CS 319 - Object-Oriented Software Engineering

System Design Report

Medieval Tower Defense

Group 1-A

Buğra Aydın

Deniz Başaran

Mahir Özer

1.Introduction

1.1 Purpose of the System

Purpose of the Medieval Tower Defense Game is to offer a quality gameplay experience to the strategy game audience. With an easy to use and control user interface and dynamic gameplay, players will enjoy the game. Also, audience will be able to obtain historical information by while playing the game. Medieval Tower Defense is a 2-D tower defense game. Gameplay of MTD will be designed as simplistic as it is challenging to maximize the pleasure of coming up with a defense strategy for player. The concept of the game will be very simple, yet the player will have to generate different strategies in order to handle different enemies and bosses with different attributes. MTD will also have distinguishable content from other tower defense games as it will have unique boss encounters, difficulty levels and game mods. MTD's goal is to improve decision making, situation judging and sense of strategy while having a good time.

1.2 Design Goals

It is really important to decide on the design goals of the system in order to have a brief idea on the qualifications that the program contain. Due to that, while deciding on the design goals, we focused on our functional and nonfunctional requirements of the system. Important design goals of this system are explained below.

End User Criteria:

Ease of Use

Easiness in the usage is an important design goal in such games in terms of user's comfort. It makes the game more friendly and attractive. Therefore, the system will be designed such that user can easily interact with the system. The game will have a help document in the main menu and in-game menu. However, user friendliness of the system does not imply making the gameplay easier, which might make the player bored sooner than expected.

Ease of Learning

Since our system is a game, it should provide good entertainment for the player. In order to provide the entertainment player should not have a difficulty in learning our system. In this respect, system will provide player friendly interfaces for menus, by which player will easily find desired operations, navigate through menus and perform the desired operations. User can easily access information about units from the information screen and develop strategies corresponding to those informations.

Maintenance Criteria:

Portability

Java is one of the programming languages which provide cross-platform portability. Java works on a portable virtual machine that can be adapted to different machines. That makes our system to be possible to work on different platforms. So, our system is portable.

Extendibility

It is important to add new content and features to keep the interest and excitement to a game to keep it alive. With our design perspective, it is possible to easily extend and add new features to the existing system such as new enemies, new tower types, new waves and different difficulty settings.

Performance Criteria:

Efficiency

The system is going to be responsive and able to run with high performance. This is the most important design goal because performance of the game has a crucial role for users' excitement.

Reliability

System will be bug-free and consistent in the boundary conditions. There should be no crashes and unintended actions such as glitches and bugs.

Trade Offs:

Efficiency – Reusability

We are not considering to integrate any of our classes in any different game or any other projects, so reusability is not our main concern. Therefore, the classes will designed specifically for the tasks of our game so the code is not made more complex than necessary. This design approach will fortify the our most important design goal, which is efficiency. So our design is focusing more on efficiency instead of reusability.

Memory – Perfomance

Our system is to be expected to be interactive, interesting and smooth. So, performance is our main aim since it is not a big project that needs to consider the memory efficiency. We may need to sacrifice memory in order to gain performance. For example, we store all waves, towers and enemies in our memory in order to gain performance, even though sacrificing memory is needed.

Functionality – Usability

We are aiming to be used by a wide range of players and to be focusing on a big audience. Therefore, the game will have plain usage. The system should not be too complex to play. It means that the functionality of the system will be basic as possible for a tower defense game. Our first aim is usability due to that. The game has a simple interface and easy controls to play instead of complex menus and various features. So we decided on to sacrifice on functuonality in order to make our system more usable by a wider range of audience.

1.3 Definitions, acronyms and abbreviations

Cross-platform

Cross-platform refers to ability of software to run in same way on different platforms such as Microsoft Windows, Linux and Mac OS X.[1]

JRE

The Java Runtime Environment (JRE), also known as Java Runtime, is part of the Java Development Kit (JDK), a set of programming tools for developing Java applications.[2]

FPS

Abbreviation of “frames per second”. Fps represents the number of graphical layouts can be prepared by the system each second.[3]

1.4 References

[1]https://www.webopedia.com/TERM/C/cross_platform.html

[2]http://www.theserverside.com/definition/Java-Runtime-Environment-JRE

[3]https://www.computing.net/define/fps

2. Software Architecture

2.1. Overview

When designing the software architecture, our main goal was to put the methods in their respective classes, aiming to have minimum redirections in between the classes & design the layer interactions neat and no more complex than it needs to be, in order to keep the software straightforward and easily updatable.

2.2 Subsystem Decomposition

In order to organize the system and make the implementation more manageable when working together, we tried to have a lesser number of Manager classes. In terms of performance, this decision doesn’t affect the running time of the game.

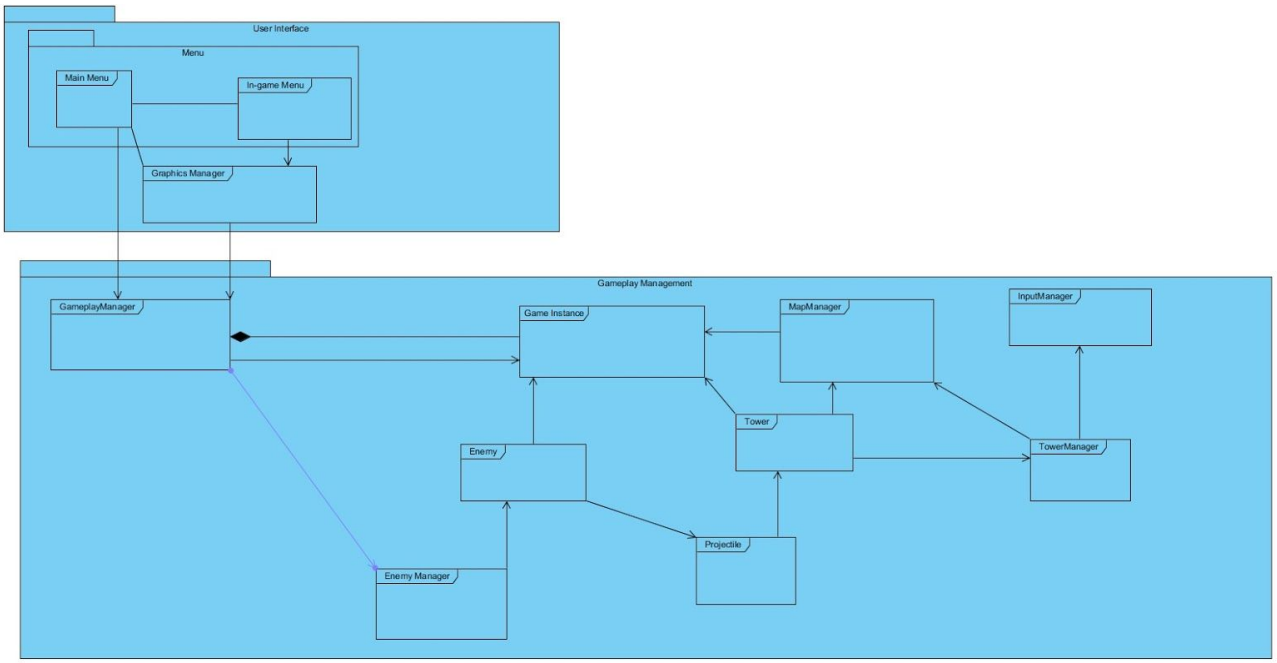


Figure 2.2.1: UI and Gameplay Layer interactions(Basic)

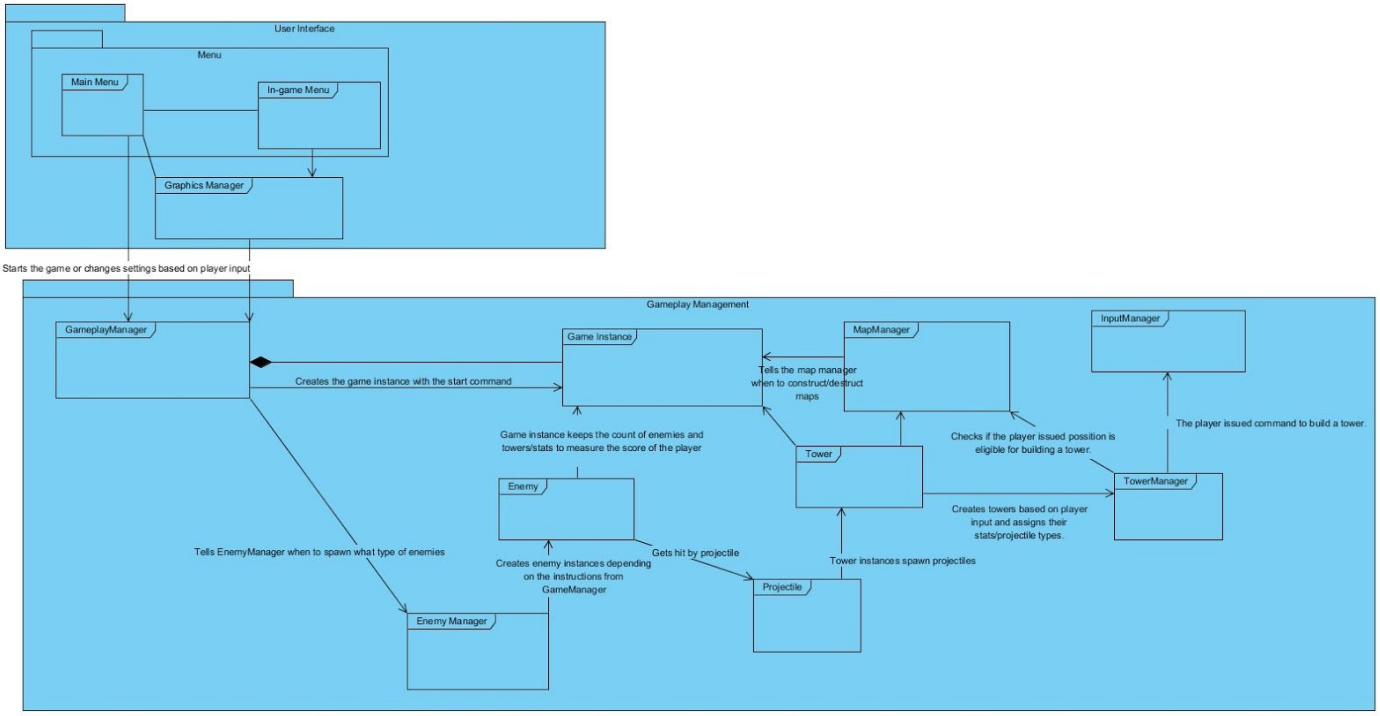


Figure 2.2.2:UI and Gameplay Layer interactions(Detailed)

2.3 Architectural Styles

2.3.1 Layers

In the system design of Medieval Tower Defense, we have composed the system from two different layers, one of which is the User Interface layer, handling functionalities such as settings, starting the game, quitting the game, viewing highscores/help/instructions panels. The other layer handles all of the gameplay, in this layer, the “GameplayManager” holds all the smaller components such as the “TowerManager,” “EnemyManager,” “InputManager,” “SoundManager,” etc. Each of these smaller layers handle relevant game objects, therefore handling all of the objects.

2.3.2 Model View Controller

In this architectural design approach, the user interacts with the system using our controllers, such as listeners. Controllers manipulates the model of the game by making corresponding changes. Then, the model updates the view of the game. User sees this update and the cycle continues.

The underlying objects of our classes are accessed and controlled by our manager classes such as GameplayManager(controller). GUIManager class makes the desired changes to the view (view) and classes such as Tower and Invader corresponds to the model part of this structure.

2.4 Hardware / Software Mapping

During the implementation of this project, latest version of JDK will be used. As hardware configuration, a mouse and optionally a keyboard is required. The game is really small in sizes and requires really low CPU power, so it will be possible to run it on almost any PC with java installed. Java’s portability will be useful in this stage since it will help us to code it only once and it will be converted to corresponding forms in java virtual machine.

2.5 Persistent Data Management

Medieval Tower Defense will store the high scores in a .text file. When the player clicks on the HighScores tab, the game will read from the text file using Java’s File I/O libraries. New high scores are written on the .txt file when a player finishes a game with a high score and enters their name.

2.6 Access Control and Security

In order to access the game, downloading the .jar file is enough. There won't be any security checkings going on with the system. But the game requires no personal information except a nickname, therefore a security issue is not the case.

2.7 Boundary Conditions

Initialization

Medieval Tower Defense does not require any installation. Game will be in the .jar format.

Termination

The user will be able to terminate the game by clicking the 'back' icon in the main menu. During the game, pausing the game will provide this button also.

Error

If an error occurs during the gameplay, the data will be lost. If an error occurs while loading images or sounds, game will not be able to start.

3. Subsystem Services

This section provides the detailed information about the information of subsystems.

3.1 Design Patterns

Façade Design Pattern:

Façade design involves a single class that provides easy to use and understand methods required by the user, and abstracts the underlying system classes and their methods. Façade class can be considered as an interface for the client in order to acces the system. It is useful because it hides the underlying complexities and makes it easy to reflect any changes in the system.

In this design, façade pattern was used in order to connect different classes in a single manager class. For example, our GUIManager class is a Façade class that communicates with other subsystems and abstracts other UI classes.

3.2 Graphical User Interface Subsystem

UI subsystem makes the users able to iterate through the contents of the Medieval Tower Defense game and providing them graphical contents. It also manages the sections of the main menu. It also alerts the game manager object according to the choices of the user (such as settings).

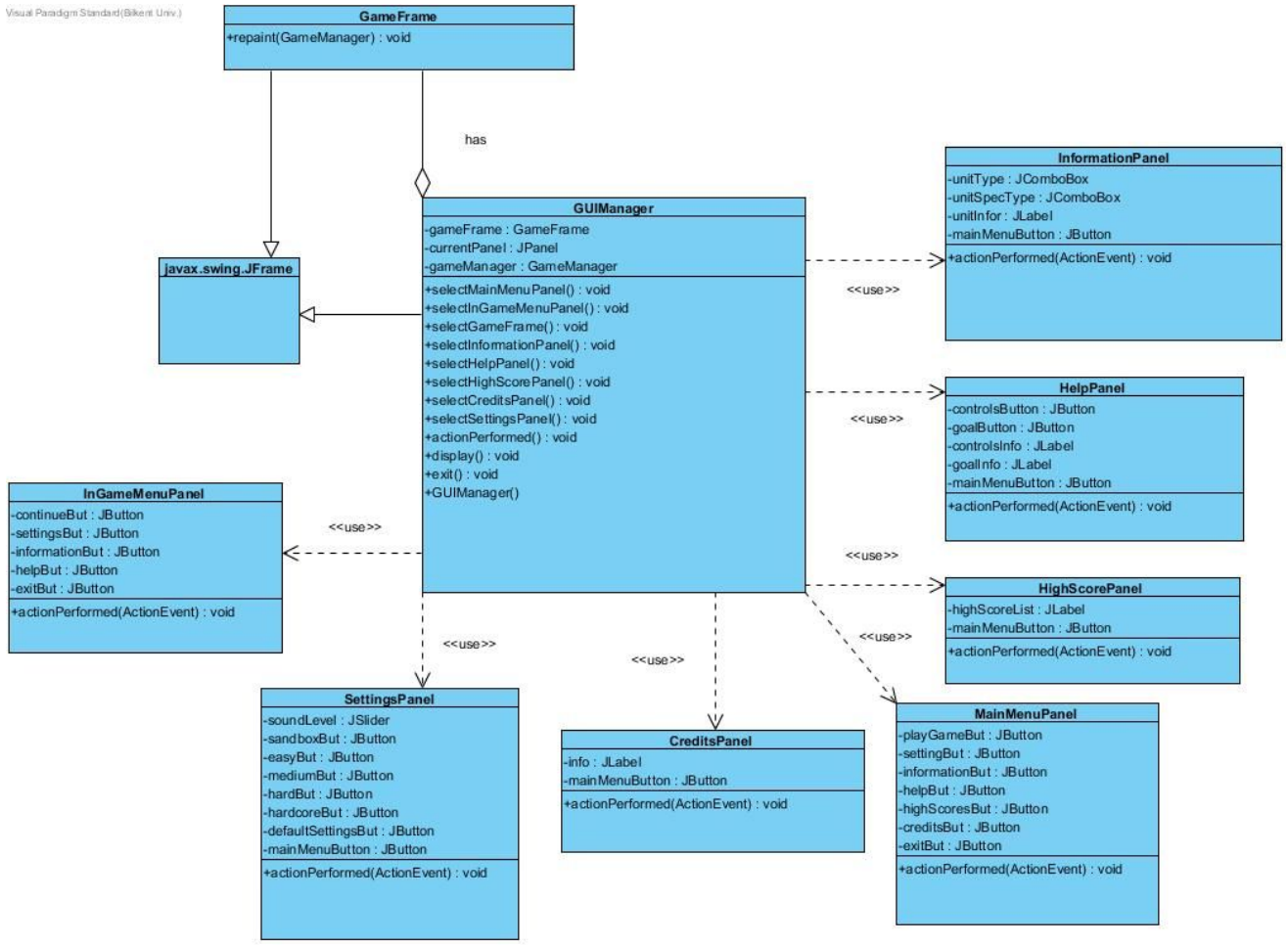


Figure 3.2.1 (User Interface Subsystem)

GUIManager Class

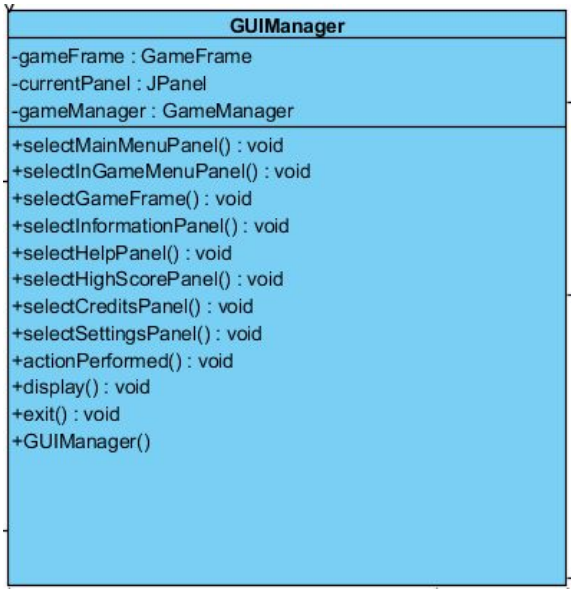


Figure 3.2.2 (GUIManager class)

Attributes:

private GameFrame gameFrame: This is the frame where the game is displayed

private JPanel currentPanel: This panel is the panel which the current selection is shown

private GameManager gameManager: This is the object which connects the game and the gui features. GUI gets updated from this object.

Constructors:

public GUIManager: Initilizes panels such as help panel, information panel and others. Also initilizes settings.

Methods:

public void selectMainMenuPanel(): This method changes the current view to the main menu screen.

public void selectInGameMenuPanel(): This method changes the current view to the in game menu screen.

public void selectGameFrame(): This method changes the current view to the game screen.

public void selectInformationPanel(): This method changes the current view to the information screen.

public void selectHelpPanel(): This method changes the current view to the help screen.

public void selectHighScorePanel(): This method changes the current view to the high score screen.

public void selectCreditsPanel(): This method changes the current view to the credits screen.

public void selectSettingsPanel(): This method changes the current view to the settings screen.

public void actionPerformed(): This method makes it possible to communicate with the user, interpreting his/her actions.

public void display(): This method is used by select methods above to display the current view.

public void exit(): This method terminates the game when called.

MainMenuPanel Class

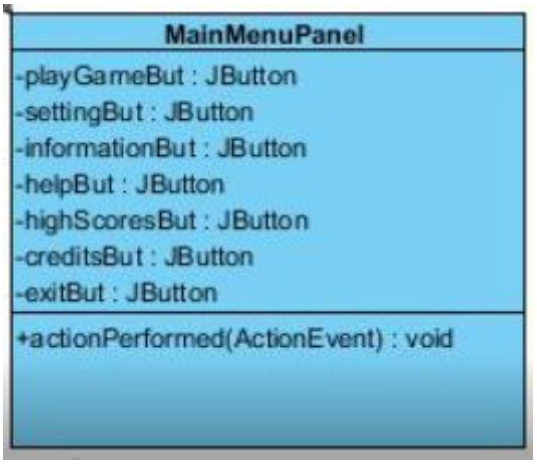


Figure 3.2.3 (MainMenuPanel Class)

Attributes:

private JButton playGameBut: This attribute provides a button to the screen. When this button is selected by the user, game starts by selecting the game frame.

private JButton settingBut: This attribute provides a button to the screen. When this button is selected by the user, settings screen appears by selecting the settings panel.

private JButton helpBut: This attribute provides a button to the screen. When this button is selected by the user, help screen appears by selecting the help panel.

private JButton informationBut: This attribute provides a button to the screen. When this button is selected by the user, information screen appears by selecting the information panel.

private JButton helpBut: This attribute provides a button to the screen. When this button is selected by the user, help screen appears by selecting the help panel.

private JButton highScoresBut: This attribute provides a button to the screen. When this button is selected by the user, high scores screen appears by selecting the high scores panel.

private JButton creditsBut: This attribute provides a button to the screen. When this button is selected by the user, credits screen appears by selecting the credits panel.

private JButton exitBut: This attribute provides a button to the screen. When this button is selected by the user, exit() method of the GUIManager is called and the game ends.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

GameFrame Class

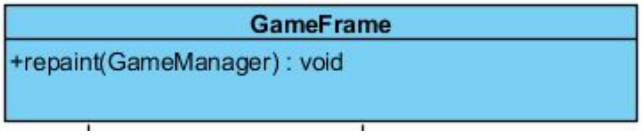


Figure 3.2.4 (GameFrame Class)

Methods:

public void repaint(GameManager): If the selectGameFrame method is called on the GUIManager class, GameFrame keeps painting the game objects on their corresponding positions. This class uses the gameManager object from the GUIFrame, taking arguments from it.

InGameMenuPanel Class

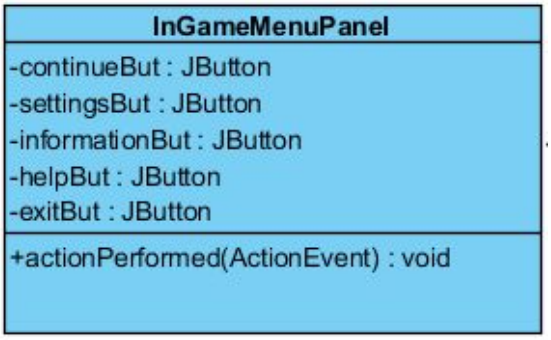


Figure 3.2.5 (InGameMenuPanel Class)

Attributes:

private JButton continueBut: This attribute provides a button to the screen. When this button is selected by the user, game frame is selected and game continues.

private JButton settingsBut: This attribute provides a button to the screen. When this button is selected by the user, settings panel is selected and settings screen is provided to the user.

private JButton informationBut: This attribute provides a button to the screen. When this button is selected by the user, information panel is selected and information screen is provided to the user.

private JButton helpBut: This attribute provides a button to the screen. When this button is selected by the user, help panel is selected and help screen is provided to the user.

private JButton exitBut: This attribute provides a button to the screen. When this button is selected by the user, exit() method of the GUIManager is called and the program terminates.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

SettingsPanel Class

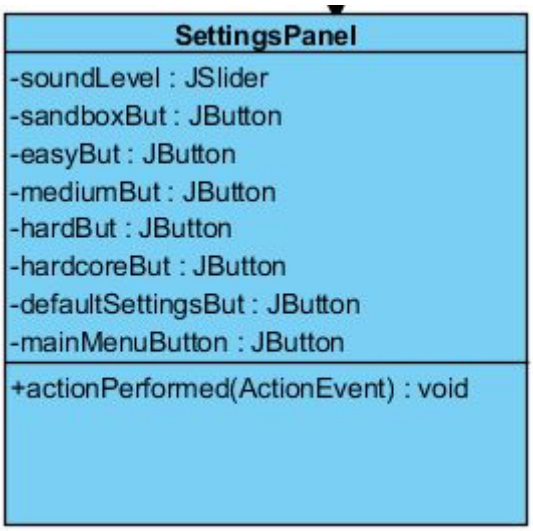


Figure 3.2.6 (SettingsPanel Class)

Attributes:

private JSlider soundLevel: This attribute provides a JSlider to the settings panel. By using it, user will be able to change the sound level to his/her desires.

private JButton sandboxBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the difficulty level to sandbox.

private JButton easyBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the difficulty level to easy.

private JButton mediumboxBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the difficulty level to medium.

private JButton hardBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the difficulty level to hard.

private JButton hardcoreBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the difficulty level to hardcore.

private JButton defaultSettingsBut: This attribute provides a button to the settings panel. When selected, calls the corresponding method from the gameplaymanager and sets the current settings back to default.

private JButton mainMenuButton: This attribute provides a button to the settings panel. When selected, calls the selectMainMenu method of the GUIManager and changes the view.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

CreditsPanel Class

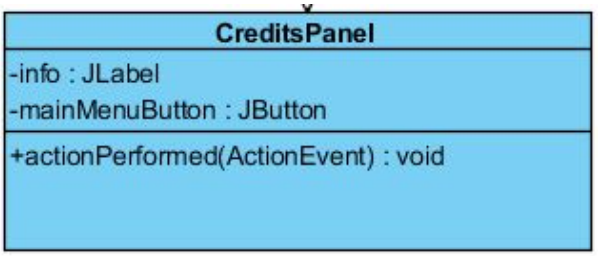


Figure 3.2.7 (CreditsPanel Class)

Attributes:

private JLabel info: This attribute provides a text label to the credits panel. It contains the credits as text format.

private JButton mainMenuButton: This attribute provides a button to the credits panel. When selected, calls the selectMainMenu method of the GUIManager and changes the view.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

HighScorePanel Class

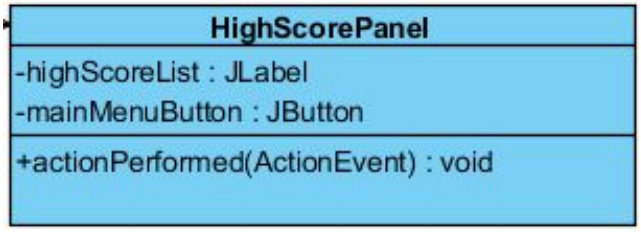


Figure 3.2.8 (HighScorePanel Class)

Attributes:

private JLabel highScoreList: This attribute provides a text label to the highScorePanel. It contains the high score information.

private JButton mainMenuButton: This attribute provides a button to the high score panel. When selected, calls the selectMainMenu method of the GUIManager and changes the view.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

HelpPanel Class

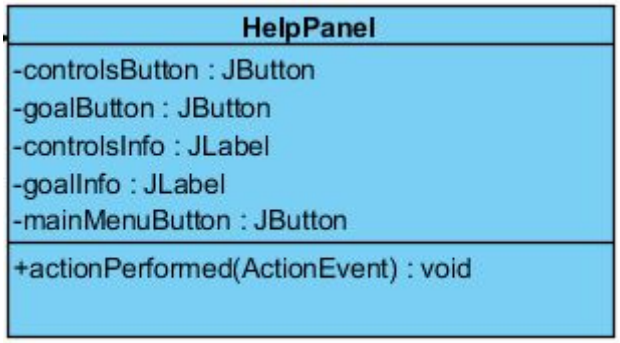


Figure 3.2.9 (HelpPanel Class)

Attributes:

private JButton controlsButton: This attribute provides a button to the help panel. When selected, changes the information displayed on the screen from goal info to controls info.

private JButton goalButton: This attribute provides a button to the help panel. When selected, changes the information displayed on the screen from controls info to goal info.

private JLabel controlsInfo: This attribute provides a JLabel to the help panel. It contains the corresponding controls information about the game.

private JLabel goalInfo: This attribute provides a JLabel to the help panel. It contains the corresponding information about the aims of the game.

private JButton mainMenuButton: This attribute provides a button to the help panel. When selected, calls the selectMainMenu method of the GUIManager and changes the view.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

InformationPanel Class

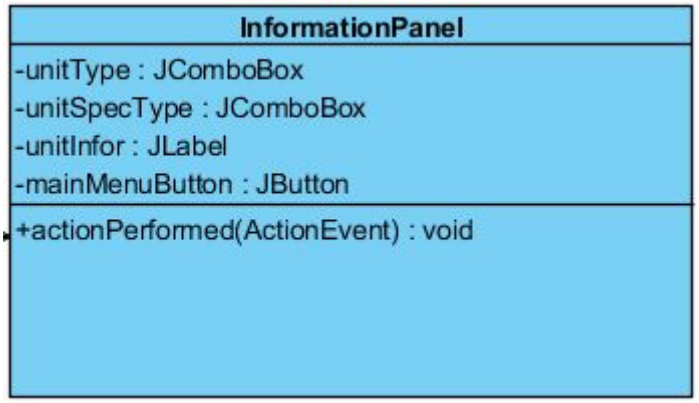


Figure 3.2.10 (InformationPanel Class)

Attributes:

private JComboBox unitType: This attribute provides a combo box to the information panel. User can select the unit type from this combo box (such as enemies, bosses, towers). This selection changes the corresponding options in the unitSpecType combo box.

private JComboBox unitSpecType: This attribute provides a combo box to the information panel. User can select the units' specific type from this combo box (for example for the enemy selection from unitType, the options avaible in the unitSpecType combo box will be Chengiz Khan, Knight, Trojan Horse etc) This selection changes the corresponding information in the unitInfor label.

private JLabel unitInfor: This attribute provides a text label to the information panel. According to the user's selections, corresponding information about the specific unit appears here.

private JButton mainMenuButton: This attribute provides a button to the information panel. When selected, calls the selectMainMenu method of the GUIManager and changes the view.

Methods:

public void actionPerformed(ActionEvent): This method gets alerted when user selects a button and makes acts in the program corresponding to the selection.

3.3 Gameplay Subsystem Interface

In this subsystem, we have the facade class "Map" which handles the communication between the "Grid", the "Player", and all instances of "Enemy" and "Tower" classes. Since this class has the reference to the "Grid" as an attribute, it also has access to all of the

“gridSlot”s, and therefore all of the objects on those “gridSlot”s, which makes it easier for us to manipulate objects without getting redirected too much in between classes/interfaces.

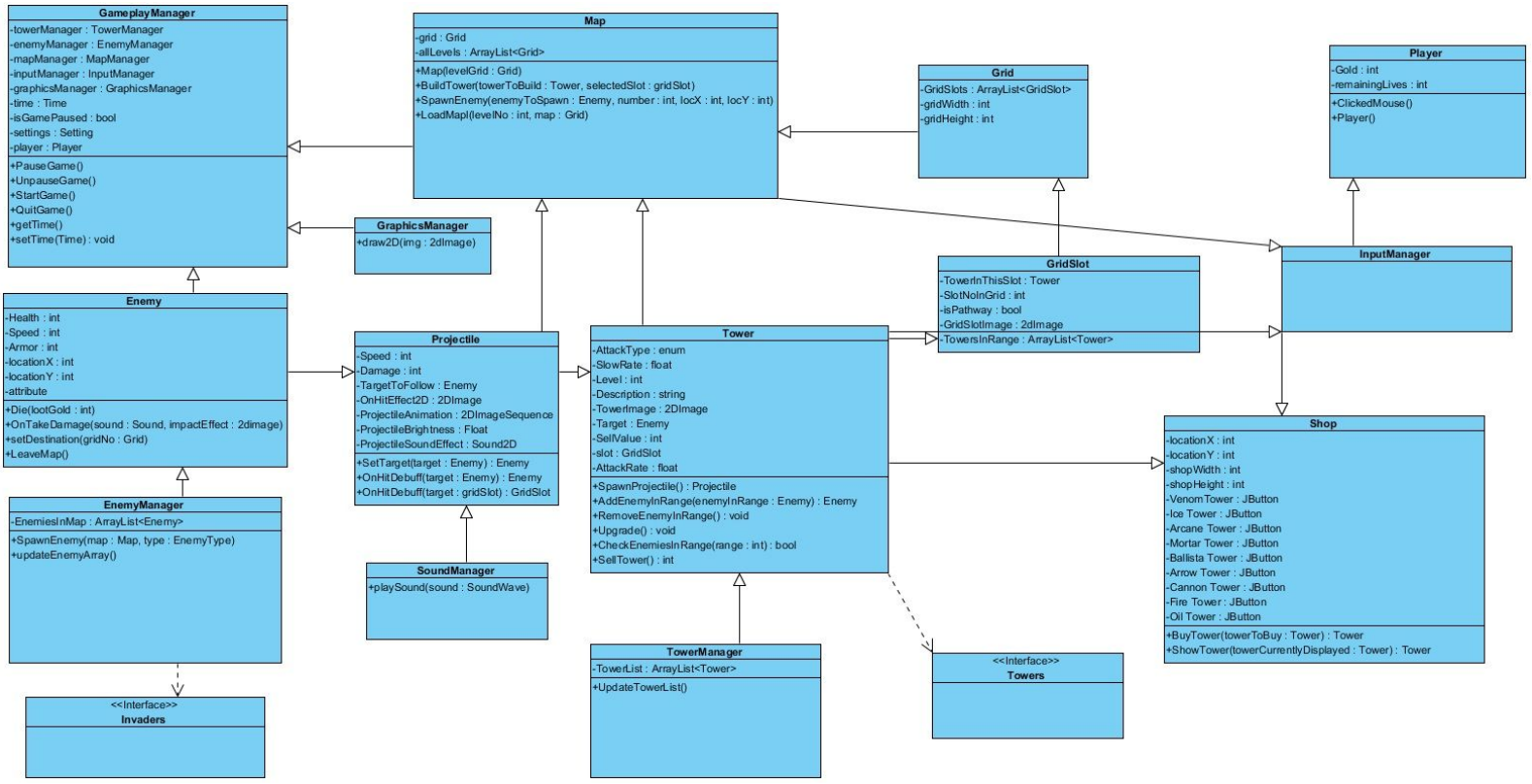


Figure 3.3.1 Gameplay Objects Subsystem

GameplayManager Class

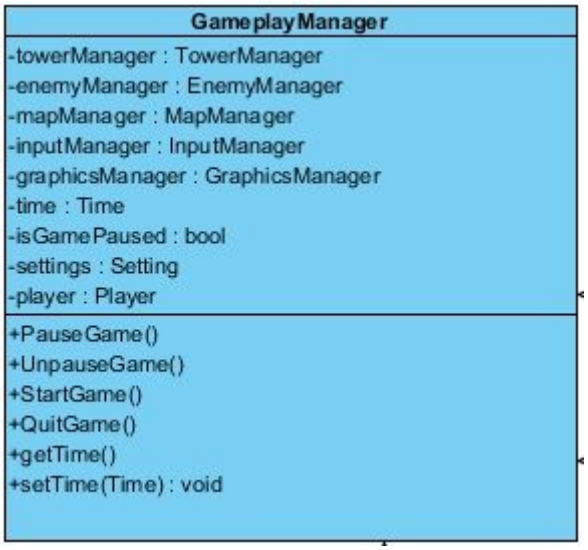


Figure 3.3.2 (GameplayManager Class)

Attribute:

private TowerManager towerManager: A reference to the TowerManager that holds all the tower instances in an array.

private EnemyManager enemyManager: A reference to the EnemyManager that holds all the enemy instances in an array.

private MapManager mapManager: A reference to the MapManager that holds a reference to the map.

private InputManager inputManager: Listens for mouse click events from the player.

private GraphicsManager graphicsManager: Draws all the 2D images and animations.

public Time time: This value keeps a track of time in order to calculate the score of the player at the end of the game.

public bool isGamePaused: Whether the game is paused.

private Setting settings: A reference to the class where the settings ‘sound volume, difficulty(sandbox, medium, easy, hard, hardcore)’ are kept.

Methods:

public PauseGame(): Game may be paused anytime by the player if Pause Break button is pushed. When paused, player will be directed to in-game menu.

public UnpauseGame(): If player is in in-game menu, or if the game is paused, player may unpause the game by hitting the button Pause Break once again.

public StartGame(): New game sequence will be initiated once StartGame button is hit.

public QuitGame(): Current game progress may be abondened by hitting the QuitGame button, which is available at in-game menu

public getTime(): Calculated time can be retrieved with getTime method.

public void setTime(): Time is calculated with this method. It measures the player's speed of destroying the enemy wave. Quicker the player destroys the wave, better the score will be.

Map Class

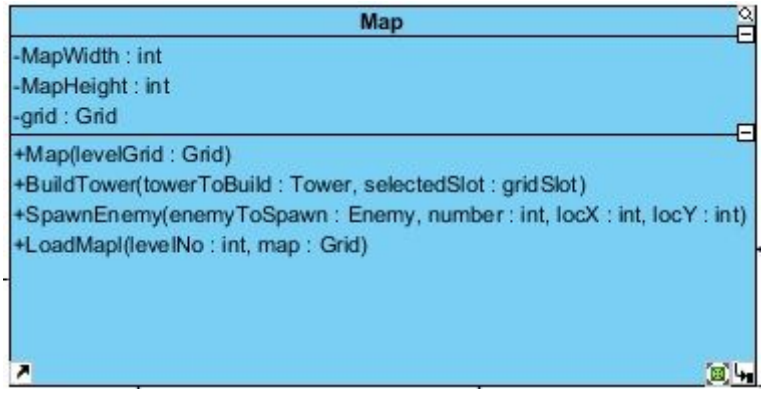


Figure 3.3.3 (GameplayManager Class)

Constructors:

public Map(levelGrid : Grid): Constructs a game map with the given "Grid" object. Since the graphics are just seamless images of each grid slot piled up together, no further work is needed on this class for the map construction, that is handled on the "Grid" object.

Methods:

public Tower BuildTower(towerToBuild : Tower, selectedSlot : GridSlot): When the player selects a tower from the shop and clicks on the game map, this function is called to build a tower on the selected "gridSlot," if the selected gridSlot is available. The availability of the gridSlot is checked by first looking at "isPathway" boolean value of the gridSlot object. If the gridSlot is not a pathway, and if it has no tower object on it, then the tower is built, otherwise the player click is dismissed.

private Map LoadMap(levelNo : int): Can load a map by the level number.

public Enemy SpawnEnemy(enemyToSpawn : Enemy, number : int, locX : int, locY : int): Spawns an enemy

GraphicsManager Class

Attributes:

draw2D(img : 2dImage): This method draws the image that is passed to it by parameter. Since GraphicsManager has access to the GameManager, it can draw the needed objects. For instance, all the towers on the map are stored in the TowerManager, and the GameManager has access to the TowerManager, therefore the GraphicsManager can also access all the individual towers and draw them.

Grid Class

Figure 3.3.4 (Grid Class)

Attributes:

GridSlots: All the slots in the grid stored in the Grid Class. There is only one “Grid” class instance for each level/map.

gridWidth: Stores the width of the grid.

gridHeight: Stores the height of the grid.

GridSlot Class

Figure 3.3.5 (GridSlot Class)

Attributes

private Tower TowerInThisSlot: Stores the tower in this slot, if this attribute is NULL and “isPathway” is “FALSE,” then the player can build a tower on this “gridSlot.”

private int SlotNoInGrid: The slot number that this gridSlot corresponds to in the whole Grid.

private bool isPathway: If this boolean is true, than this gridSlot is where the enemies can move to, and the player cannot build a tower on this slot. If this boolean is false, the player can build a tower on this slot and the enemies cannot move through this gridSlot.

private 2dImage GridSlotImage: The 2d image to draw for a particular gridSlot instance. These images are seamless so that the same type of floors in the game work together.

private ArrayList<Tower> TowersInRange: This attribute holds the towers that can attack the enemies in this gridSlot.

Tower Class

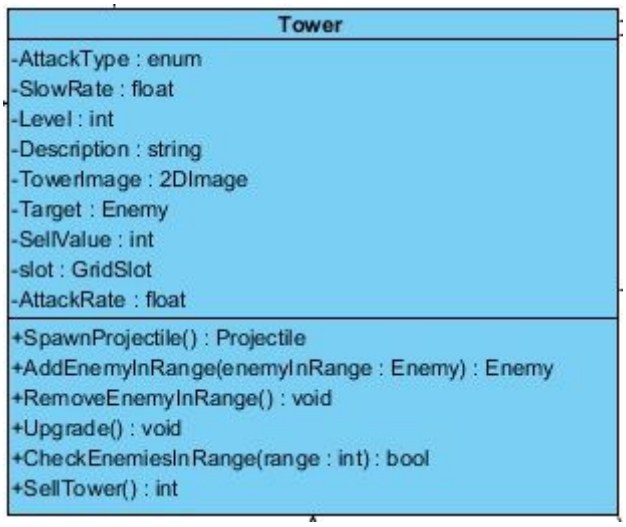


Figure 3.3.6 (Tower Class)

(Moved Function) CheckEnemiesInRange(range : int) was a function of the “Tower” class, but we decided to move it to the gridSlot class for performance reasons, here’s a short explanation:
“CheckEnemiesInRange” is a function that is called every frame for each tower, which can result in framerate drops because of the CPU time when there are many towers. So the CPU time is the downside of placing this function on this class. The good side is to this method is that towers can take float values as range and the game would benefit a bit more variety.

Our decision was to define the tower ranges as integers which correspond to gridSlots. So that a tower can hit the 8 “gridSlots” around it, or 16, 24 and so on… For this way of handling the targeting, when each tower is built, or upgraded in range, it checks all the gridSlots in range and adds a reference of itself to the “TowersInRange” attribute of the corresponding gridSlots.(notice that this action is called only 1 time either when a tower is built or upgraded in range) The “gridSlots” that have the boolean value “isPathway” set to true, check for the enemies inside them and sometimes apply debuffs like slow etc. After doing this check, the gridSlot sends references of those enemies each tower in the array “TowersInRange.” And that is how the targets are given to the towers.

Attributes:

private enum AttackType: There are attack type enumerations such as splash, single target, orb effect.

private float SlowRate: If this tower instance doesn’t have the ability to slow enemies, the slowRate is 0, if it has, than the slow rate is more than 0.

private int level: The level of the tower, can be upgraded.

private string description: The description of the tower, giving short info for a quick read.

private 2dImage TowerImage: The image of this tower, GraphicsManager gets a reference to this image via the “GameManager->TowerManager->ArrayList<Towers>->this tower instance” route.

public Enemy Target: The target that this tower instance attacks. This reference is passed to the projectiles spawned by this tower.

private int sellValue: The sell value of this, usually half of the buy value.

public int slot: A reference to the slot that this tower instance is built onto.

private float AttackRate: The attack speed(projectile spawned per second) value of this tower instance.

Methods:

private SpawnProjectile(): As soon as the “Target” attribute of this class changes from “NULL” to a valid object, the SpawnProjectile() function starts getting called every x seconds depending on the AttackRate float value.

public AddEnemyInRange(enemyInRange : Enemy): Sets the “Target” attribute of this class to the parameter “enemyInRange.”

public RemoveEnemyInRange(): Sets the “Target” attribute of this class to “NULL.”

private Upgrade(): Upgrades the tower, increasing its’s attack rate, damage, and for some upgrades it’s range as well.

private SellTower(): Sells this tower and destroys this tower instance.

Deleted

Shop Class

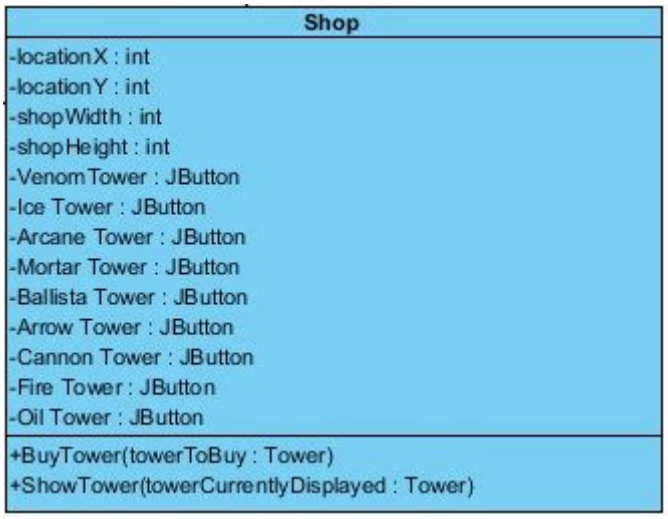


Figure 3.3.7 (Shop Class)

Attributes

Tower Types

private JButton Arrow Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Arrow Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Cannon Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Cannon Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Ice Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Ice Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Fire Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Fire Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Oil Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Oil Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Poison Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Poison Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Arcane Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Arcane Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Ballista Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Ballista Tower” on a “GridSlot” the is not an “isPathway”.

private JButton Mortar Tower: If the player has enough gold, when clicked, this button gives the player the ability to place an “Mortar Tower” on a “GridSlot” the is not an “isPathway”.

Methods

public BuyTower(towerToBuy : Tower): Player will choose the tower to be bought via the side menu. towerToBuy will be the tower that’s been chosen by player. If player has enough gold, player will be able to click on tower and place it to a specified grid location. Yet if player lack of gold, BuyTower will not be activated and player will be warned instead.

public ShowTower(towerCurrentlyDisplayed : Tower):
The behaviours that player can apply are buy tower, and sell tower. These are defined in functions called BuyTower(). There is also a constructor that defines the overall Player object and sets default attributes to this object, such as life count, gold, and username. BuyTower() function allows player to purchase one of the towers and place it to a specified grid. Player will be able to choose a tower from the shop menu, in which all towers and their costs are displayed. Once the player has clicked a tower that he is able to buy, player will click again onto a gridSlot and BuyTower() function will be concluded. If the player clicks on a tower that he is unable to buy, there will be a pop-up informing the player about that matter.

Player Class

Figure 3.3.8 (Player Class)

Attributes:

private int Gold: The amount of gold that the player has which is used for buying new towers. For each enemy eliminated, this attribute increases.

private int remainingLives: The remaining lives of the player,

Constructors:

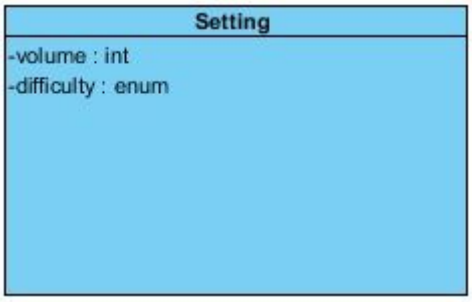
public Player(): Sets the default values for the “remainingLives” and the “gold.”

Methods:

public void ClickedMouse(): Method that is executed when the player has hit the left mouse button.

There is player class which describes the certain behaviour and attributes of the user. Attributes that are declared will also describe the player’s momentary statistics. As ordered, the attributes are username, gold, and remaining lives. Gold is an integer value and the currency of player. It allows player to buy new towers, and it’s gathered by player as the invaders are vanquished. Remaining lives is another integer value which determines the condition of player, by each trespassing enemy the remaining lives of the play drops by 1. If it drops down to 0, the player will lose the game.

Setting Class



Attributes:

public int volume: Keeps the volume proportion as an integer over 100 max value.
private enum difficulty: Difficulty types are one of the 5 enumerated types: “sandbox, easy, medium, hard, hardcore.”

Projectile Class

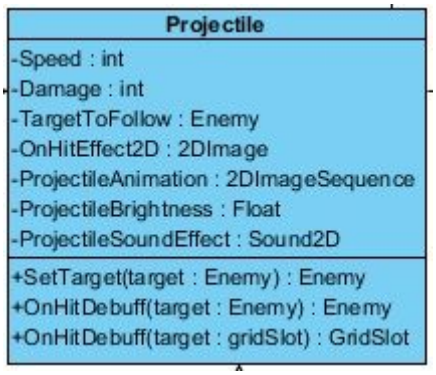


Figure 3.3.9 (Projectile Class)

Attributes:

private int Speed: This attribute indicates the speed of projectile. Each tower has different projectile speed and so different attack frequency.
private int Damage: There is also damage attribute which remarks the damage output of the projectile.

private Enemy TargetToFollow: Projectile's target is identified with TargetToFollow attribute. Each tower can target one enemy at a time.

public 2DImage OnHitEffect2D: When enemies got hit by projectile, there will be a special effect generated and this effect is denoted with OnHitEffect2D attribute.

public 2DImageSequence ProjectileAnimation: This image sequence is drawn by the graphicsManager.

public float ProjectileBrightness: This value is used when drawing the projectile animation and it determines the brightness of the projectile. The brightness usually increases when the power of the projectile is stronger.

public Sound2D ProjectileSoundEffect: The sound effect to be played when the projectile hits an enemy.

Methods:

public Enemy SetTarget: Sets the "TargetToFollow" attribute of this class. This method is called by the tower instance that has spawned this class.

private Enemy OnHitDebuff: This is the method for applying the minus armor, slowRate orb effects on the enemies that are hit by this projectile instance.

private GridSlot OnHitDebuff: This is the method for applying the minus armor, slowRate orb effects on a particular gridSlot so that all the enemies that are in that slot are affected..

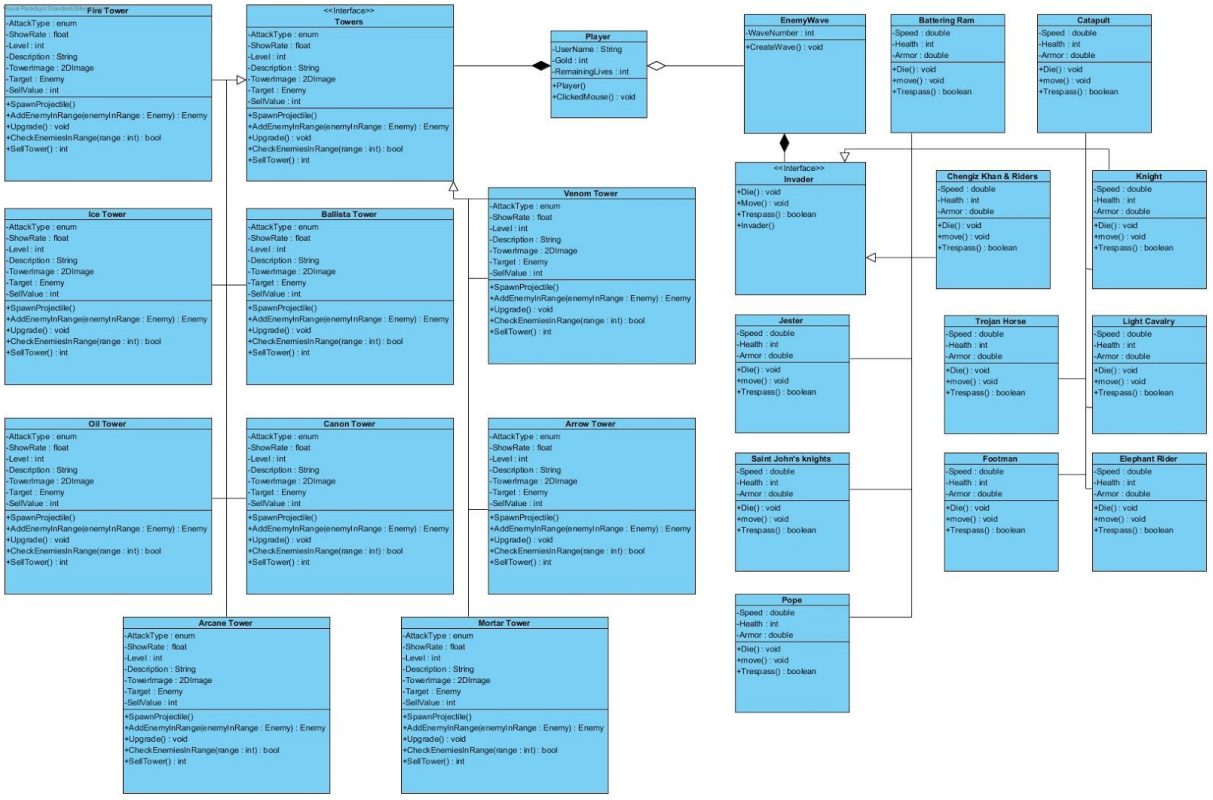


Figure 3.3.10 (Interface Diagram)

Towers<Interface>

Towers class will serve as an interface for the specialized tower classes. The concepts of inheritence and polymorphism will take place so that there will be towers with different characteristics and different attributes. Yet at the end, behaviours and attribute types of all towers will be same. Attributes specified until now are Description, Damage, Cost, and **UpgradeLevel.** Description is a string variable that describes the characteristic of the tower that player points on. It helps player to choose the most fitting tower for the situation that player needs to handle. Damage is the raw power that towers beholds It is identified as a double variable, because the damage will vary according to the armor of the target invader. Cost variable will be on charge when the player decides to buy a new tower. Towers which have the higher capability of crowd control or the ones which can take individual targets easily will be more expensive, so buying a new tower can be a tactical aspect of the game. Upgrade level is the last attribute to be represented and it will deny player from upgrading the tower forever. Towers that are cheaper will have more upgrade levels, yet upgrading them will cost less when compared to more expensive towers.

In the current state, towers have two behaviours and these are Shoot(), and UpgradeLevel(). Both of them have the return type of void. Shoot() function acts differently

for each specialized tower. This function certainly describes the characteristic of each specialized tower. The splash damage made, speed and armor debuffs that's been caused, raw damage output, and damage frequency are all handled in this function for each different polymorphic tower class. Last function, UpgradeLevel() basically handles the constraint of upgrade limit specified for different types of towers. It permits or doesn't let player to upgrade the tower that's been chosen.

Methods:

public Projectile SpawnProjectile(): This function acts differently for each specialized tower. Certainly, it describes the characteristic of each specialized tower. The splash damage made, speed and armor debuffs that's been caused, raw damage output, and damage frequency are all handled in this function for each different polymorphic tower class.

public bool CheckEnemiesInRange(int range): Serving as an auxiliary function, CheckEnemiesInRange will function as the sensor of tower. The integer parameter denoted as range will check the enemies around in each time interval.

public Enemy AddEnemyInRange(Enemy enemyInRange): In case of an enemy detection, AddEnemyInRange will notify tower about enemy activity, and will let tower to activate the function responsible for taking down the invaders.

public void Upgrade(): It basically handles the constraint of upgrade limit specified for different types of towers. It permits or doesn't let player to upgrade the tower that's been chosen.

public int SellTower(): When player decides to sell the chosen tower, SellTower function returns the amount of gold that's specified for the tower to be sold. If tower has been upgraded previously, player will receive more gold by selling it.

Attributes:

private enum AttackType: Attack type may be splash damage or single target.

private float SlowRate: Slow rate to be applied to the enemy.

private int Level: This attribute will indicate the current level of tower

private String Description: Each tower will have a description indicating its certain characteristics with a brief description

private 2DImage: There will be an image display of each tower. Every tower will have different image.

private Enemy: This attribute will identify the enemy targeted by the tower.

private Int SellValue: Sell value remarks the chosen tower's value in case of sell consideration

Classes derived from Towers Interface

ArrowTower Class:

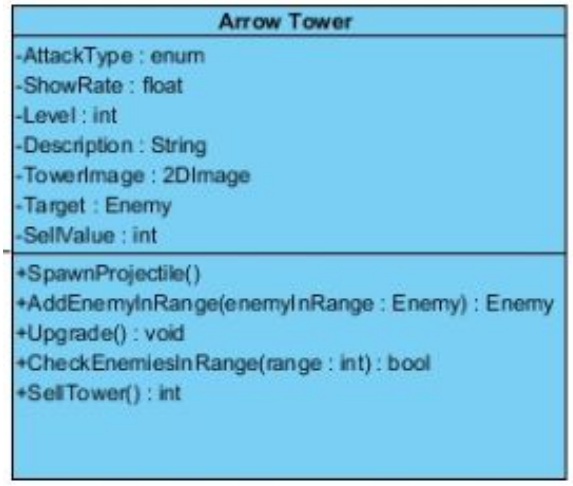


Figure 3.3.11 (Interface Diagram)

Constructor:

public ArrowTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

CanonTower Class:

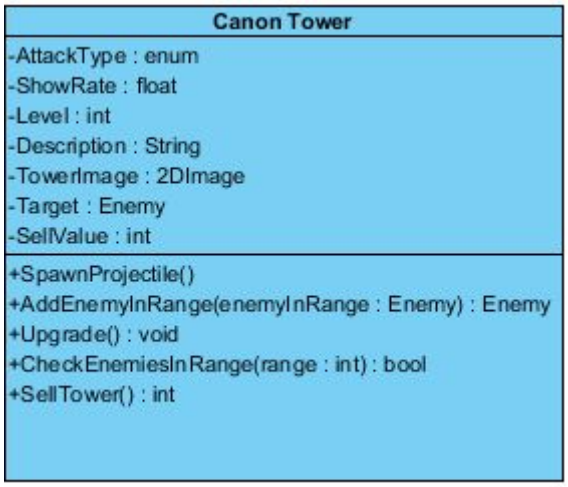


Figure 3.3.12 (Interface Diagram)

Constructor:

public CanonTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

IceTower Class:

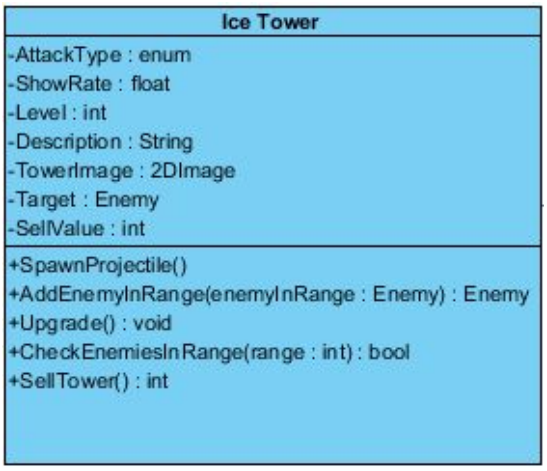


Figure 3.3.13 (Interface Diagram)

Constructor:

public IceTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

FireTower Class:

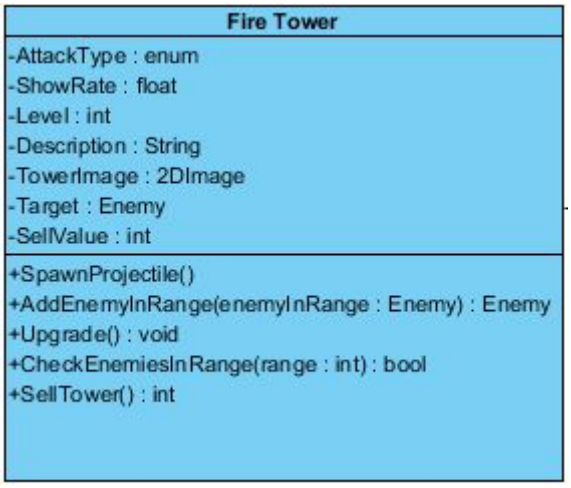


Figure 3.3.14 (Interface Diagram)

Constructor:

public FireTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

OilTower Class:

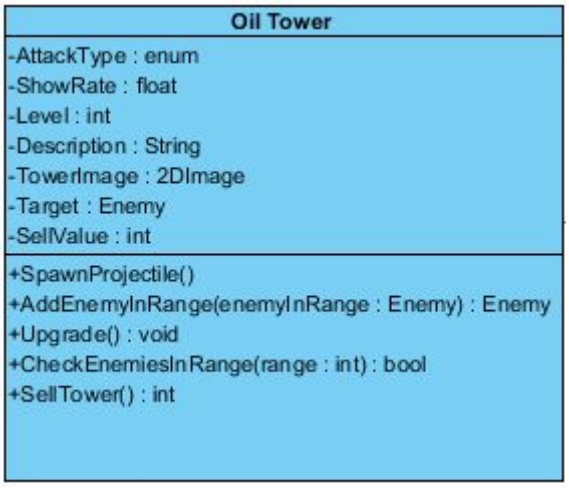


Figure 3.3.15 (Interface Diagram)

Constructor:

public OilTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

PoisonTower Class:

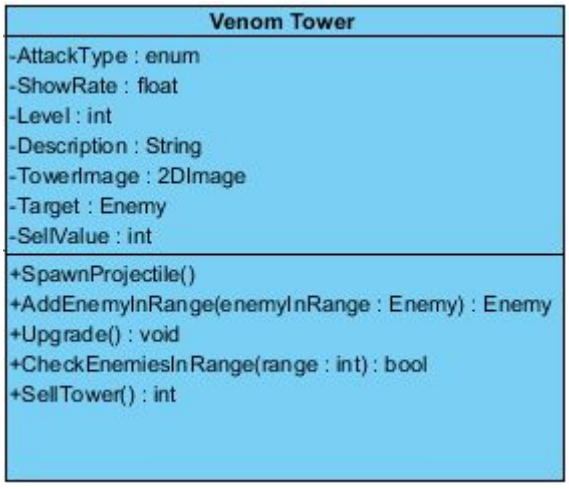


Figure 3.3.16 (Interface Diagram)

Constructor:

public PoisonTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

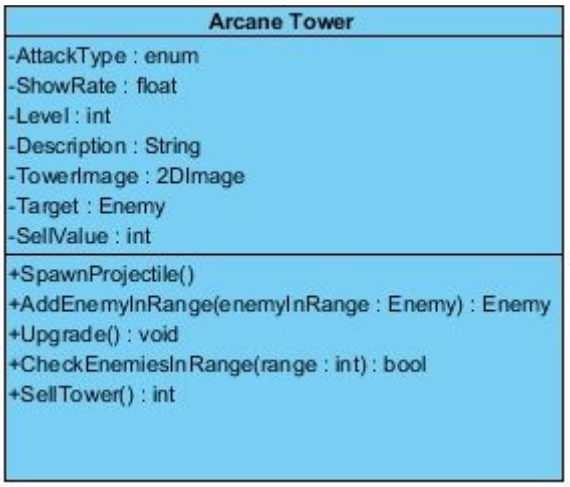


Figure 3.3.17 (Interface Diagram)

Constructor:

public ArcaneTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

BallistaTower Class:

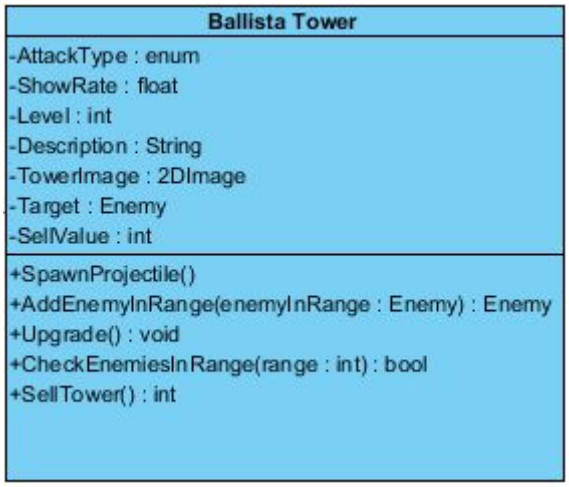


Figure 3.3.18 (Interface Diagram)

Constructor:

public BallistaTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

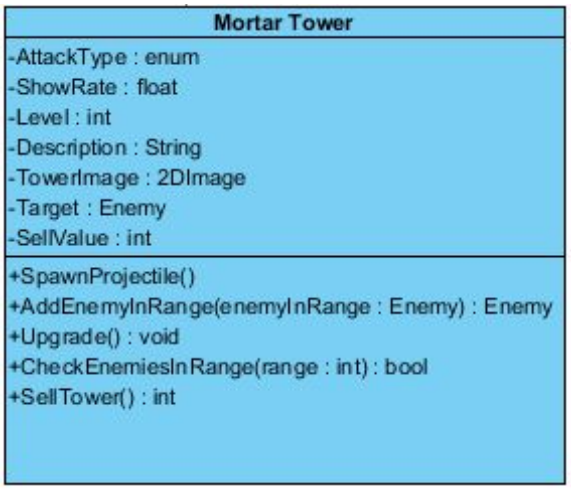


Figure 3.3.19 (Interface Diagram)

MortarTower Class:

Constructor:

public MortarTower: The constructor of this class. The proper image and attributes will be specified with the constructor.

Invader

Invaders are individual unit types that assemble the body of an enemy wave. Invader class serves as an interface of polymorphic enemy types that have different attributes. Main goal of an enemy invader is to survive the defense mechanisms that's been set by player and move along to the player castle. Attributes identified for the Invader class are speed, health, and armor. As default, all of these attributes are set in the constructor of different enemy types. Speed is an integer value and it declares how fast the invader is. Health is another integer value and it's the measure of how much can invader endure to the damage input. Once the invader loses all health points, player will earn gold. Armor is a double type attribute and it reduces the damage input with the multiplier identified for each different invader.

Methods:

public void Die(): It informs the invader's progress to the player. If Die function is been activated in the case of health bar drops zero, player will receive money in conclusion.

public void Move(): Another void function Move() will command invader to move on as long as the invader is not dead. In each frame, invader will move from one place to another according to the speed value.

public bool Trespass(): Trespass() will be active if invader manages to cross aside the player keep. In conclusion the player's health bar will be reduced by one.

Attributes:

private double Speed: Speed attribute will differ for any type of enemy. It basically describes how speed an enemy is.

private int Health: Health value shows how much damage an enemy can endure.

private double Armor: Armor will split the damage input of an enemy individual and more armor means less health points to be reduced.

Classes Derived From Invader Interface

Catapult

Figure 3.3.20 (Interface Diagram)

Constructor:

public Catapult(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Chengiz Khan & Riders

Figure 3.3.21 (Interface Diagram)

Constructor:

public ChengizKhanAndRiders(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Footman

Figure 3.3.22 (Interface Diagram)

Constructor:

public Footman(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Jester

Figure 3.3.23 (Interface Diagram)

Constructor:

public Jester(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Knight

Figure 3.3.24 (Interface Diagram)

Constructor:

public Knight(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Light Cavalry

Figure 3.3.25 (Interface Diagram)

Constructor:

public LightCavalry(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Pope

Figure 3.3.26 (Interface Diagram)

Constructor:

public Pope(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Battering Ram

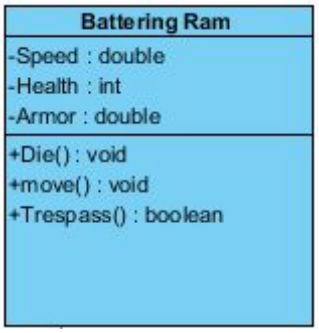


Figure 3.3.27 (Interface Diagram)

Constructor:

public BatteringRam(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Trojan Horse

Figure 3.3.28 (Interface Diagram)

Constructor:

public TrojanHorse(): The constructor of this class. The proper image and attributes will be specified with the constructor.

Saint John's Knights

Figure 3.3.29 (Interface Diagram)

Constructor:

public SaintJohnKnights(): The constructor of this class. The proper image and attributes will be specified with the constructor.

EnemyWave

The invaders will come as divisions, and these divisions are called as enemy waves. There will be a certain number of waves and this number will be recorded in WaveNumber attribute.

Methods:

public void CreateWave: This method acts if all individual invaders are dead and WaveNumber constraint allows to new waves be generated.