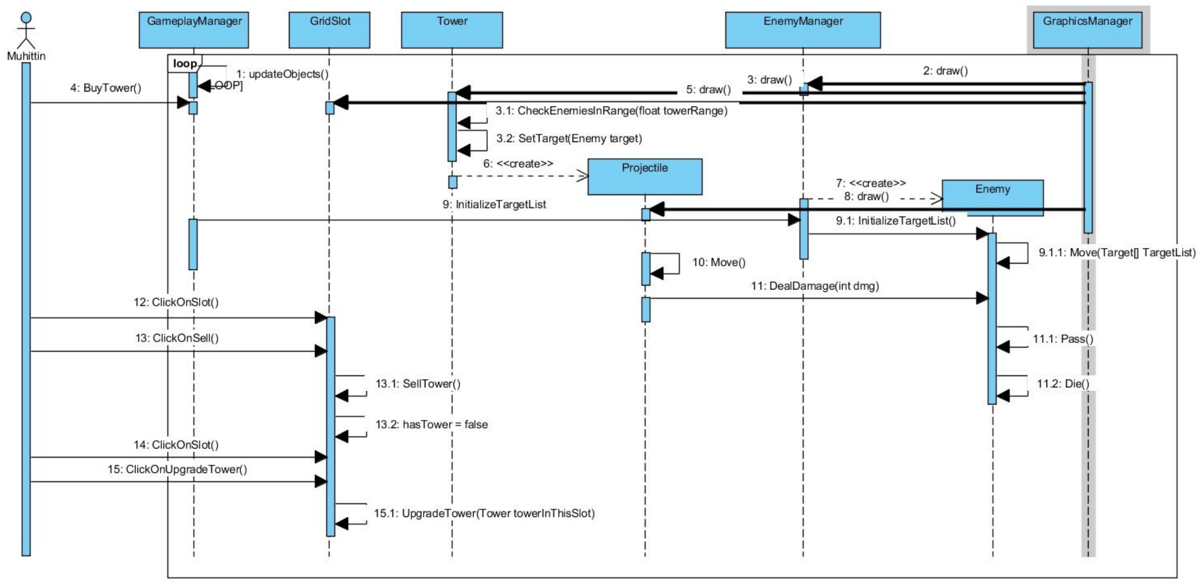


Figure 4.2.6.1 Play Game Visualized

Scenario Name: Play Game

Event Flow:

Once the game is started Gameplay manager updates all the objects on a frequent time frame defined in the source code. This frequency is controlled by a variable and scales the game's performance and smoothness of the objects' movements. Graphics managers draws all the drawable game objects such as Tower, GridSlot etc. based on this frequency. The higher the frequency, the more draw calls, more work on the hardware but smoother game experience. The towers frequently check if there is an enemy within their range. Once a tower has a target in their range, i.e. it's target attribute is not null, it goes into a loop of spawning projectiles until the target is out of range. The projectiles and the enemies call the move function based on a Timer. Since the projectiles are much faster than the enemies(Their speed is set in their attributes) they catch up with their target. When the direct distance between the projectile and it’s target is less than 5 pixels, it is considered a hit, and damage is dealt to the target enemy instance.

5.2 Object and Class Model

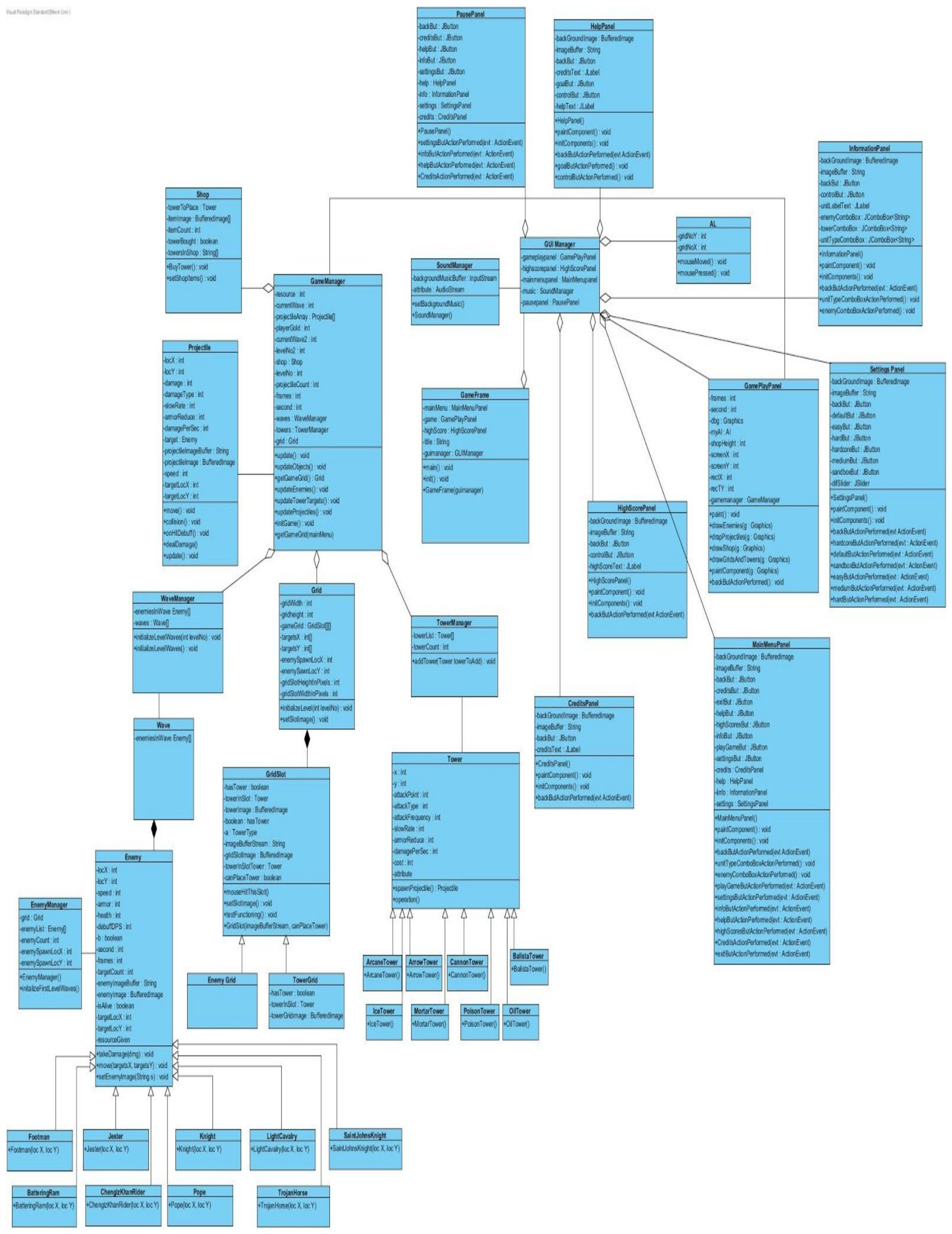


Figure 5.2.1 Class diagram illustrating the complete class diagram

Class diagram shows the interaction of application objects. These interactions include assocation relationships such as composition and aggregation, dependency relationships that represent the inheritence, and general association. All objects are bound to the chosen Facade class which is GameFrame class.

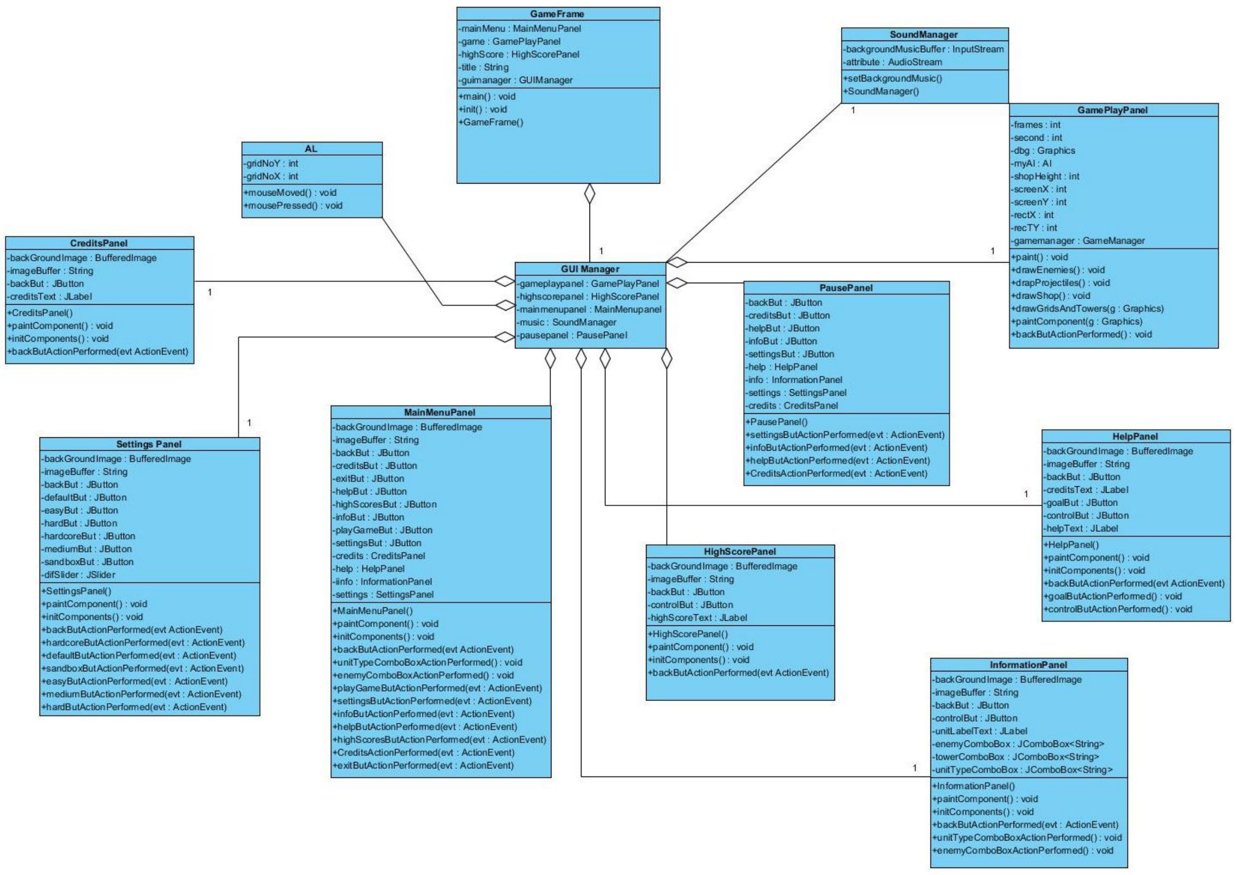

GameFrame
-mainMenu : MainMenuPanel
-game : GamePlayPanel
-highScore : HighScorePanel
-title : String
-guimanager : GUIManager
+main() : void
+init() : void
+GameFrame()
SoundManager
-backgroundMusicBuffer : InputStream
-attribute : AudioStream
+setBackgroundMusic()
+SoundManager()
GamePlayPanel
-frames : int
-second : int
-dbg : Graphics
-myAI : AI
-shopHeight : int
-screenX : int
-screenY : int
-rectX : int
-recTY : int
-gamemanager : GameManager
+paint() : void
+drawEnemies() : void
+drapProjectiles() : void
+drawShop() : void
+drawGridsAndTowers(g : Graphics)
+paintComponent(g : Graphics)
+backButActionPerformed() : void
AL
-gridNoY : int
-gridNoX : int
+mouseMoved() : void
+mousePressed() : void
CreditsPanel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-creditsText : JLabel
+CreditsPanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt ActionEvent)
GUI Manager
-gameplaypanel : GamePlayPanel
-highscorepanel : HighScorePanel
-mainmenupanel : MainMenupanel
-music : SoundManager
-pausepanel : PausePanel
PausePanel
-backBut : JButton
-creditsBut : JButton
-helpBut : JButton
-infoBut : JButton
-settingsBut : JButton
-help : HelpPanel
-info : InformationPanel
-settings : SettingsPanel
-credits : CreditsPanel
+PausePanel()
+settingsButActionPerformed(evt : ActionEvent)
+infoButActionPerformed(evt : ActionEvent)
+helpButActionPerformed(evt : ActionEvent)
+CreditsActionPerformed(evt : ActionEvent)
Settings Panel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-defaultBut : JButton
-easyBut : JButton
-hardBut : JButton
-hardcoreBut : JButton
-mediumBut : JButton
-sandboxBut : JButton
-difSlider : JSlider
+SettingsPanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt ActionEvent)
+hardcoreButActionPerformed(evt : ActionEvent)
+defaultButActionPerformed(evt : ActionEvent)
+sandboxButActionPerformed(evt : ActionEvent)
+easyButActionPerformed(evt : ActionEvent)
+mediumButActionPerformed(evt : ActionEvent)
+hardButActionPerformed(evt : ActionEvent)
MainMenuPanel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-creditsBut : JButton
-exitBut : JButton
-helpBut : JButton
-highScoresBut : JButton
-infoBut : JButton
-playGameBut : JButton
-settingsBut : JButton
-credits : CreditsPanel
-help : HelpPanel
-iinfo : InformationPanel
-settings : SettingsPanel
+MainMenuPanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt ActionEvent)
+unitTypeComboBoxActionPerformed() : void
+enemyComboBoxActionPerformed() : void
+playGameButActionPerformed(evt : ActionEvent)
+settingsButActionPerformed(evt : ActionEvent)
+infoButActionPerformed(evt : ActionEvent)
+helpButActionPerformed(evt : ActionEvent)
+highScoresButActionPerformed(evt : ActionEvent)
+CreditsActionPerformed(evt : ActionEvent)
+exitButActionPerformed(evt : ActionEvent)
HelpPanel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-creditsText : JLabel
-goalBut : JButton
-controlBut : JButton
-helpText : JLabel
+HelpPanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt ActionEvent)
+goalButActionPerformed() : void
+controlButActionPerformed() : void
HighScorePanel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-controlBut : JButton
-highScoreText : JLabel
+HighScorePanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt ActionEvent)
InformationPanel
-backGroundImage : BufferedImage
-imageBuffer : String
-backBut : JButton
-controlBut : JButton
-unitLabelText : JLabel
-enemyComboBox : JComboBox<String>
-towerComboBox : JComboBox<String>
-unitTypeComboBox : JComboBox<String>
+InformationPanel()
+paintComponent() : void
+initComponents() : void
+backButActionPerformed(evt : ActionEvent)
+unitTypeComboBoxActionPerformed() : void
+enemyComboBoxActionPerformed() : void
1
1
1
1
1
1
1

Figure 5.2.2 Class diagram of classes associated with user interface

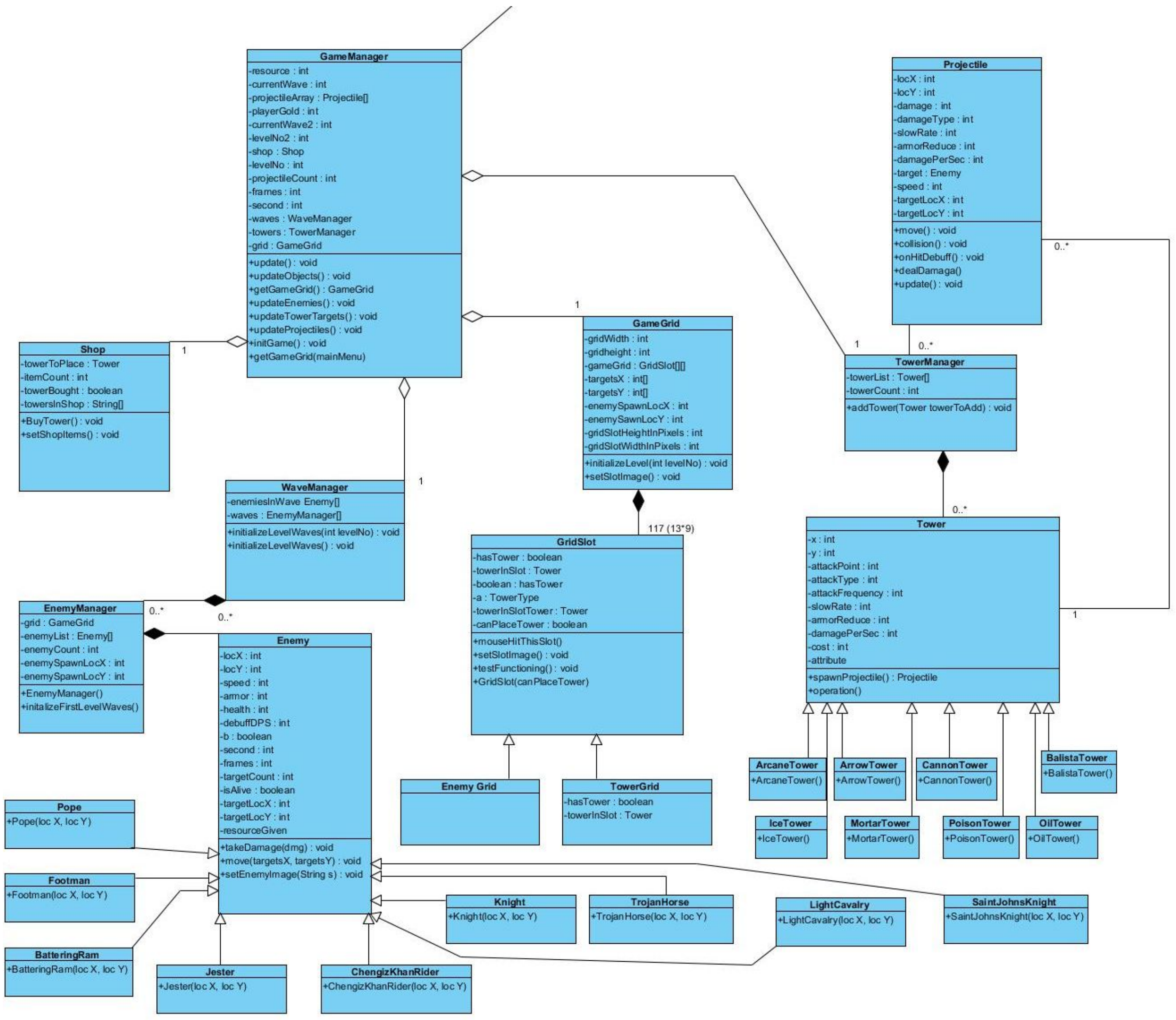


Figure 5.2.3 Class diagram Illustrates the game components

GameFrame is the class that stores all the user interface elements and the class to start execution. It will hold and create several user interface components and SoundManager class which will construct game music and sounds.

GameManager is the class that holds several objects and will execute the main game. It holds several manager objects that controls entity objects. AL is the class responsible for all user interactions.

6. User Interface

6.1 Navigational Path

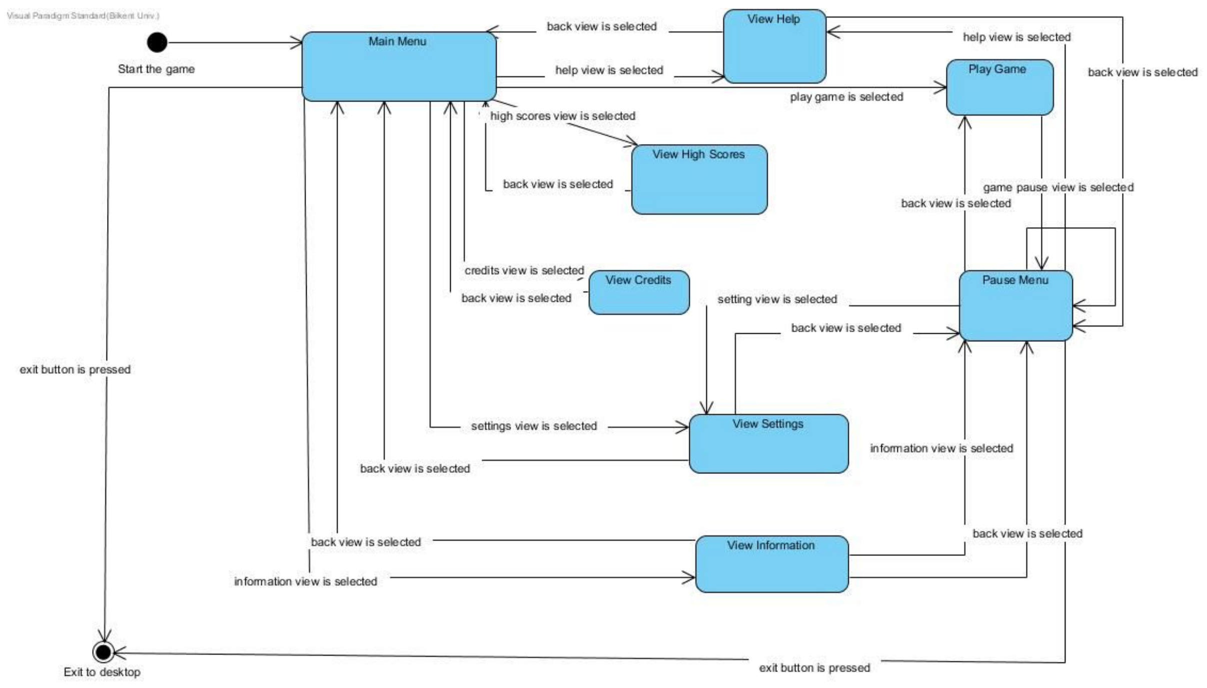


Figure 6.1 State machine diagram Illustrates the Navigational Path

6.2 Screen Mock-ups

6.2.1 Main Menu

When the game is started, player will see the main menu screen. Main menu contains six options as buttons and an exit icon. Player will be able to choose between playing the game, change settings, get information about the units in the game, get help, see his/her and other player's high scores, see credits and exit the game. (Figure 6.2.1)

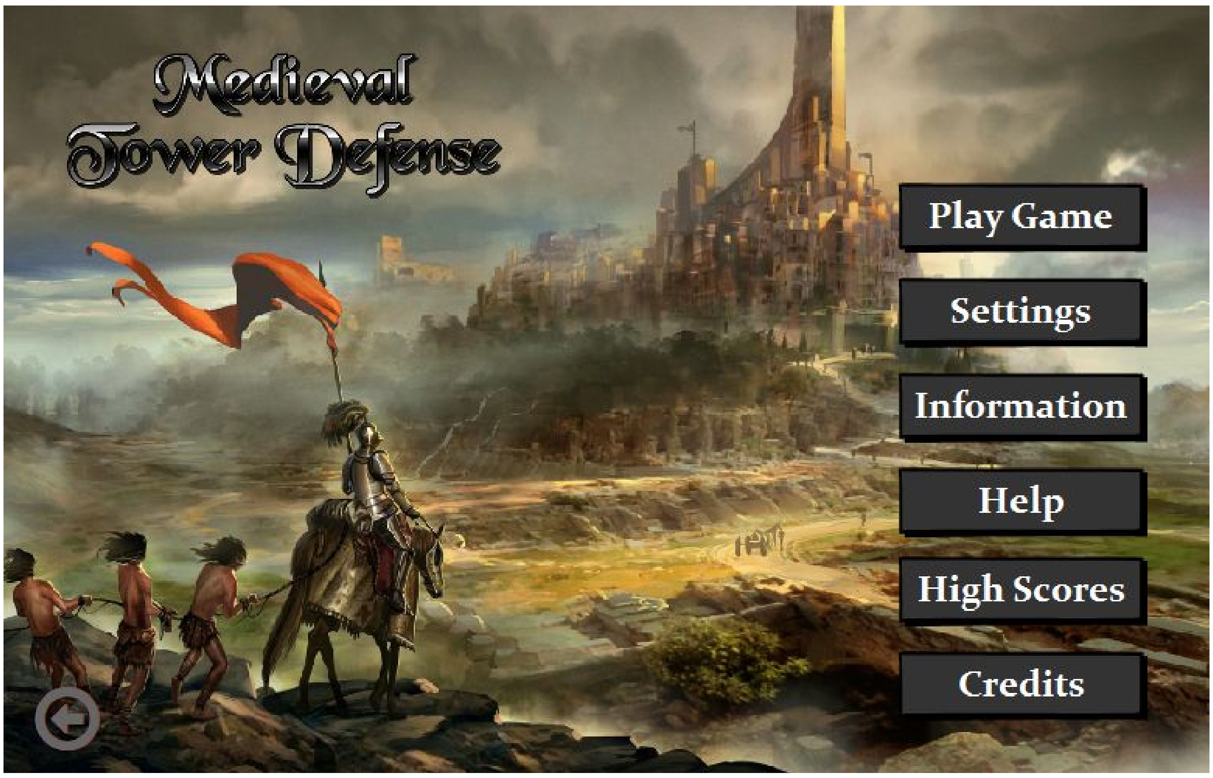


Figure 6.2.1 A Balsamiq mock-up of the main menu

- Play Game:

If the user selects the play game option, game starts with the selected settings. (Figure 6.2.1.1)

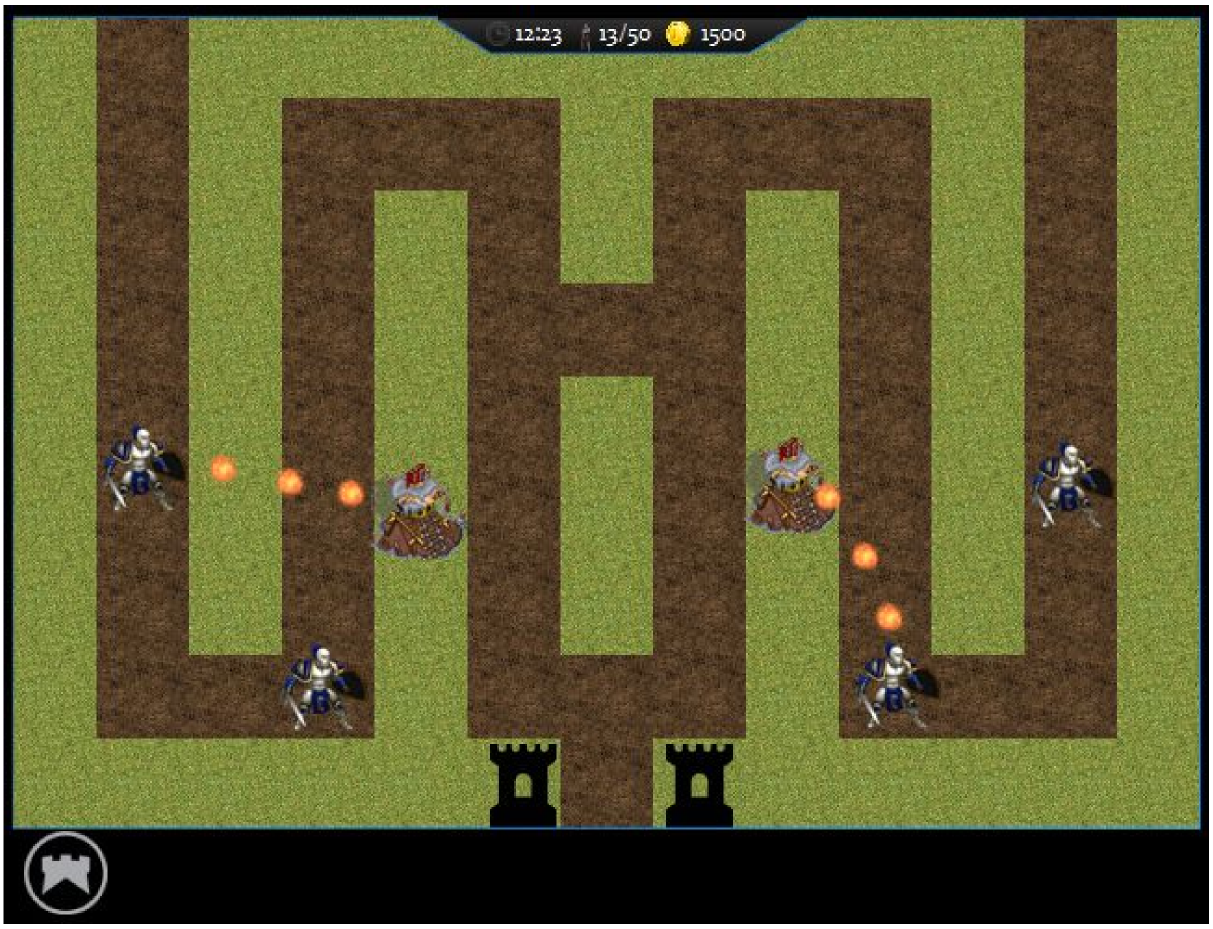


Figure 6.2.1.1: Balsamic mock-up of the gameplay

- Settings:

When the player selects the settings button, a screen containing several options appears on the screen. (Figure 6.2.1.2) These settings can be changed to the desired levels by the player. Automatically, default settings are selected. Players can change game difficulty and turn the volume up and down.

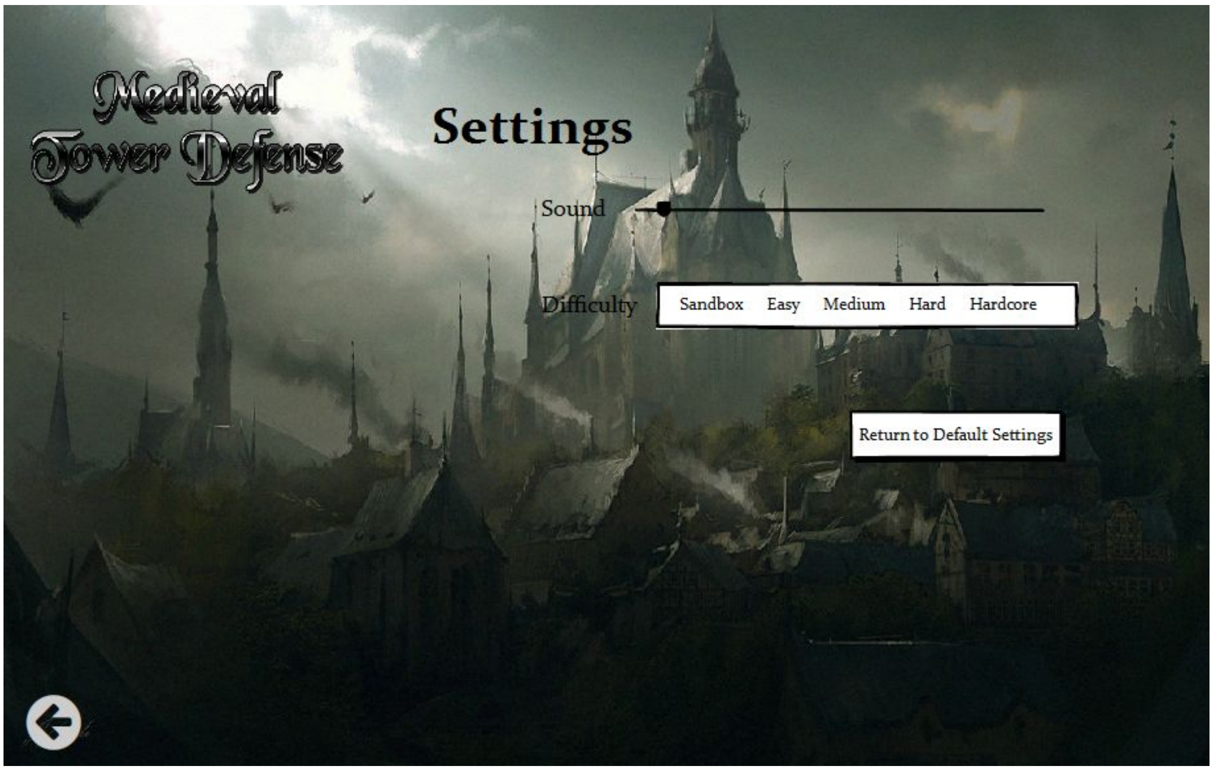
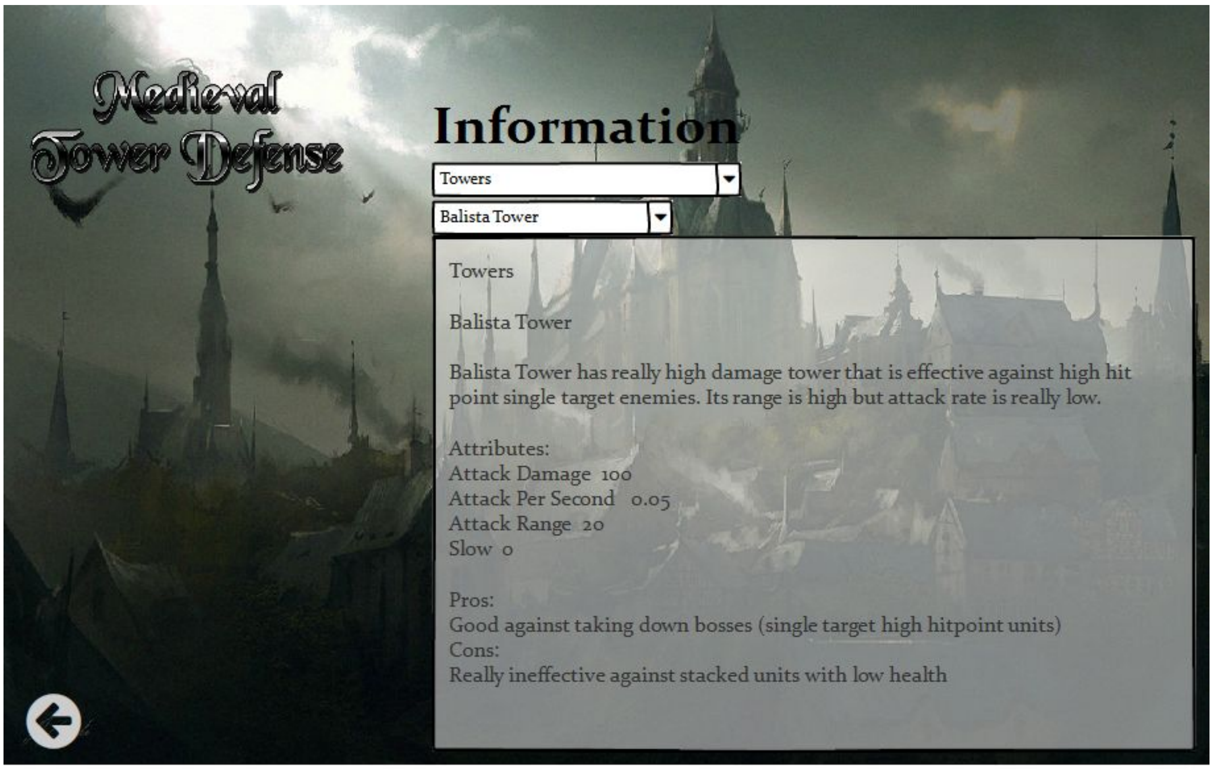


Figure 6.2.1.2: Balsamiq mock-up of the change settings

- Information:

When the player selects the information button, a screen containing the tech-tree of the units in the game appears.(Figure 6.2.1.3) The player can read about the attributes of the enemies, towers, waves and gather information about their strategies in this window.



Towers

Balista Tower

Balista Tower has really high damage tower that is effective against high hit point single target enemies. Its range is high but attack rate is really low.

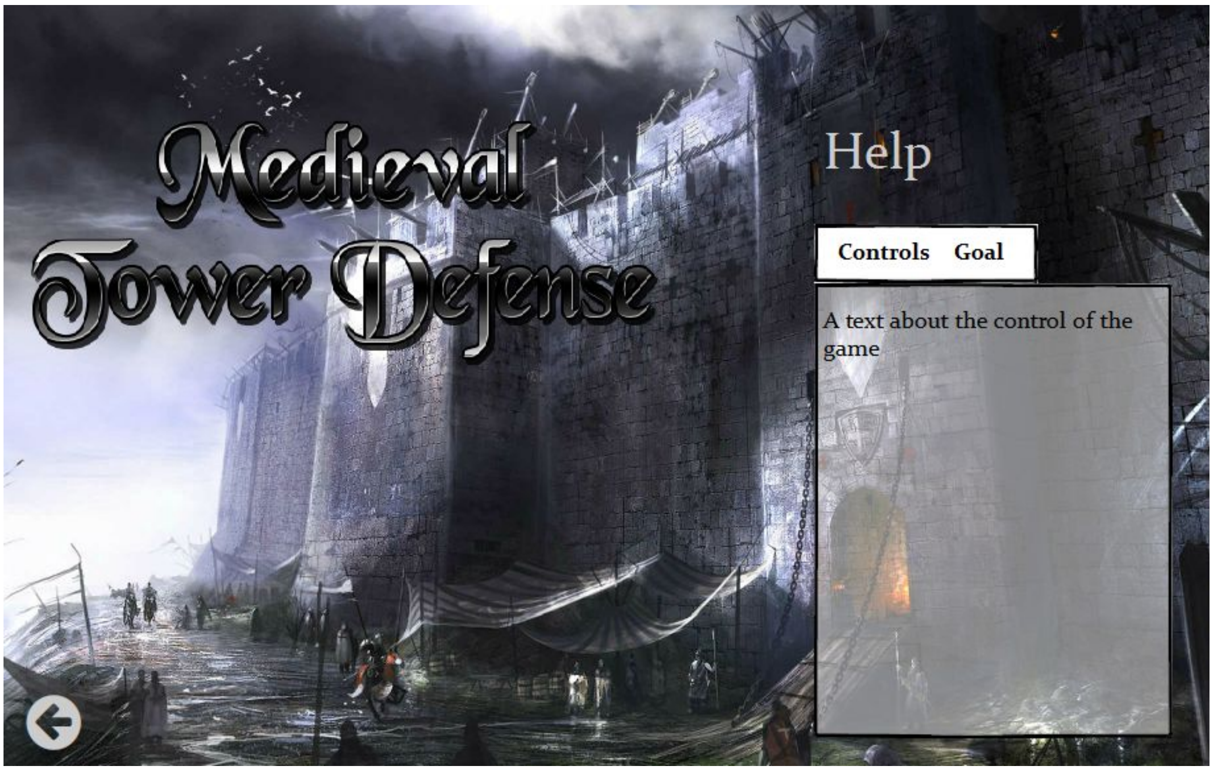
Attributes:
Attack Damage 100
Attack Per Second 0.05
Attack Range 20
Slow 0

Pros:
Good against taking down bosses (single target high hitpoint units)
Cons:
Really ineffective against stacked units with low health

Figure 6.2.1.3: Balsamiq mock-up of the information screen

- Help:

When the player selects the help button, a screen containing the controls, basic concepts and aims of the game appears. (Figure 6.2.1.4) The player can read about the controls and how to play the game in this screen.

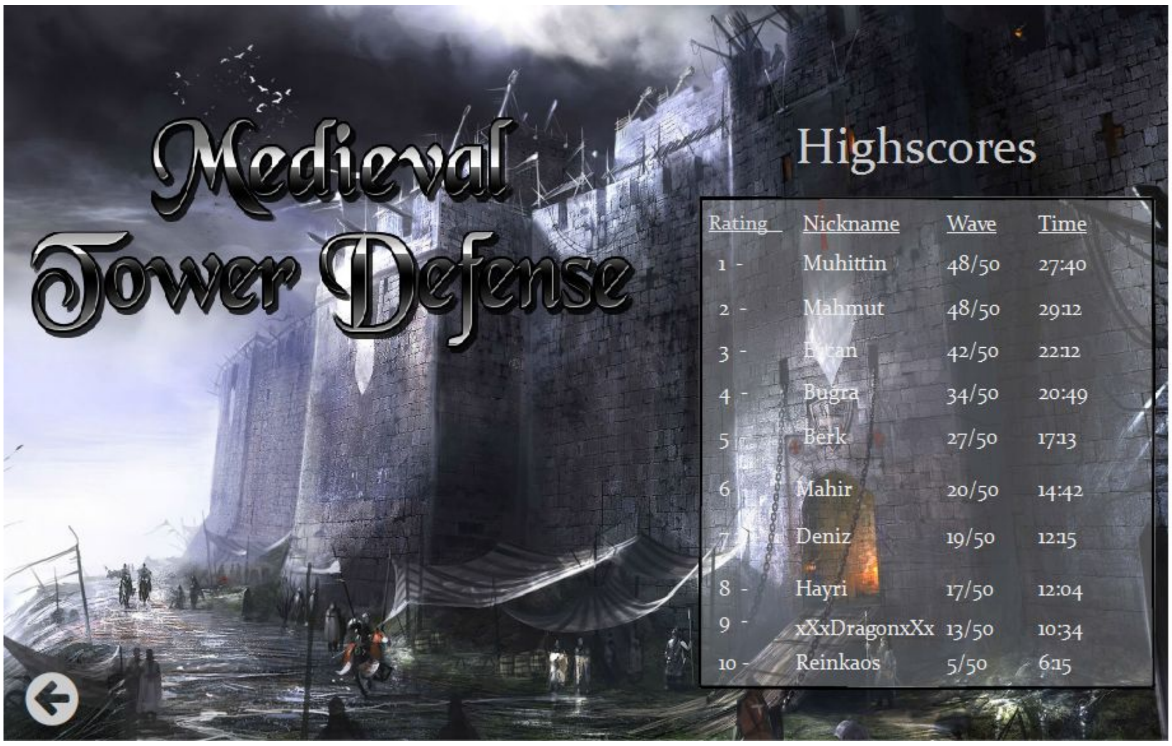


A text about the control of the game

Figure 6.2.1.4: Balsamiq mock-up of the help screen

- High Scores:

When the player selects the high scores button, a screen that shows the top 10 high scores of the game appears. (Figure 5.2.1.5) These top 10 high scores are arranged according to the wave progress of the players. It priorities the wave progress, but if two players' wave progress is same, then it chooses the one with less time spent.



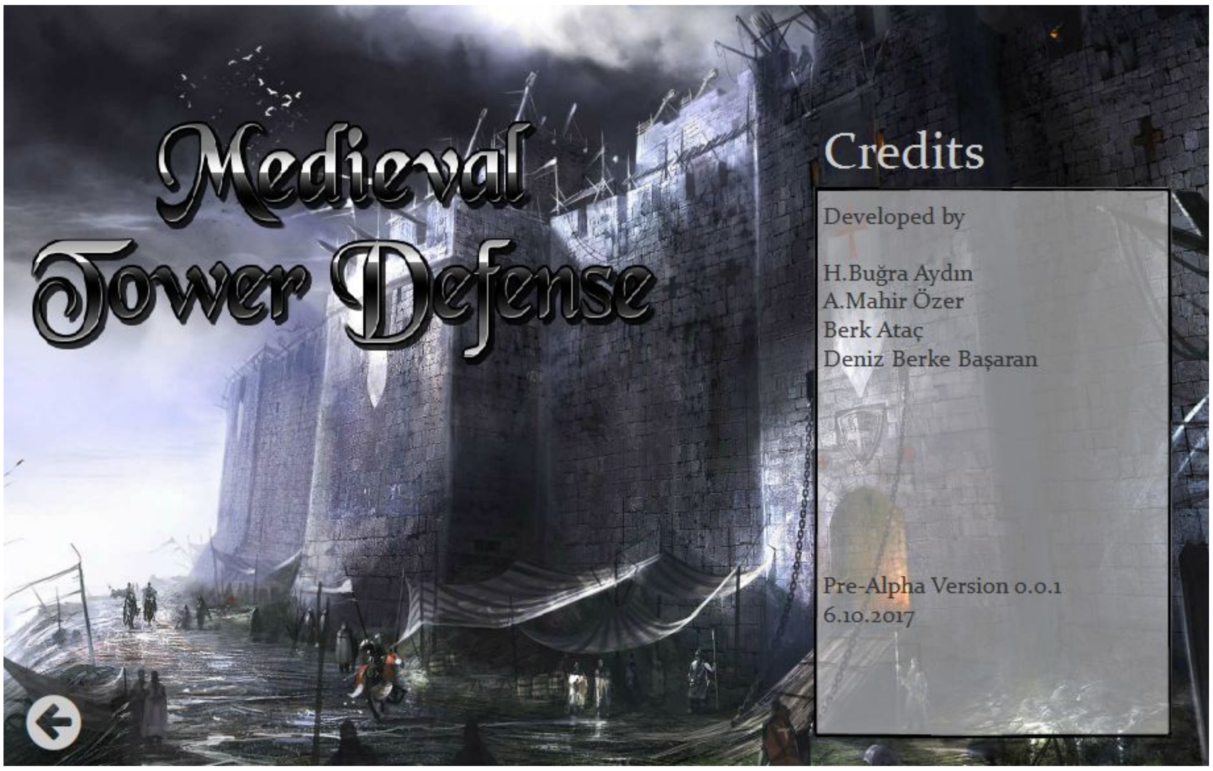
Highscores

Rating	Nickname	Wave	Time
1 -	Muhittin	48/50	27:40
2 -	Mahmut	48/50	29:12
3 -	Ertan	42/50	22:12
4 -	Bugra	34/50	20:49
5	Berk	27/50	17:13
6	Mahir	20/50	14:42
7	Deniz	19/50	12:15
8 -	Hayri	17/50	12:04
9 -	xXxDragonxXx	13/50	10:34
10 -	Reinkaos	5/50	6:15

Figure 6.2.1.5: Balsamiq mock-up of the high scores screen

- Credits:

When the player selects the credits button, a screen containing the information about the developers appears (Figure 6.2.1.6)



Developed by

H.Buğra Aydın
A.Mahir Özer
Berk Ataç
Deniz Berke Başaran

Pre-Alpha Version 0.0.1
6.10.2017

Figure 6.2.1.6: Balsamiq mock-up of the credits screen

- Pause Menu

If the player decides to pause the game during play game feature, a screen containing several options appears. (Figure 6.2.1.7)

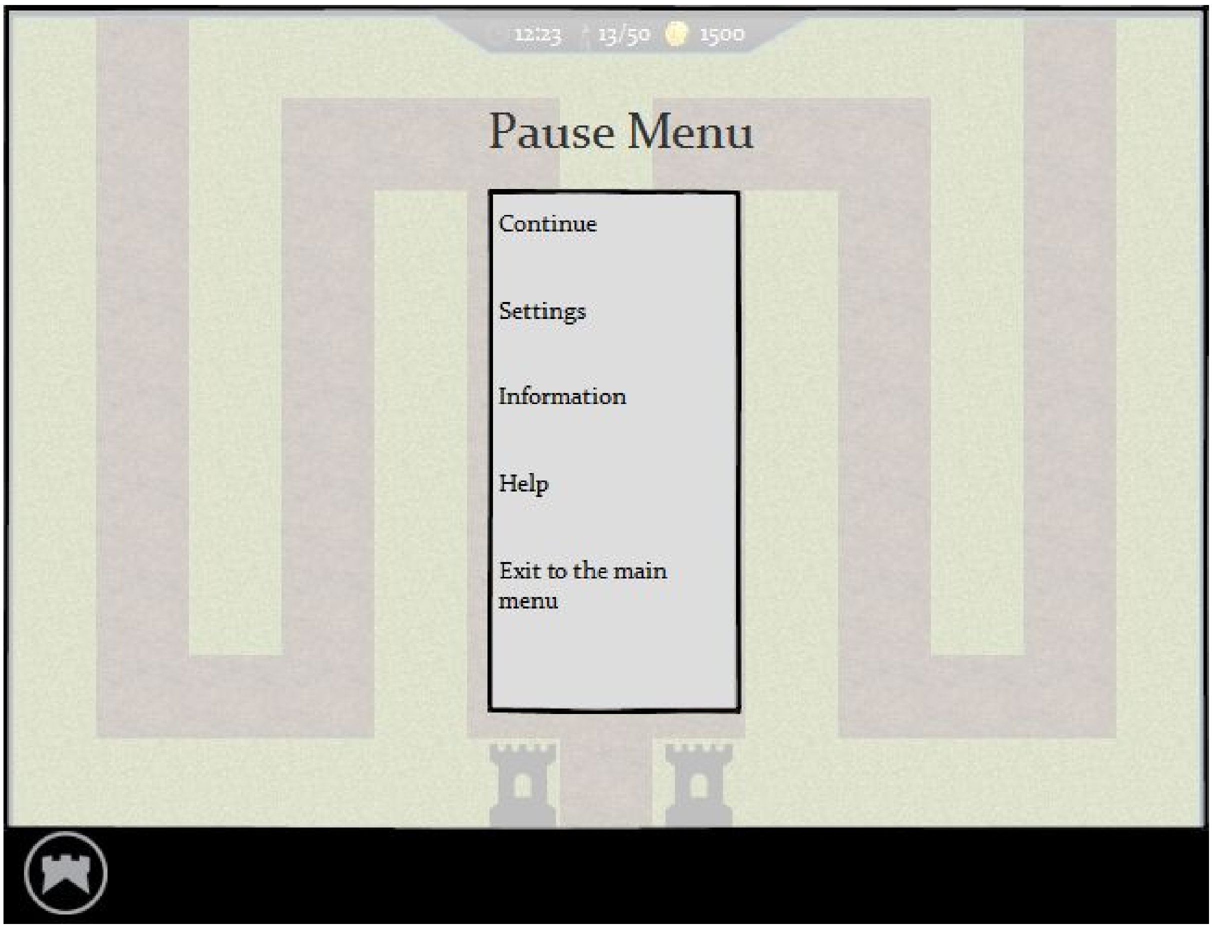


Figure 6.2.1.7: Balsamiq mock-up of the pause menu

7. Conclusion

In the second iteration of our design report, we have updated the use case diagram by adding more sophisticated use cases. These new use cases add more functionality to the game and extends the options of player. In addition, new dynamic models were created accordingly so that scenarios can be visualized. New object and class models are created so that main functioning of the game and relation between classes can be demonstrated. There are two different diagrams: one represents the panels and their associations, other represents the associations between game engine classes.

In this report, the requirements and conditions were specified, then the system models were given. Several changes were made according to the feedback given to the first iteration of the analysis report. The iterative development approach aided us to see our mistakes and take necessary actions accordingly. Basic principles are still not abondened such as readibility

8. Improvement Summary

In the second iteration, we added several new functional requirements for users. For example, user will be able to place traps on the enemy roads which gives different debuffs to enemies or damage them. User will also be able to upgrade towers and increase their attributes with his/her gold. User will be able to choose normal or survival game mode. Game will generate the map randomly in order to create different experiences for users. Navigational path was demonstrated as state machine diagram and contributes to the dynamic modeling as well as sequence diagram.

For the non-functional requirements, we decided to make our tech-tree very informative and easy to understand in order to access readibility and ease of use.

We updated our class and object models which was missing in the first iteration, analysing the requirements and deciding on different classes for both user interface and game components. Our game classes follows a full object oriented approach, by making every real components into class representives. Associations and amount specifications are also shown by the links.

We thinked and constructed many different flow of events which may occur on our game and created several sequence diagrams for them. This gave our report a better dynamic model perspective in the second iteration.

9. References

https://sites.google.com/site/knightsvszombies/
http://www.cs.bilkent.edu.tr/~ugur/teaching/cs319/proj/11_2C/analysis.doc