BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS 319 FALL–2009

GROUP 3 - TOWER DEFENSE GAME

FINAL REPORT

with revised versions of ANALYSIS REPORT and DESIGN REPORT

Aytek Aman	20601029	section 1	aman@ug
Ezgi Mercan	20501196	section 1	e_mercan@ug
Semih Energin	20603671	section 1	energin@ug
Tuğçe Yurtseven	20600837	section 1	g_yurtseven@ug

Table of Contents

1. Introduction

"The Last Defense" (or with its long formal name, "Chronicles of Sies Wars: The Last Defense") is a tower defense game, a relatively young genre of strategy games. Tower defense games usually provide a quick fun to casual gamers with the rapid movement of the game and quick strategically thinking required to pass levels. "The Last Defense" takes place in a fictional world called "Sies"; a world created by the developers of the game and has many similarities to other Tolkienesque fantasy fiction worlds. In this game, players create a character, which is the commander of this defensive war, and build defensive formations to prevent enemies to breach defenses.

Unlike most tower defense games, "The Last Defense" is neither a casual game nor a web browser game; it is designed for hardcore gamers, both for personal computer users and for console owners. It provides more than satisfactory graphical features, by utilizing both three dimensional models and a two dimensional game play.

The aim of this game is that building defensive formations to block enemies to breach defenses. First of all, user creates a profile by selecting which represents commander of the defensive game backgrounds and characteristic features. Player selects background and determines the path before playing the game. During the game, player sells towers and places them in strategic positions to eliminate the enemies by using bullets. The more levels are passed, the harder killing the enemies.

This report consists of two sections; the revised version of analysis report and the design report of the system to be developed.

The first part includes the problem statement, a requirement analysis supported by functional and non-functional requirements, use case models that are explained further with scenarios and user interfaces to visualize requested product. Also, analysis models (both object and dynamic) are also given, including a domain lexicon, class diagram, state charts and sequence diagrams that are corresponding with previously given scenarios.

The design report of the tower defense game consists of 8 parts:

In the first part the design goals of tower defense game is explained in detailed way. Our system gives priority to good documentation, reliability, high performance, ease of using ,ease of remembering, minimum number of errors, increased productivity and rapid development. Here, why these features are very major for the system are explained. Further, the trades of the system are described in terms of cost (High Performance, Reliability),functionality (Ease of Use, Ease of Learning) and rapid development (Good Documentation, Minimum Number of Errors)

In the second part the decomposition of systems are given in detailed way. The system is divided into 5 parts mainly as GameLogic Subsystem, Input Management Subsystem, Sound Management Subsystem, Viewer Subsystem and Data Management Subsystem. GameLogic

Subsystem is decomposed in two sections as Logic Manager Subsystem and Profile Management System. Viewer system is divided into Game Screen and Screen Elements subsystems.

In the third part, two architectural patterns are implemented to display the structure of the subsystem. Firstly, layer pattern design is implemented with bottom-up approach and described in detailed way. Secondly Model View Control pattern are implemented. In addition to this which classes belongs to which concepts are shown.

In the fourth part, the mapping between hardware and our system is described and deployment diagram is created .

In the fifth part, how to manage persistent data is explained. Here, the choice of text file is explained and described in detailed way with respect to easy operation, the size, raw data, keeping for a short time, density

In the sixth part, the access control and security of the system are depicted.

In the seventh part, the global software control of the system are described. Global control flow is event-driven in Tower Defense Game. Consistently with its Model-View-Controller architecture, program waits for an event, in this case a key stroke on the keyboard or a mouse click inside the buttons; and updates the views. The control flows are described for each subsystem. In addition to this, the structure of the program is explained

In the last part of the report is told boundary conditions of the systems are depicted. The boundary conditions are analyzed as 3 parts initialization, termination and failure.

2. Problem Statement

Tower defense games are relatively younger than other game genres, especially compared to other strategy games. Contemporary games are each others clone and there are only few original products which distinguish from the others. Thus, a game to be created should distinguish itself from the others. A combination of the implementation of the developments in the game industry and “old school” features which are generally approved by the gamer society should be included in the game.

Contemporary tower defense games are almost always casual games, which are targeted to at a mass audience of casual gamers, who do not have much sense of professional gaming and seek to be entertained by simple games in short time game-plays and do not have any sense of commitment to to the game itself. Such games are simple to understand, have basic rules and lack many features that a game can provide; a sense of long-term commitment, depth of background scenario, re-playability. Having personalized profiles, having a character in the game, mixing game dynamics to a meaningful story and game-play are commonly used methods, and this game should use them properly.

The best and well known products of the interactive entertainment industry however, products have the depth of a gripping novel's depth and provide a personalized sense of commitment to the user. However, it can also be argued true that casual games are simply easier to understand and play, making it more enjoyable than others. Game to be developed should have a precise balance between being a “hardcore game” and a “casual game”, benefiting from both types.

Virtually all tower defense games are deployed and played online in web browsers, thus, they can reach large group of users but are limited buy the capacities of these users' system and the general constraints of the web based developments in terms of playability and performance. Although such games are on the rise; as professional personal computer owner gamers and users, this product should have the concern of having the best performance as a desktop PC application on every day computers, like good graphics. Although it aims to satisfy PC users, it would be better if this project should also be used to have a web browser game only with little tweaks.

Today's many PC-based games are being converted to be played on game consoles since many players turn to these devices for a period of time. As our game can be implemented on web browser (with some additional tweaks, since it is only a secondary concern), it also should be playable on a popular game console. This problem, however, has higher concern than its playability on the web.

3. Requirements Analysis

3.1. Overview

The system to be developed is a tower defense game. As the name of the games genre implies, the most important components of the game are the towers. Towers are the only tools which player can use to defend its domain. By using different towers provided to him, which change drastically to the character s/he creates, player tries to optimize her/his resources to his tower building and upgrade activities, and by using the space provided to him as efficient as possible. Rapid and changing attacks from enemies force player to think all possible angles, and provide a rich gameplay experience. There are various kinds' towers for all backgrounds. The diversity of the towers not only provides different gameplay, but further character specification also allows different strategies to implement.

"The Last Defense" can be considered as both a turn based, and a real time strategy game. Player is able to think and decide in between turns, but may take simultaneous decisions and do actions while the game runs.

Although players may successfully finish a level, it is impossible to "win" the game, because there is no finishing in this system, as it is in most of the tower defense game. Game continues to create new challenges, one after another, until player decides to finish the game, or loses it. It is aimed to provide fun as much as a player can take.

3.2 Functional Requirements:

- The user should be able to create a new profile.
    - The user should be able to enter a new name for the new profile.
    - The user should be able to select a background for the new profile.
    - The user should be able to select a trait for the new profile.
    - The user should be able to select a portrait for the new profile.
- The user should be able to create a new map.
    - The user should be able to select the terrain for the new map.
    - The user should be able to determine the path that enemies will follow.
    - The user should be able to determine the location of home base.
    - The user should be able to determine the location of enemy base.
- The user should be able to view/select one of the saved profiles.
    - The user should be able to view the name, background and statistic of the current profiles.
    - The user should be able to view her/his achievements in her/his profile.
- The user should be able to view/select one of the saved maps.
    - The user should be able to view the name, background and statistic of the current profiles.
- The user should be able to select the difficulty level of the game.
- The user should be able to add a new tower to the map during the game or between levels.
- The user should be able to upgrade the towers on the map during the game or between levels.
- The user should be able to purchase lives or request new tower building options during the game or between levels.
- The user should be able to exit the game during the game or between levels.
- The user should be able to start the next level when the current level finishes.

3.3 Non-Functional Requirements:

- Usability Requirements:
  - Inexperienced player should be able to start a new game within 60 seconds.
  - Map editor must be easy to use for players.
  - Character creation screen must give enough information to the player about available backgrounds and traits.
  - Game mechanics must be adjusted properly for each background and trait options in order to provide balanced game play.
  - Difficulty of the game should be adjusted carefully to avoid boringly easy or frustrating hard gameplay experience.
- Implementation Requirements:
  - For developers, adding new item (tower or enemy) should be a trivial process and shouldn't involve any coding process. System should be able to recognize added content files and responds properly.
- Portability Requirements:
  - Resolution of the game must be suitable for all common monitor types.
- Performance Requirements:
  - Game should be able to run at a rate of 60 frames/second in order to provide smooth frame transitions.
  - In game graphics must be clear, well designed and nice looking in order to create nicer atmosphere for players.

3.4 Constraints

The implementation language must be C#, using .NET and XNA Framework. These tools allow developer to work more efficiently on game-related products, especially for efficiently using graphics and sound mediums. Also, virtually all personal computer gamers use Microsoft Windows, and about half of the professional console gamers are Microsoft Xbox users. In addition, Microsoft itself highly supports games developed on XNA and .NET Framework, giving more reason to satisfy these requirements.

It should be designed and implemented by following object oriented software engineering techniques, mainly as they were instructed on CS 319 course. Since this game is a class project, it should match the criteria of the course itself.

3.5 Scenarios

1. SCENARIO NAME: USING OPTIONS SCREEN

Participating Actor: Turgay Bayraktar (Player)

Entry Condition: Player has successfully started the program and is in the main menu.

Flow of Events:

Turgay decides to play the game, but he is concerned that his system does not satisfy the hardware requirements of the game for maximum graphical features. Therefore, he decides lower graphical features. Initially, he clicks the option menu found on the main menu screen. After that, Input Manager reaches Game Screen Manager via changeScreen method. Game Screen Manager creates the Option Screen Manager.

Turgay decides to give up modifying the view of screen. The view of screen remains the same. According to Turgay, the shadows in the game are too morbid; and he wants to disable them. He clicks the SHADOW button found on Input Manager. He toggles SHADOW button found on Option Screen. So, Turgay has successfully disabled shadows.

Then, he clicks the EXPLOSION button found on Input Manager. He toggles EXPLOSION button found on Option Screen. Similar to previous action, he just disabled explosions.

Turgay thinks that the balance of sound is very low. He wants to make the sound higher. Turgay clicks the SOUND button. This command reaches Input manager. Input manager reaches GameScreen Manager via changeScreen method after Turgay presses sound browser. He changed the balance of sound and Turgay is glad to increase the volume.

Turgay has mastered tower defense games; so he wants to play the most difficult level. He presses DIFFUCULTY button and input manager reaches game screen manager buy pressing difficulty browser clicked by him. Turgay makes the game harder by pressing “ON” button the level reaches the 12 which is the most difficult in the game. After making changes, he wants to begin playing the game so he clicks the BACK button.

Exit Condition:

Turgay has successfully returned main menu and settings he had changed are applied.

2. SCENARIO NAME: USING MAP EDITOR

Participating Actor: Turgay Bayraktar (Player)

Entry Condition: Player has successfully started the program and is in the main menu.

Flow of Events:

Turgay wants to play the game with his own map. He presses the "MAP EDITOR" button found on main menu screen. His interaction with the interface directs him to the Map Editor Screen.

After reaching Map Editor Screen, he first wants to choose a snowy background. He clicks the terrain browser buttons. His interaction changes the state of the screen so that it shows another terrain with each click.

After finding a snowy terrain, he needs to determine the route of enemy. Initially, he presses the CREATEPATH button. Now, screen indicates that he is building a path. By clicking the empty terrain, he draws a line for enemies to walk on. He interacts with the system by clicking the empty snowy map while in CREATEPATH mode.

He decides to locate the beginning position of the route at the most upper right by clicking SOURCE button and he determines the end position as the upper left screen by clicking TARGET button. As putting paths, Turgay had put a start and an end point for monsters to walk on.

After creating her map, Turgay presses CREATE button found on MAP editor screen. Turgay decides that he should try other another terrain and just changes the terrain as hilly. He clicks "CHANGE BACKGROUND" button, only to see an error message, saying that there already exists a map with that name.

Realizing his mistake, he changes the name of the map as CivilWar and click CREATEMAP for the last time. After creating the map, he returns the Main Menu Screen by pressing BACK button . Map Editor Screen is destroyed.

Exit Condition:

Turgay has successfully returned main menu, with a newly created map kept in by the system.

3. SCENARIO NAME: CREATING PROFILE AND THE PROCESS OF PLAYING GAME

Participating Actor: Turgay Bayraktar (Player)

Entry Condition: Player has successfully started the program and is in the main menu.

Flow of Events:

Turgay wants to play the game, so he clicks START GAME button. Input manager reaches Game Screen Manager by changeScreen method. Game Screen Manager creates the Option Screen Manager. Game Screen Manager creates Start Game Screen Manager object. Start Game Screen Manager creates Game Logic Manager. Game Logic creates Game Play Screen.

Firstly, he clicks CONTINIUE button to select his character. Start Game Manager reaches by StartNewGame operation. Game Logic creates Game Play Screen.

Before beginning to play, Turgay prefers creating new character rather than playing with existent character. Initially, he clicks CREATE button. Then, Input manager reaches Game Screen Manager via characterCreation method. Game Screen manager creates Character Creation Screen. Here, Turgay decides to determine the race of character as Elf. He determines the features of his character as mage on the Start Game Screen Manager. He puts his character's name as Aura. After creating the character of the Turgay, Game Logic creates the profile of player.

Turgay reaches Game Screen Manager. In the beginning of the game, he has 220 gold, 12 lives He begins to select the tower to locate them strategic positions. He can chose a tower whose splash Damage is very low. After selecting his tower, he presses EMPTY TITLE button. He gives the tower a name. Input Manager reaches Game Screen Manager. Game screen Manager goes back Game Logic after initializing new Tower values. After adding tower on Map editor, the tower object is created.

Turgay presses SELL TOWER Button. Input Manager reaches Game Screen Manager. Game screen Manager goes back Game Logic after initializing new Tower values. After removing tower on Map editor, the tower object is destroyed.

The more Turgay wins, the harder levels become. After the sixth level, Turgay decides to upgrade the tower which has stronger with respect to speed hit. Therefore, he clicks TOWER UPGRADE button. Input manager reaches Game Screen Manager. Game Screen Manager gives the demand upgrading the tower to Game Logic.

Turgay loses much lives the value of lives becomes 3.Threfore he decides to buy lives. He clicks BUY LIVE button. Input manager reaches Game Screen Manager. Game Screen Manager gives the demand upgrading the tower to Game Logic.

Exit Condition:

Turgay has successfully returned main menu and settings he had changed are applied.

3.6 Use Case Models

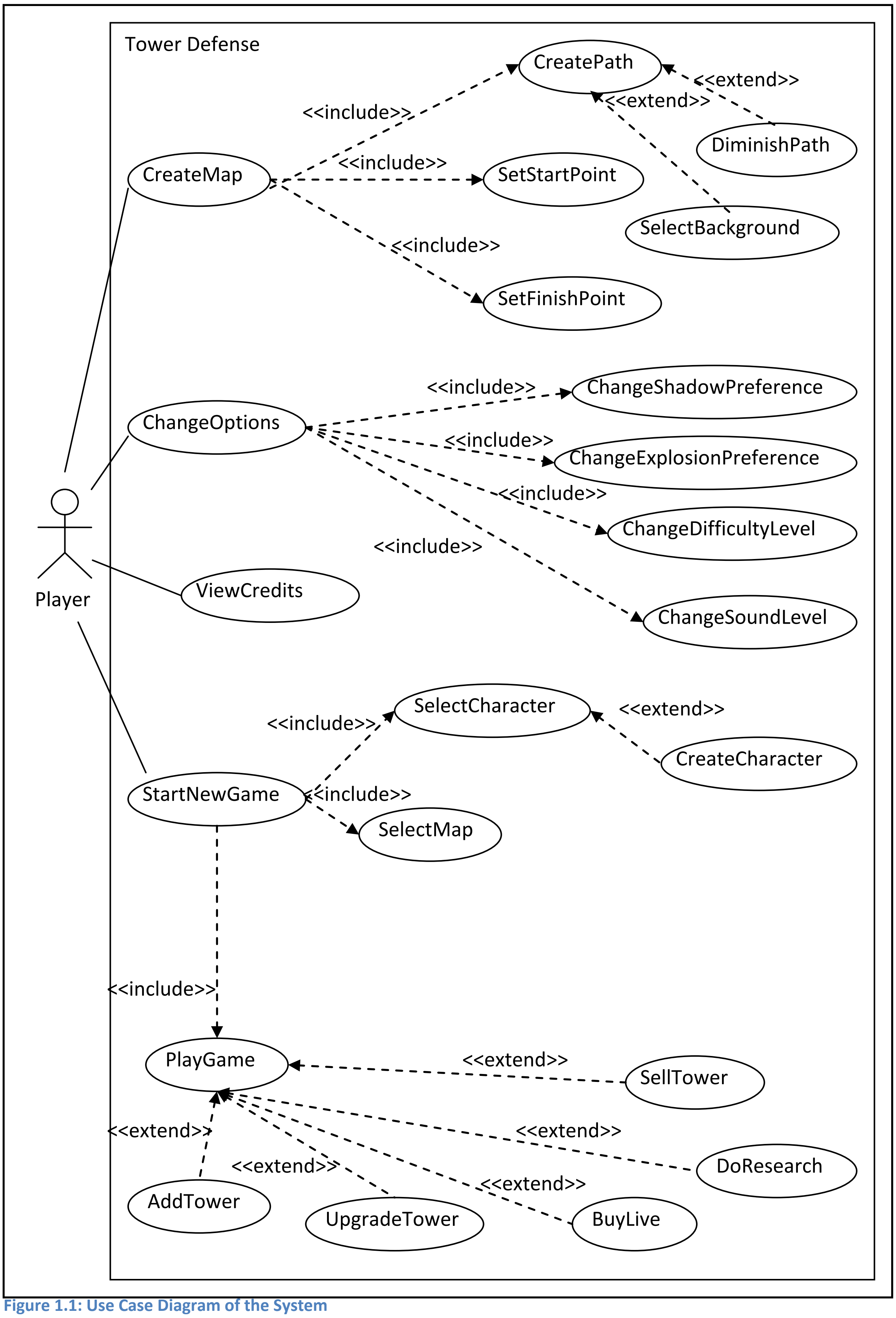


Figure 1.1: Use Case Diagram of the System

Tower Defense game is a well known real-time strategy game and the system should respect and follow the contentions of this genre. Game play should be in accordance with the conventional rules. Player should choose a character before playing the game. If he likes, he should be able to create a new character. Then, he should choose a map and play the game. In game play, traditionally, the player should be able to add towers to the map, upgrade existing towers, sell existing towers, conduct researches and buy lives with his points.

System should allow gamers to create their own maps through a map editor in addition to the maps that are existent in the system. In map editor, the player should be able to select a start point and finish point which are correspond to entrance and exit points of the enemies. Between those two points, the player should be able to create the path with chosen background or delete the parts he does not like.

The game options should be adjustable. The player should be able to disable the shadows or explosions in the game. He should be able to change the sound and game difficulty levels.

The credits –developers- should be presented in the system. The player should be able to view the names of them.

3.7 User Interface

1. Main Menu:

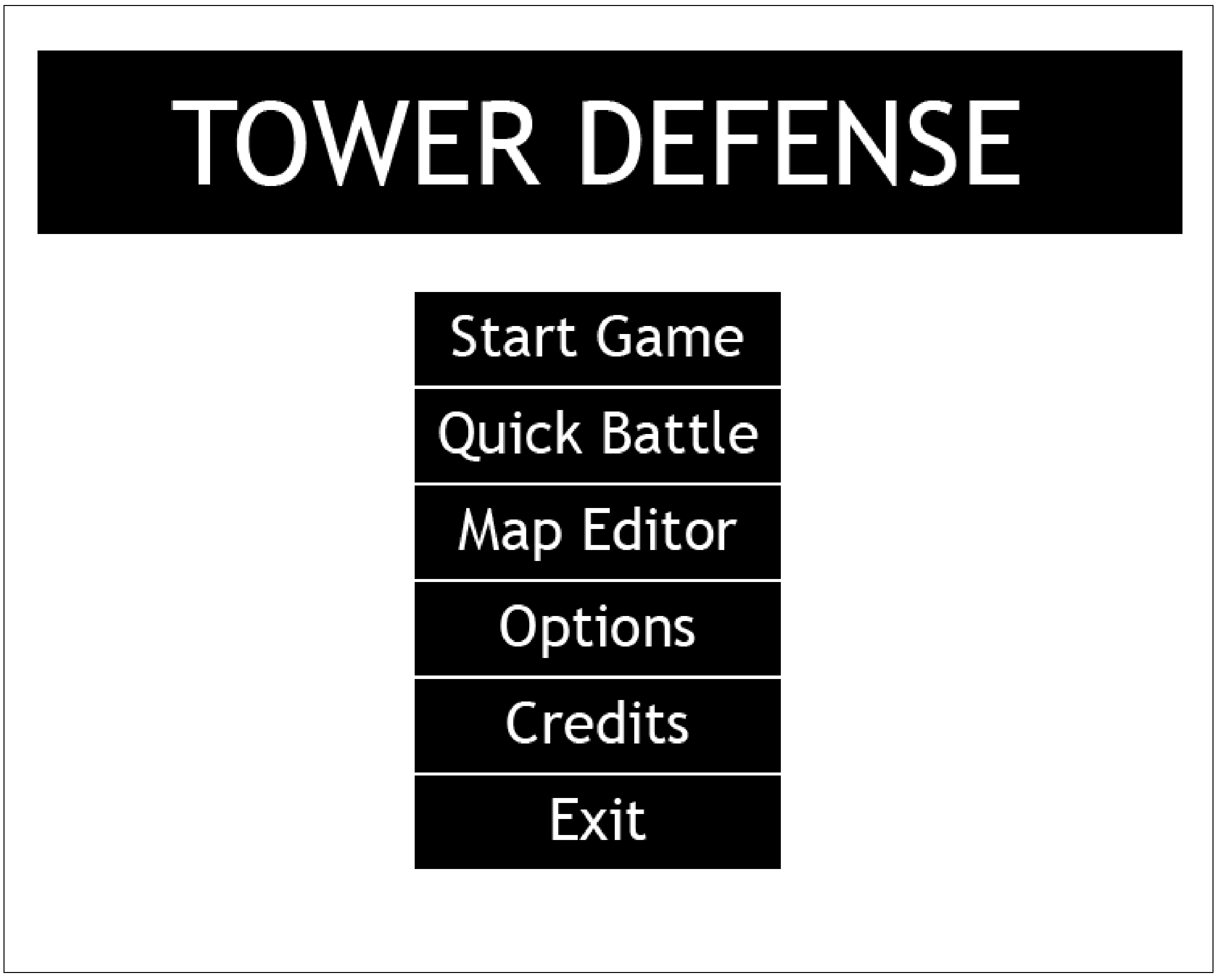


Figure 2.1: User Interface of Main Menu

This is the first screen when application is started. User is provided with options as given above. User can click one of the buttons in Main Menu to proceed. Start Game button directs user to the Start Game Screen. Quick Battle immediately starts the game with randomized settings. Map Editor button directs user to the Map Editor Screen. Options button links to the Options Screen while Credits button directs user to the Credits Screen. Exit button terminates application.

2.Start Game Screen

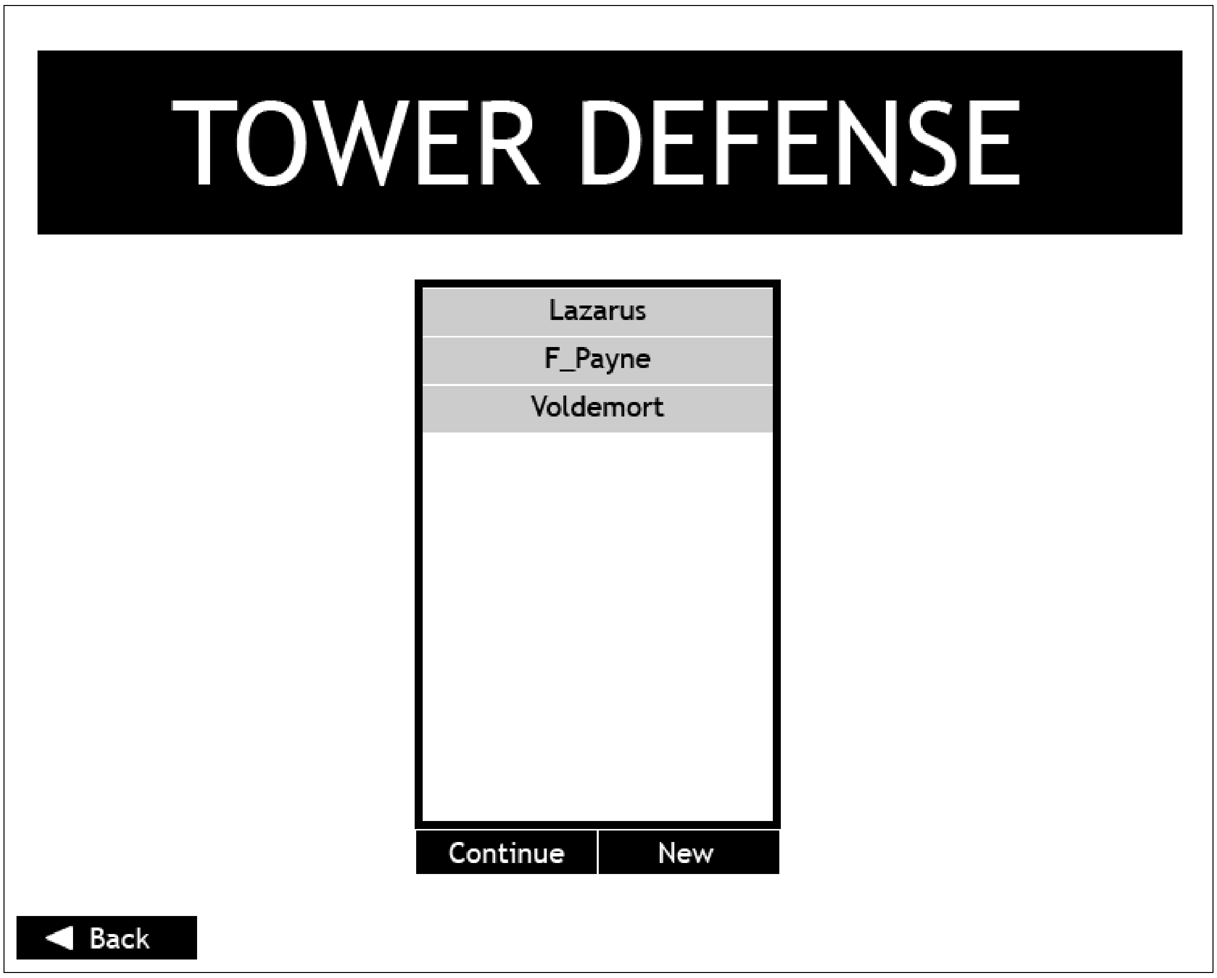


Figure 2.2: User Interface of Start New Game Screen

In this screen user either creates a new character by pressing New button or if there is already created characters, user is free to choose one of them from list to continue. New button directs user to Character Creation Screen, while Continue button opens Profile Screen for selected profile. Back button returns Main Menu.

3. Character Creation Screen

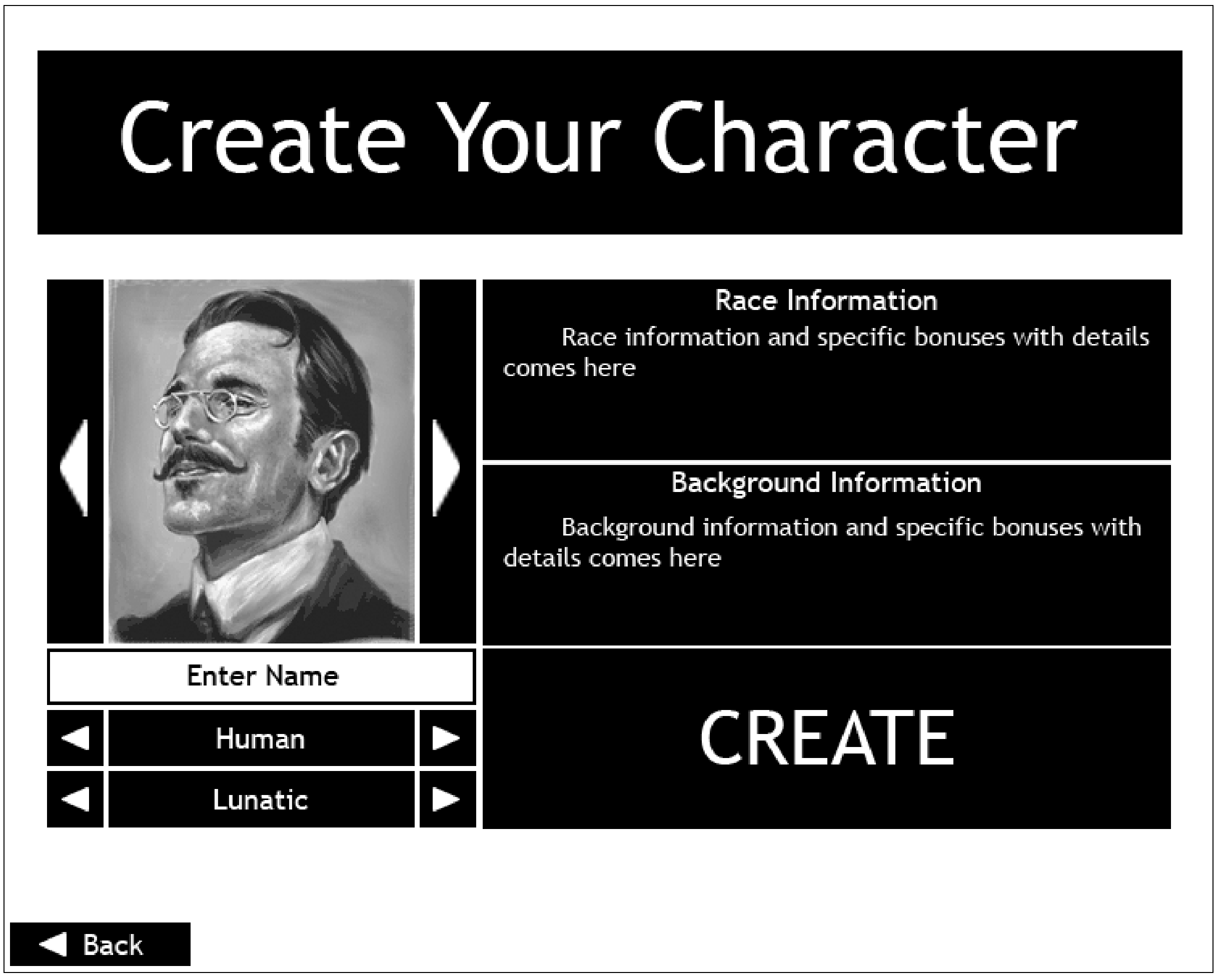
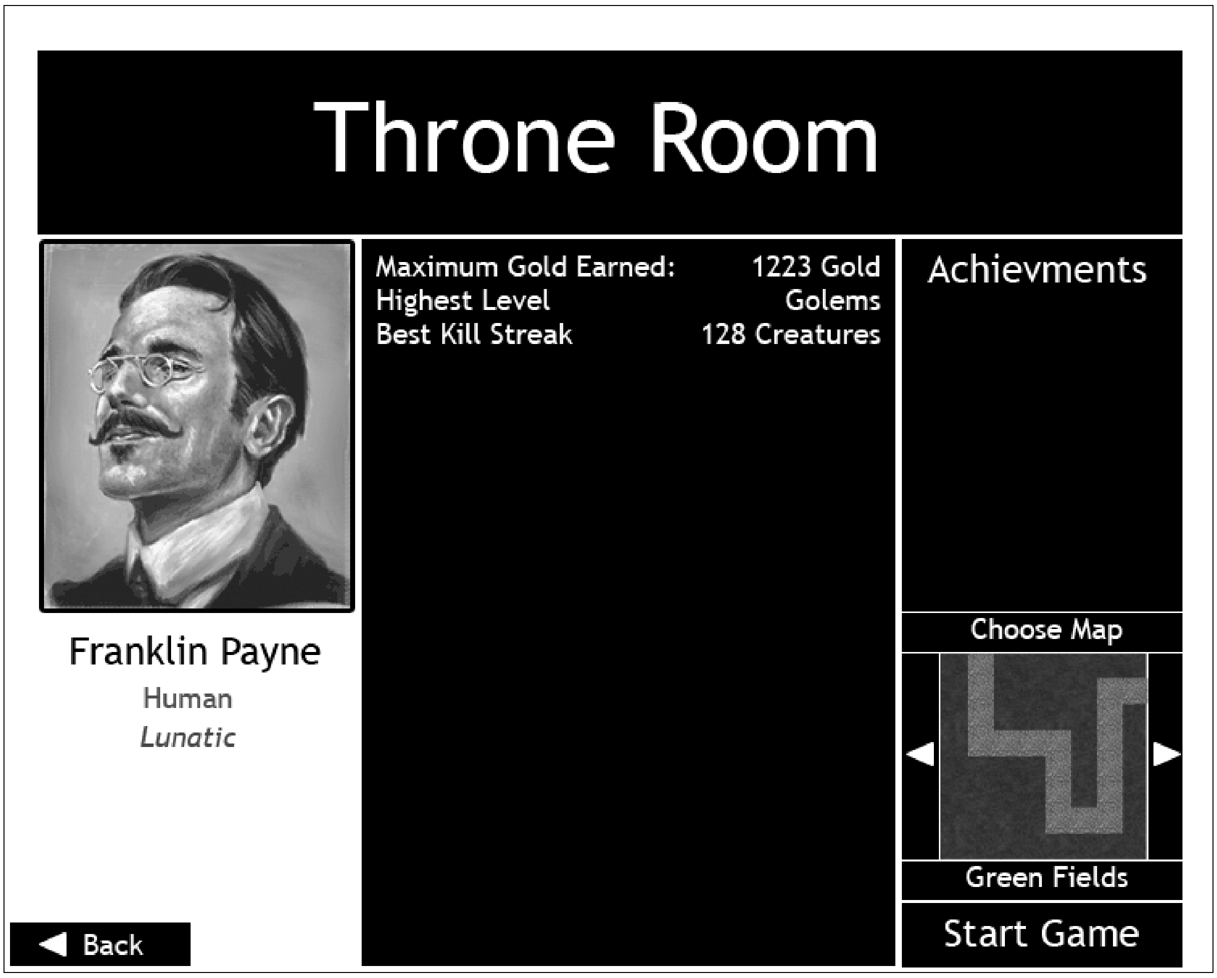

Create Your Character
Race Information
Race information and specific bonuses with details comes here
Background Information
Background information and specific bonuses with details comes here
Enter Name
Human
Lunatic
CREATE
Back

Figure 2.3: Character Creation Screen

In this screen, user creates and customizes his character. He can choose avatar, specify name, background and trait. Traits, backgrounds and avatars are predefined, so user has to choose one of the predefined ones. However, user can enter any name on Enter Name field. Right part of the screen is reserved for detailed information for background and trait selections. Back returns user to the Character Creation Screen. Create button proceeds to the Profile Screen.

4. Profile Screen

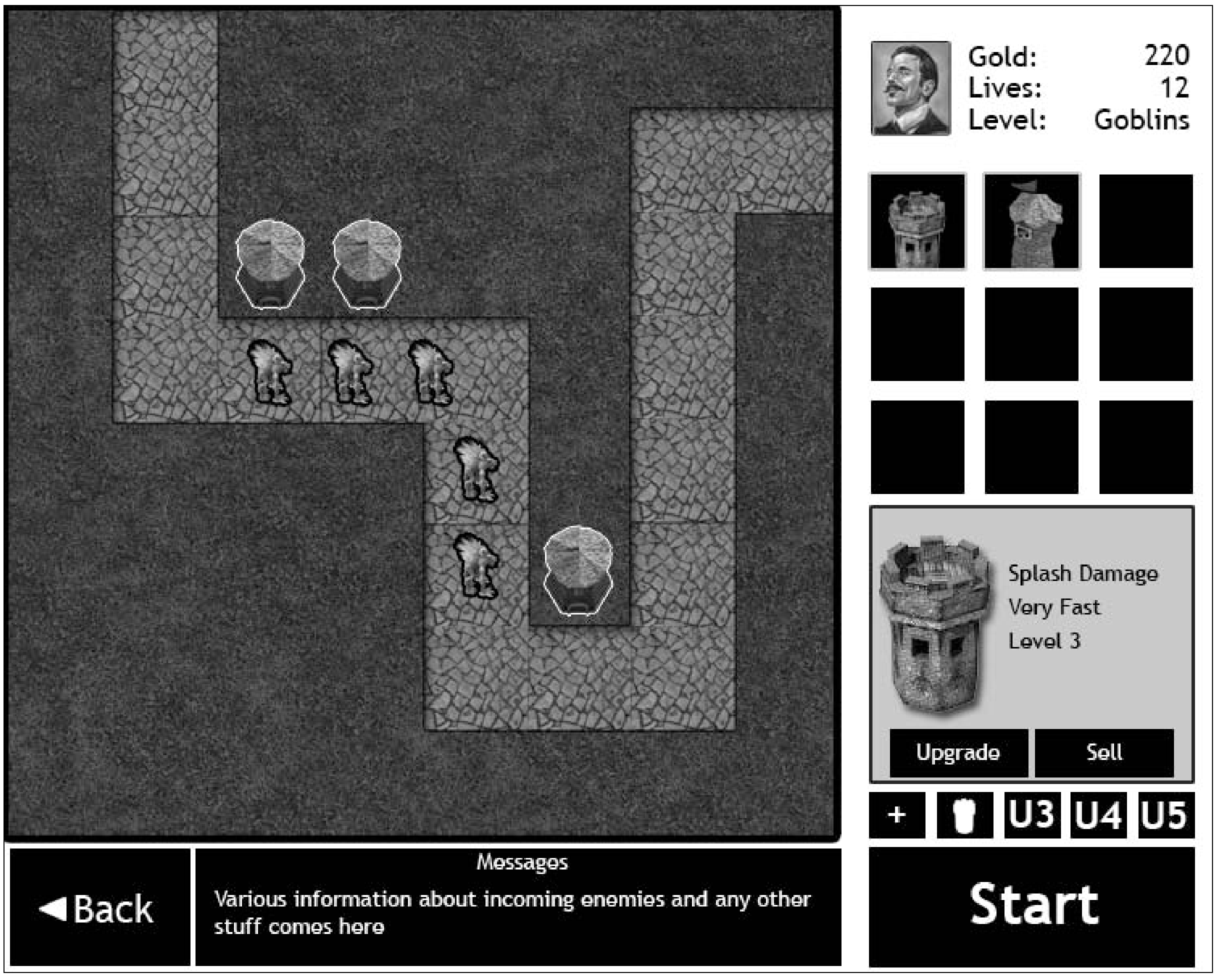


Maximum Gold Earned:	1223 Gold
Highest Level	Golems
Best Kill Streak	128 Creatures

Figure 2.4: Profile Screen

This is a profile screen. User can see his characters name, background trait and of course avatar. Middle part of the screen shows statistical information about player. Right part is allocated for achievements that are unlocked. Also, in this screen, user can specify a map to play. With Start game button, user can start the game. Back button returns user to the Start Game Screen.

5. Gameplay Screen



Various information about incoming enemies and any other stuff comes here

Figure 2.5: Gameplay Screen

In Gameplay Screen, user can see gameplay map, towers and enemies. Right panel is allocated for user actions such as building a tower or upgrading existing ones. Upper-right part gives brief information about player and current level. Below there is a tower list in which user can select and build. While navigating through towers, user can see detailed information about them in Tower Info Panel. Part below is to show incoming messages to the user. Back button returns user to the Profile Screen. Start button starts game. Once game is started, this button can be used to pause game.

6. Map Editor Screen

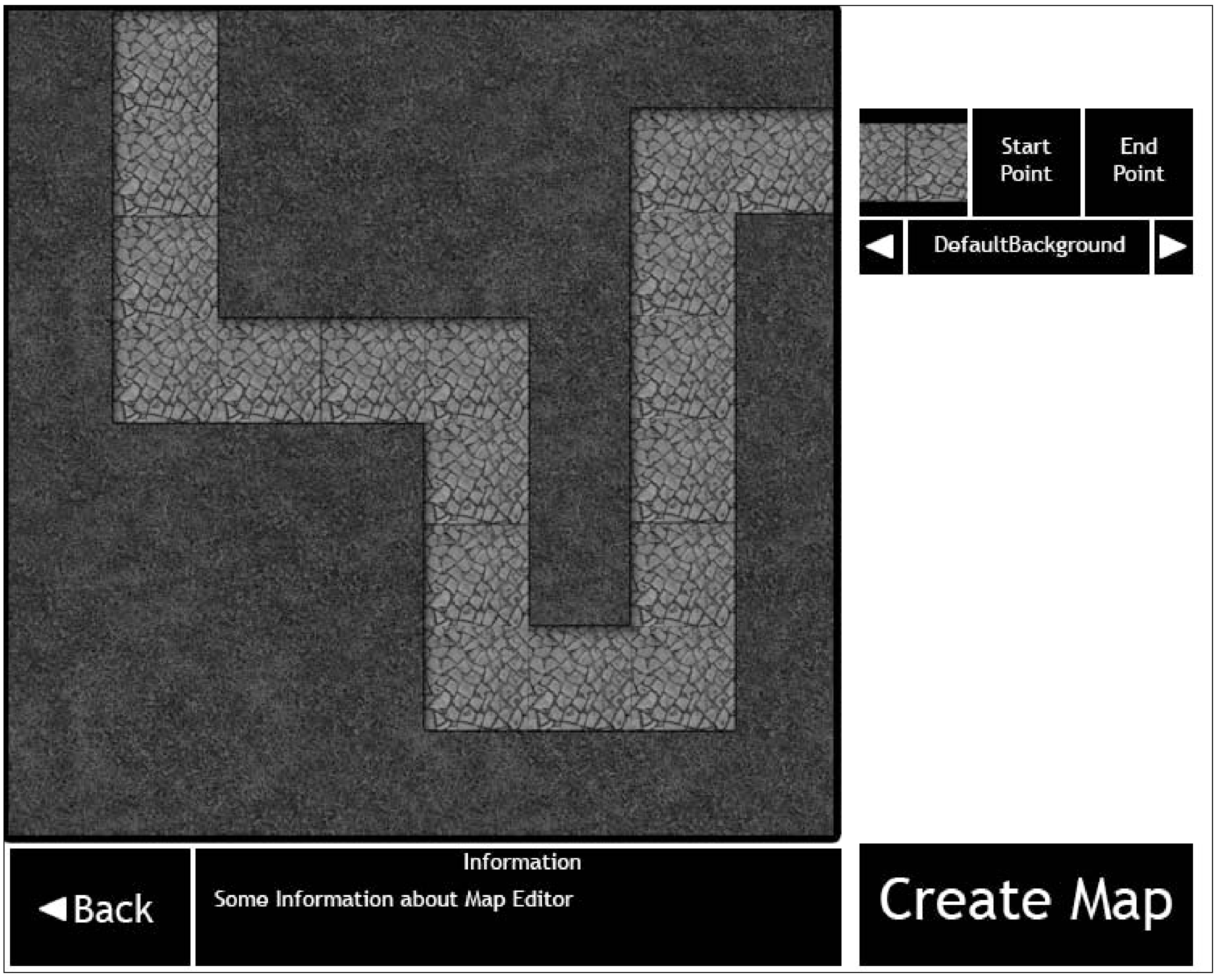
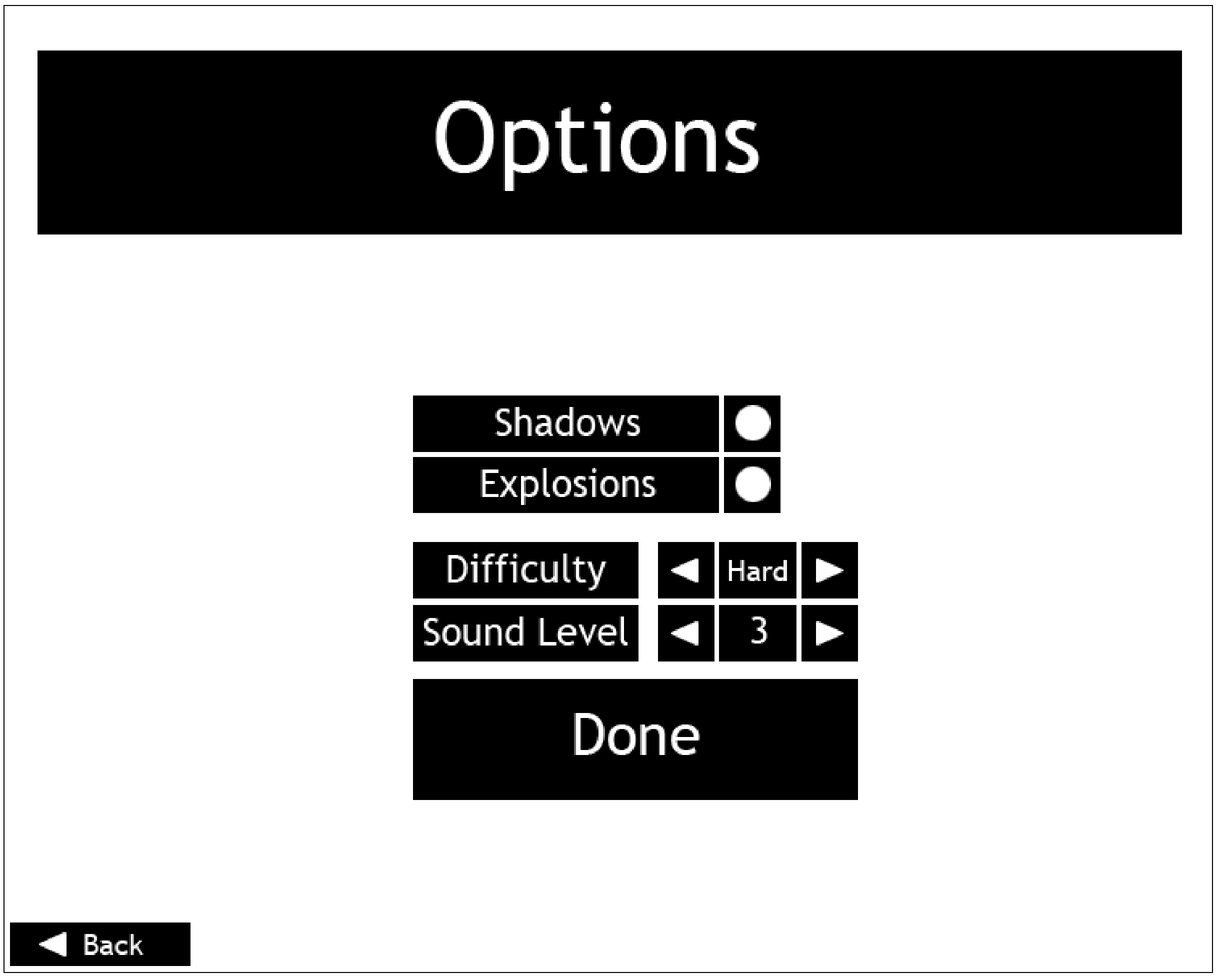


Figure 2.6: Map Editor Screen

In this screen user can create new maps. User must choose a background, then start and end point for path. After that user can connect these points with roads. All these building blocks are available at the right side of the screen. Create Map button creates the map. While Back button takes user to the main menu.

7. Options Screen



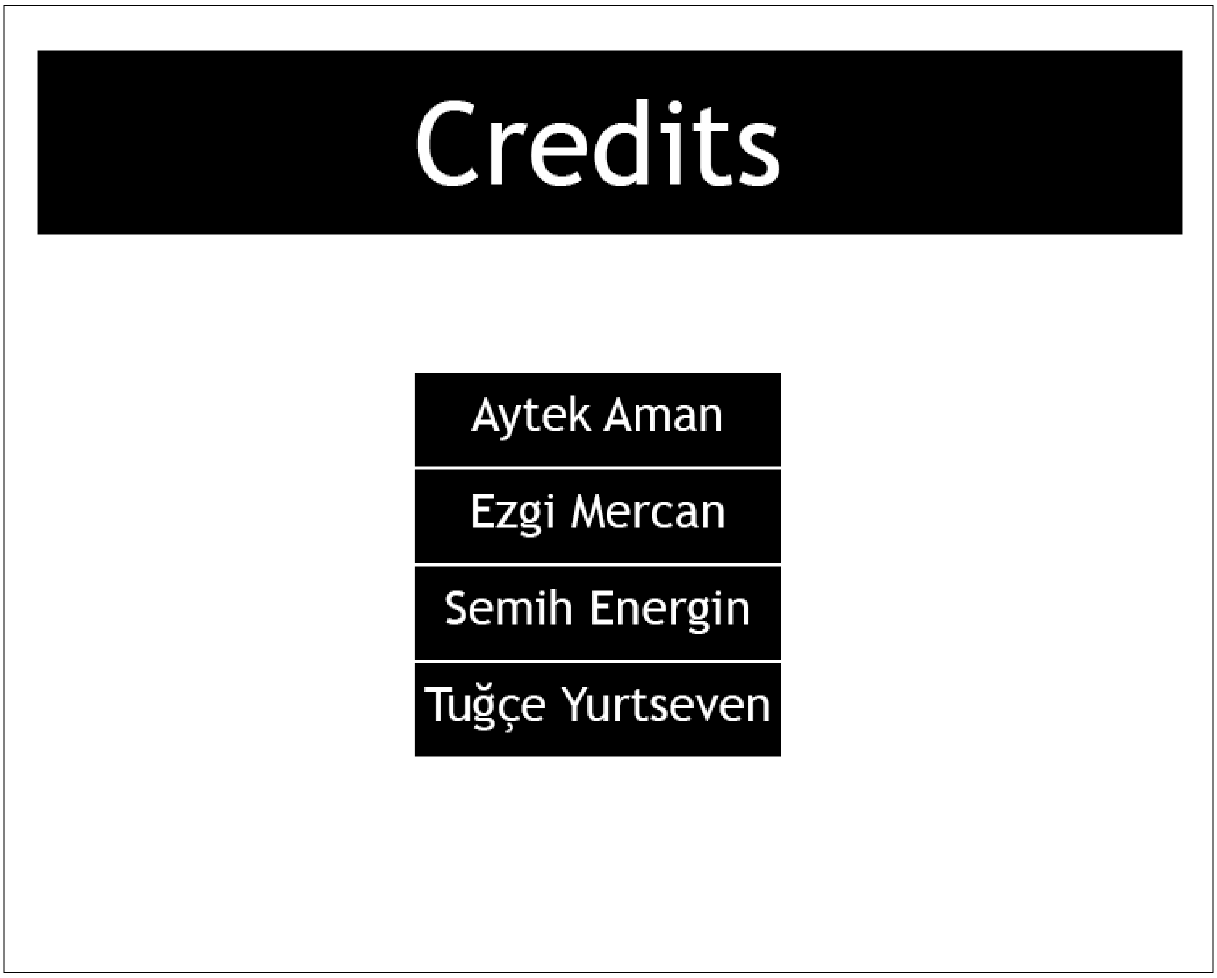
Difficulty: Hard

Sound Level: 3

Figure 2.7: Options Screen

In this screen user can modify settings of the game such as difficulty and sound level. Also, the player should disable some visual effects in the game for better performance.

8. Credits Screen


Credits

Aytek Aman

Ezgi Mercan

Semih Energin

Tuğçe Yurtseven

Figure 2.8: Credits Screen

This screen gives information about developers.

4. Analysis Models

4.1 Object Model

4.1.1 Domain Lexicon

During the designing of a tower defense game, more than one comprehensive domain had to be examined: gaming, game development, computer graphics game engines, user interface, content and logic. These examinations had been done both by doing research on existing systems and techniques, by doing some questionnaire, or from our general knowledge. Most of the resulting terms are familiar to gamers,   but they still should be reported.

A **Game** is the program to be developed, all of the system, is called a game. It is the resulting artifact user should play/uses, and **Player** refers to the targeted user of the game, the live person itself.

Game Engine:

It is the part of the system which does not interact with the user. However, it is the core of the all system. Does all needed calculations, maintenance of graphical, audible and interactive control objects and ultimately changes in other objects. Ideally, it is completely separated from game logic and game content.

Animation: Animations are the rapid display of a sequence of images of two-dimensional or three dimensional artwork or model positions in order to illustrate an effect.

Collision: A major term in game development, collision refers both the notion of “colliding” agents in the game, and the logical aspect of such collisions. Although it is a major concern of game engines, in tower defense games it is often trivial

Draw: Rendering all drawable objects by the game engine, it allows user to see actually all graphical aspects of the game.

Game Time: Elapsed time in the game logic. System time is not included; it refers the time considered to be spent in the game, not the real time.

Grid: In most systems, not only in games, a screen is separated to little sub-screens, to the smallest units which a user can take action on. It is usually a square, a hexagon or an octagon.

Light: A computer graphics term, light refers both the lightning effect on three dimensional objects and the source of light itself. Light allows user to see rendered objects in the three dimensional space.

Real Time: A genre of strategy games. In such games, all players (either artificial intelligence or human) play the game at the same time, decide and take action simultaneously.

Rotation: A computer graphics term, rotation specifically refers to the action of rotating an image according to a specified origin, to some extent. It is commonly used to illustrate stationary movements in two-dimensional games.

Scaling: A computer graphics term, scaling refers re-sizing an image on the screen.

Shadow: A computer graphics term, a shadow is partly illuminated or un-illuminated areas in the three dimensional space. The renderer dynamically creates them.

Sprite: A computer graphics term, sprites are two-dimensional or three-dimensional images or animations that are integrated into a larger screen. Usually, they consist of a number of images or models which are displayed one after another and create an animated visualization.

System Time: Elapsed time in system creates the real time with game time and other environmental variables. System time increases as the calculations of the game increase and force the hardware.

Texture: May also considered as a game content term, textures are two dimensional image files that cover certain areas, such as a background or a model.

Update: A game development term, update refers to applying all changes in the game logic to the game objects, usually dictated by the state of a game.

X Width: A graphical term, defines the X coordinate of a drawable object in the space.

Y Length: A graphical term, defines the Y coordinate of a drawable object in the space.

Z Depth: A graphical term, defines the Z coordinate of a drawable object in the space.

Effect: Effects are all kind of experience enrichment content, either graphical or audible.

Game Control:

This part of the system deals with user interaction and its effects on both game engine and game logic

Building Menu: Visible and reachable on game-play and preparation screen on a different panel; building menu allows player to browse towers (s) he can build and choose a tower to build.

Click Area: A user interface term, click areas are defined to specific buttons and correspond to a certain area, which allows system to decide when a button is clicked.

Game-play Screen: A state of a tower defense game where game runs and enemies attack to the player's character. The most dynamic of a game, this part can be sums up the conventional meaning of playing with the "Preparation Screen".

Main Menu: The first user interactive state of a game. Unlike other computer programs, nearly all computer games start with a main menu and allows user to navigate through states by using large buttons.

Map Editor: A sub-system in the game, level editors allows an interface to users add contents (specifically, a map) to an existing game.

Options: Reached either from main menu or after starting a play, this menu allows user to change settings of the game. It usually allows user to change graphics settings, sound settings, control settings and game-play settings.

Preparation Screen: A state of a tower defense game which is just before game-play settings. In this state, enemies do not attack, and there is no movement. User prepares his actions, and usually can do what s/he can do during game-play screen.

Tower Info Menu: Visible and reachable on game-play and preparation screen, tower info menu is visible only when a built tower is selected. It gives information about towers status, values, provides a button to sell it or upgrade it.

Game Logic:

With game engine, game content, and game control creates the sum of all the game. Game logic is the model of the game, and defines what game should do, and how it should interact with the user. Do calculations and changes that are related with the rules of the game, however is not interested how in the end they will be presented or implemented in the system. Ideally, game logic is completely separated from game content and game engine.

Agent: An agent refers to an object in the game that is capable to take an action, by examining other agents and the environment.

Attack: A common term in gaming, attack refers any type of action which aims to do any kind of damage or hindrance. In tower defense games, attacking refers the whole action of aiming a creature in range, shooting a bullet to it, hit or miss action of a bullet, applying damage and damage type to creatures related.

Attack Range: Distance, which a tower can aim and shoot to, starts from its center.

Attack Speed: Attack speed is a value of an attacking agent, which decides the minimum time between two following attacks

Building: The most common and needed player action in tower defense games meaning creating fortifications (usually towers) is that preventing enemy.

Bullet: Although word of bullet mean a single bullet in general gaming usage, tower defense games refers bullets as agents that are sent by the towers to the enemy creature.

Bullet Speed: Bullet speed is a value of an attacking agent, and the bullet agent, which decides how much space a sent bullet will parse in a unit time.

Continuous Damage: A damage type which is not common in tower defense games, “continuous damage” type attacks continue to decrease applied creatures' hit point even after it is applied to the target. Fire damages are an example to that.

Creature: Also referred as “monster” or “soldier”, creatures are the tools of enemy to win the game in tower defense games. They move through the path, and wish to survive attacks from towers. They have no capacity to attack or to show any sentient resistance.

Creature Speed: Creature speed is a value of an enemy creature agent, which decides how fast it can move through the map.

Damage Type: Damage types are values that are stored in attacking towers and their bullets, to apply a certain type of damage. Damage types change the applied creatures, any possible hindrance to these creatures, damage that applies to these creatures.

Enemy: Opponent of the player. In gaming, it corresponds to all possible opponents and competitions, however tower defense games are single player, with very strict rules on player's and opponents role. Thus, enemy is the opponent of the player, which aims to finish the lives of the player.

Enemy Base: A tower defense game term, referring to the starting point of enemy creatures.

Environment: A term related to game logic and artificial intelligence, environment is the sum of all objects that are not capable to take an action and is in game logic itself.

Game Artificial Intelligence: A sub discipline of artificial intelligence, game AI is very different from academic discipline of artificial intelligence. Even to some, it is wrong to state that artificial intelligence is used in games. Artificial intelligence in games aims to achieve an enjoyable and competitive agent behavior in the game.

Game Level: In tower defense games, a game level starts when player finishes preparation screen and ends when game-play screen finishes either with success, or failures. A resulting success allows user to pass a level and start a new one. Usually, all game levels consist of a certain type of enemy.

Ground: Remaining parts of the map, excluding paths, home base and enemy base. Consists the majority of the map, and user is allowed to build on such areas.

Gold: Also referred as “coin” or “money”, it stands for the resource itself that is used in the game. It comes from the notion of playing a game in the old ages.

Hit: A common term in gaming, a hit means a successful attack. In tower defense games, it is one of possible ending states of a sent bullet; a hit means a bullet has successfully met with its target.

Hit Damage: A damage type which is common in tower defense games, “hit damage” type attacks only affect the targeted creature and decrease its hit point.

Hit Point: A common term in gaming, hit point is a value that decides how much damage an agent can take before being eliminated. In tower defense games, only enemy creatures have hit points.

Home Base: A tower defense game term, referring to the target point of enemy creatures, which is the “home” of the player.

Importer: In game systems, objects that are related to the game logic are subject to change, either during development or after release. It is also considered as a good practice of game development to separate game entities from the system as much as possible, allowing variations and rapid modification. An importer deals with such external files that keep needed parameters for game objects. They usually read a file, and allow game logic to create related objects accordingly.

Interest: Some tower defense games aim to reward users by not spending many resources and try to motivate others to challenge them to optimize resources. By having an interest rate, the gold player stores in her/his treasury will be on his interest.

Kill: A common term in gaming, killing means eliminating a creature of opponent, by decreasing its hit point to zero or below.

"Letter of Requisition": A term usually used interchangeably used with upgrade or research point in tower defense games. A letter of requisition is a point which plays gains after passing a number of levels/turns. S/he can use these points to

Life: Interchangeably used to define the notion of "life", or the number of "lives" a player has. A game starts with a number of lives, and ends when player lost all her/his lives. Extra life can be gained by spending gold or by having specific traits.

Manhattan Distance: A term related to game logic and artificial intelligence, Manhattan distance is the shortest path between two points by following only moving on one coordinate at a time.

Map: Sum of all grids, where all game runs and opponents compete with each other. A map consists of home base, enemy base and path. Towers and creatures are placed (and move) on the map; however is not part of the map.

Miss: A common term in gaming, a hit means an unsuccessful attack. In tower defense games, it is one of possible ending states of a sent bullet. A miss means a bullet has not successfully met with its target.

Path finding: Algorithms and methods to define a path for game agents. Part of game artificial intelligence, path finding is a major concern in game development. In tower defense games, however, it is a trivial task.

Path: A static way for enemy creatures to move through screen, from enemy base to home base, defined by waypoints. User cannot build towers on paths.

Profile: Sometimes referred as "account", a profile is recordings of a specific player. A player may have more than one profile, and can separate distinct plays from one another.

Slow Damage: A damage type which is common in tower defense games, "slow damage" type attacks decrease the speed of applied creatures. Cold damages are a common example to that.

Splash Damage: A damage type which is common in tower defense games, "splash damage" type attacks not only affects to aimed target, but other creatures in the range as well.

Splash Range: A term related to splash damage, splash range value is the distance that splashed bullet damage can reach at most.

Tower: Basic tool that player has, towers are the foundations of a tower defense game. They have a price (usually in terms of gold or coin), needs a clear area to build on, has a range, an attack type, attack speed and damage value. They can be upgraded to increase one or more of these attributes**.**

Tower Level: State of a built tower, starting from zero to either infinity or a magic number. As it increases, a tower becomes more effective. Tower levels are increased by upgrading.

Trait: A term related to character creation, traits allow further characterization of a created character. They usually give further advantages to player by affecting the game logic. They usually exist in role-playing games, with a few examples in strategy games.

Treasury: Also referred as "bank", it stands for the resource itself that is used in the game. It comes from the notion of having gold as the resource.

Turn Based: A genre of strategy games. In such games, players (either artificial intelligence or human) play the game by following turns, one after another. Actions and decisions do not overlap.

Upgrade: A player action, which allows him to make more benefit of a tower by spending some resource (gold or coin). An upgraded tower has an increased level, and usually further upgrades are more expansive. By upgrading, tower may simply have increased attack range, speed or damage; or may evolve into something else completely.

Waypoint: Waypoints define a set of points which defines a path for the enemy creatures. It starts from enemy base, and is aimed to home base.

Sell: A common gaming term, sell refers to the action of sacrificing a possessed agent of the player to gain some resource on it. In tower defense games, selling refers removing a built tower from a map and gaining some degree of gold from this action.

Game Content:

Part of a game system, which includes all non-implementation and non-logical (in the end, that is not related to the source code directly) features. Textures, sprites, animations, sounds, music themselves (not their implementation) are the most known examples of game content. In addition, aspects that are related with the game logic, but not the playability, are also fall into this category.

Achievement: A relatively young term in gaming, achievements are certain milestones which player gains by satisfying certain conditions. Earned achievements may or may not affect game logic. Usually, they are stored in user's profile.

Avatar: In gaming literature, an avatar is the physical representation of a character, or the player itself. It may be sprite, a three dimensional model, or a portrait. Virtually none of the tower defense games provides an avatar to the player.

Background: A term related to character creation, background allows further characterization of a created character. They usually give advantages and balancing disadvantages to player by affecting the game logic. Sometimes, backgrounds change game content too, and increases re-playability and realistic aspects of the game. They usually exist in role-playing games, with a few examples in strategy games.

Character: A term related to the story and content of the game, a character is the "thing" which player should simulate/play. It is predefined by the game, selected from a list, created by the player to an extent, or completely defined by the player. Having a character to play is usually non-existent for tower defense games.

"Kill Streak": Number of rapid kills that a player has reached without any miss, one after another.

Terrain: Terrain is the background image for the map, which is displayed on ground grids. It is read from external content files and displayed by the game engine.

4.1.2 Class Diagrams

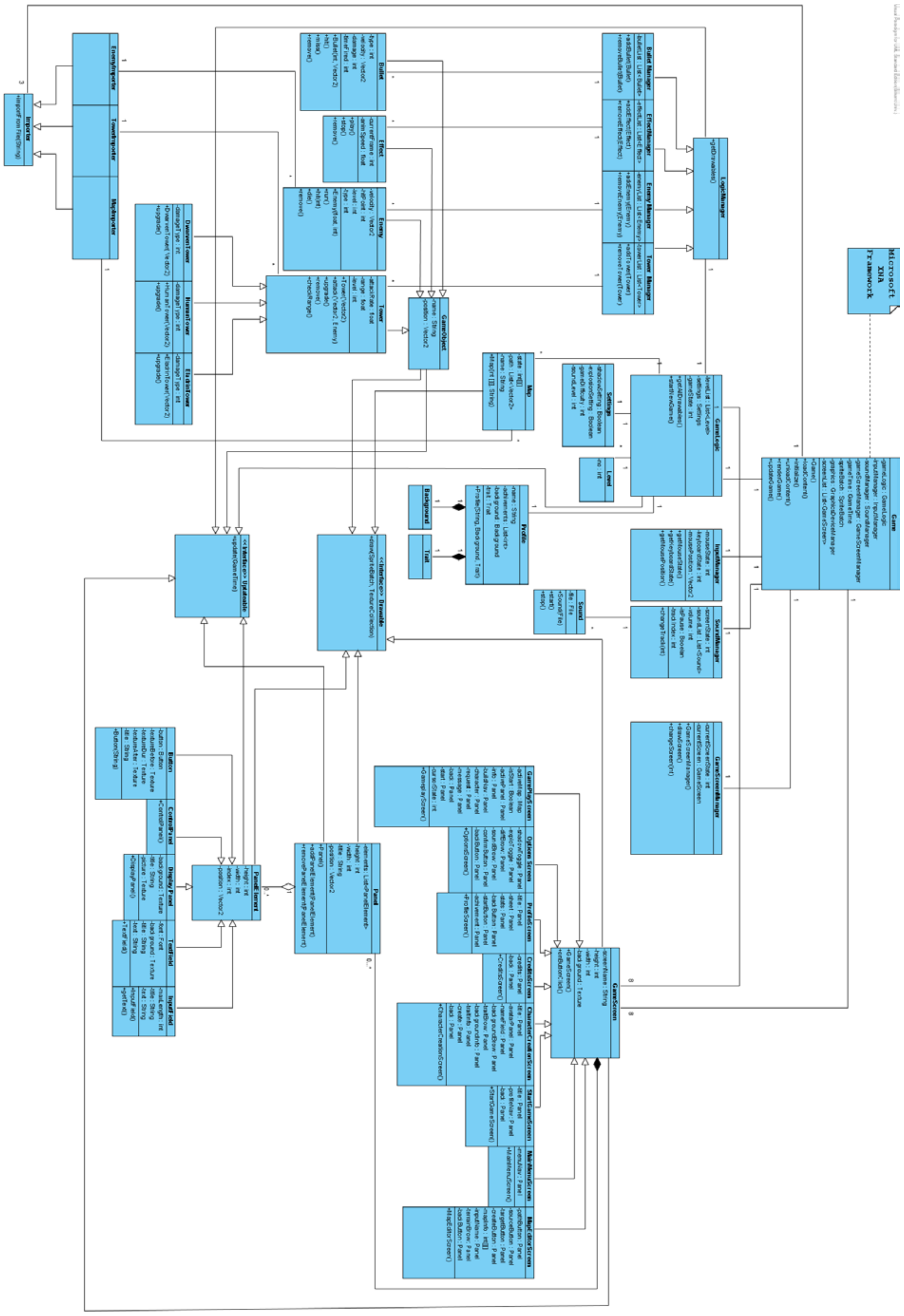


Figure 3: Class Diagram of the System

InputManager: This class handles user inputs like mouse and keyboard actions. Internally, it stores input states (mouse position, pressed keys etc.)

-getMouseState(): This method returns current state of the mouse. It can be idle, pressed or released.

getMousePosition(): Returns current position of mouse as Vector2D.

getKeyboardState(): These method returns current keyboard state (keys that is pressed etc.)

Sound Manager: This class plays sounds/musics when needed. It contains Sound list, which is a collection of all sounds/musics in the game.

changeTrack(): This method is used to change current music.

Sound: This class can be considered as a one sound effect or music.

start(): Starts to play this sound.

stop(): Stops to play.

GameScreenManager: These manager class stores all screens and handles transitions between them. It also calls draw() method for active screen.

drawScreen(): draws currently active screen.

changeScreen(int): changes currently active screen with another one.

GameScreen: is an abstract screen that represents generic game screen. It has trivial features like name, size and background texture.

onButtonClick(): This method is a listener method. It is invoked when button on the screen is pressed. Implementation must be provided by subclasses.

GamePlayScreen: This class represents game screen in where actual game is played. It draws all drawable game objects in GameLogic that must be drawn via draw method. (draw() method is inherited from Drawable Interface) It also contains buttons (Start game, back etc), and control panels where user can specify his actions. Since it is a subclass of GameScreen, it must override onButtonClick() method.

Options Screen: This class represents game screen in where game options are specified. Therefore, it has panels to show and change game the game settings. It must override onButtonClick() method.

ProfileScreen: This class represents Profile Screen in the game, similarly it has one or more Panels.

CreditsScreen: This class represents Credits Screen in the game which one or more Panels.

CharacterCreationScreen: This class represents Character Creation Screen in the game. It has one or more Panels.

StartGameScreen: This class represents Start Game Screen in the game which has Panels.

MainMenuScreen: This class represents Main Menu in the game which has Panels.

MapEditor: This class represents Start Game Screen in the game which has Panels and also has a game map to display. Map is drawed in draw() method inherited by GameScreenManager abstract class.

Panel: This class serves as a container for user interface elements that are drawable. It has PanelElement list in where all sub PanelElements are stored. When Panel is drawn, all components in it will be drawn.

addPanelElement(PanelElement): Adds PanelElement to this component. To draw any PanelElement, first it must be added to a Panel via this method.

removePanelElement(PanelElement): Removes particular element from list.

PanelElement: is an abstract class for any panel element like button or textfield. It has trivial attributes like height, width,position etc.

Button: represents clickable button. Generates event when clicked.

setTitle(String): specifies text that is to be displayed on the button.

DisplayPanel: This panel element is used to display text and pictures preferably.

TextField: This class represents plain text to be displayed.

InputField: This element is an input field whose context can be changed by user.

getText: returns current string that is in input field.

Importer: is an abstract class to import game content from file to the game logic classes: Enemy, Map and Tower. Therefore, each subclass has its own implementation for importing.

importFromFile(String): This is an abstract method to import content from file. It must be overridden.

EnemyImporter: Class to import Enemy object statistics from file.

ToweImporter: Class to import Tower object statistics from file.

MapImporter: Class to import Map object statistics from file.

BulletManager: is the controller class for the Bullet objects. It holds the all bullets in a list. All other major controllers access Bullet objects through the instance of BulletManager.

addBullet(Bullet): is the method called for adding new bullets to bulletList.

removeBullet(Bullet): is the method called for removing an existing bullet in the bulletList.

EffectManager: is the controller class for the Effect objects. It holds the all effects in a list. All other major controllers access Effect objects through the instance of EffectManager.

addEffect (Effect): is the method called for adding new effects to effectList.

removeEffect (Effect): is the method called for removing an existing effect in the effectList.

EnemyManager: is the controller class for the Enemy objects. It holds the all enemies in a list. All other major controllers access Enemy objects through the instance of EnemyManager.

addEnemy (Enemy): is the method called for adding new effects to enemyList.

removeEnemy (Enemy): is the method called for removing an existing effect in the enemyList.

TowerManager: is the controller class for the Tower objects. It holds the all towers in a list. All other major controllers access Tower objects through the instance of TowerManager.

addTower (Tower): is the method called for adding new towers to towerList.

*remove*Tower *(Tower)*: is the method called for removing an existing tower in the towerList.

Bullet: is the object abstracting the bullet that is fired from a tower object. It has *type, velocity, damage* and *timeFired* attributes. It also inherits *name* and *position* attributes from *GameObject* abstract class. Its constructor takes type and position as argument, the other attributes are determined according to type, difficulty and level.

hit(): is called when a bullet's position is at the perimeter of an enemy.

misss(): is called when a bullet's position is at the borders of the map.

remove(): is called whenever a bullet hits or misses. It calls the *removeBullet()* in the manager.

Effect: is the object abstracting an explosion effect. It has *currentFrame* and *animSpeed* attributes, which determine the point the explosion effect is on inside the effect timeline and the time before it passes to the next frame. It inherits *name* and *position* from *GameObject* class.

play(): is called when en enemy is hit and increases the *currentFrame* attribute with the speed determined by *animSpeed* attribute. It plays the explosion effect animation.

stop(): is called when the animation is terminated.

remove(): is called whenever an effect is terminated. It calls the *removeEffect()* in the manager.

Enemy: is the object abstracting the enemy in the game. It has *direction, hitPoint, level* and *type* attributes plus the *name* and *position* attributes inherited from *GameObject* class.

run(): is repeatedly called when an enemy is created. It changes the position of the enemy on the path towards the exit point.

hit(): is called when a bullet's position is at the perimeter of the enemy. It reduces *hitPoint* of the Enemy by the *damage* attribute of the bullet.

die(): is called when the *hitPoint* of an enemy is zero. The enemy is destroyed and calls *remove* operation. It also calls the associated effect.

remove(): is called to remove the enemy from the manager's list when an enemy dies.

Tower: is the object abstracting a tower object in the game. It has *range, level, attackRate* attributes plus *name* and *position* attributes inherited from *GameObject.*

attack(): is called whenever checkRange returns an enemy in the *range*. Tower creates bullets and fires them to the enemy.

4.2 Dynamic Models

4.2.1 State Charts

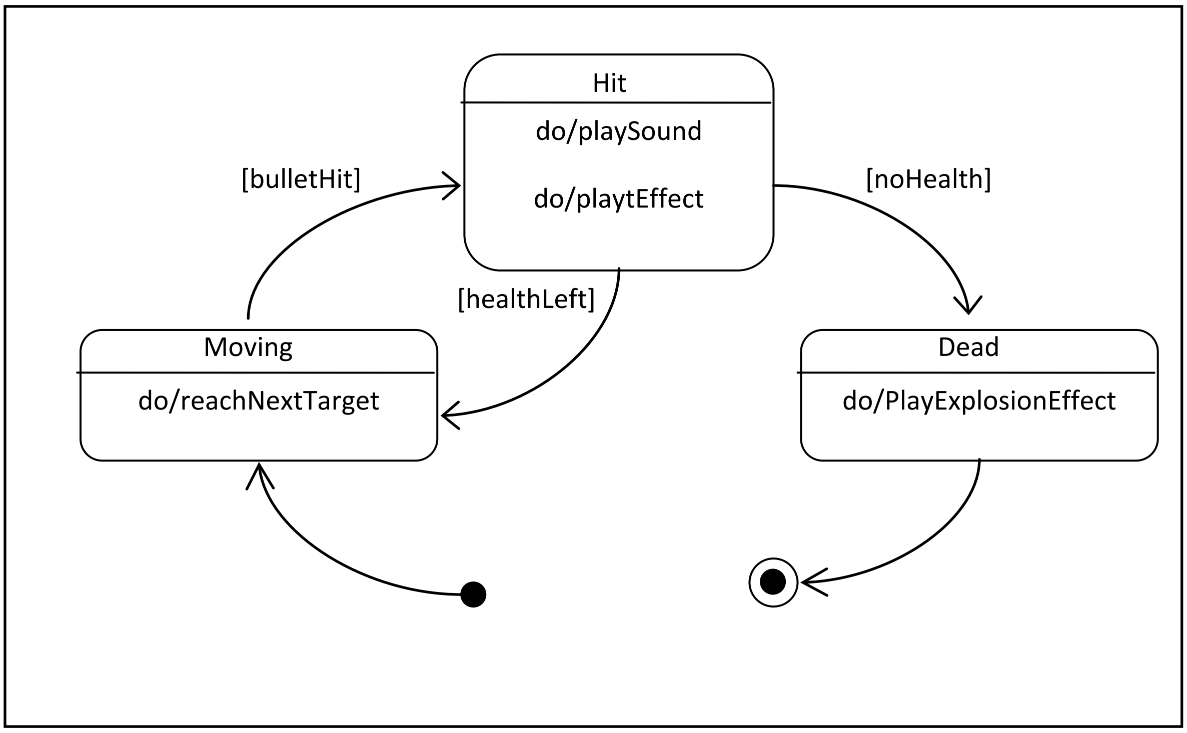


Figure 4.1: State Chart Diagram for Enemy Object

In figure 4.1, states of Enemy object are diagrammed. There are three states of Enemy object. When Enemy is in Moving state, it updates its position by trying to reaching next target, which is the next turning point in the path. If in Moving state, a bullet hits the Enemy, Enemy passes to Hit state, plays a sound and visual effect and checks if there is any health points left, if any left it returns to Moving state; if not it proceeds to Dead state. In Dead state, it plays an explosion effect and finishes the state machine.

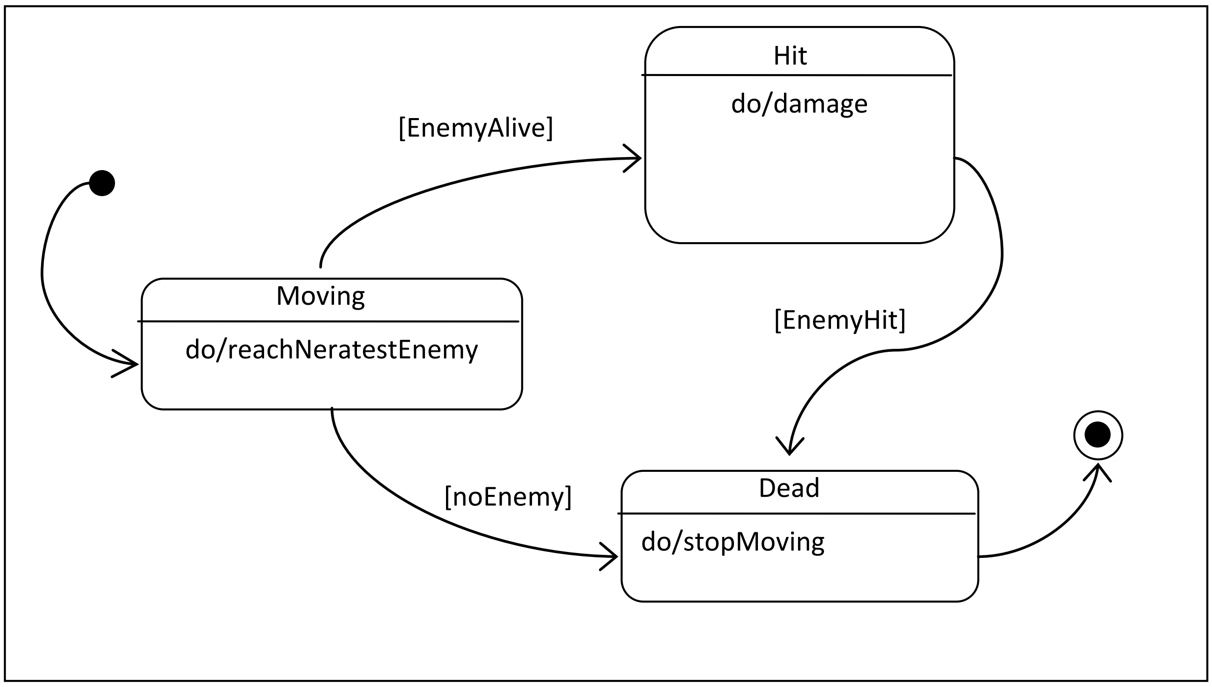


Figure 4.2: State Chart Diagram of Bullet Object

In figure 4.2, the states of Bullet object are diagrammed. Bullet starts in Moving state, it tries to reach the nearest enemy. When the enemy has health when Bullet reaches the position when enemy is in when the Bullet is fired; there are two conditions possible. If the Enemy is still alive, Bullet passes to Hit state and damages the Enemy, then proceeds to Dead state. If Enemy is not alive –killed by another Bullet-, Bullet jumps to Dead state. Dead state is the final state and Bullet finishes the state machine.

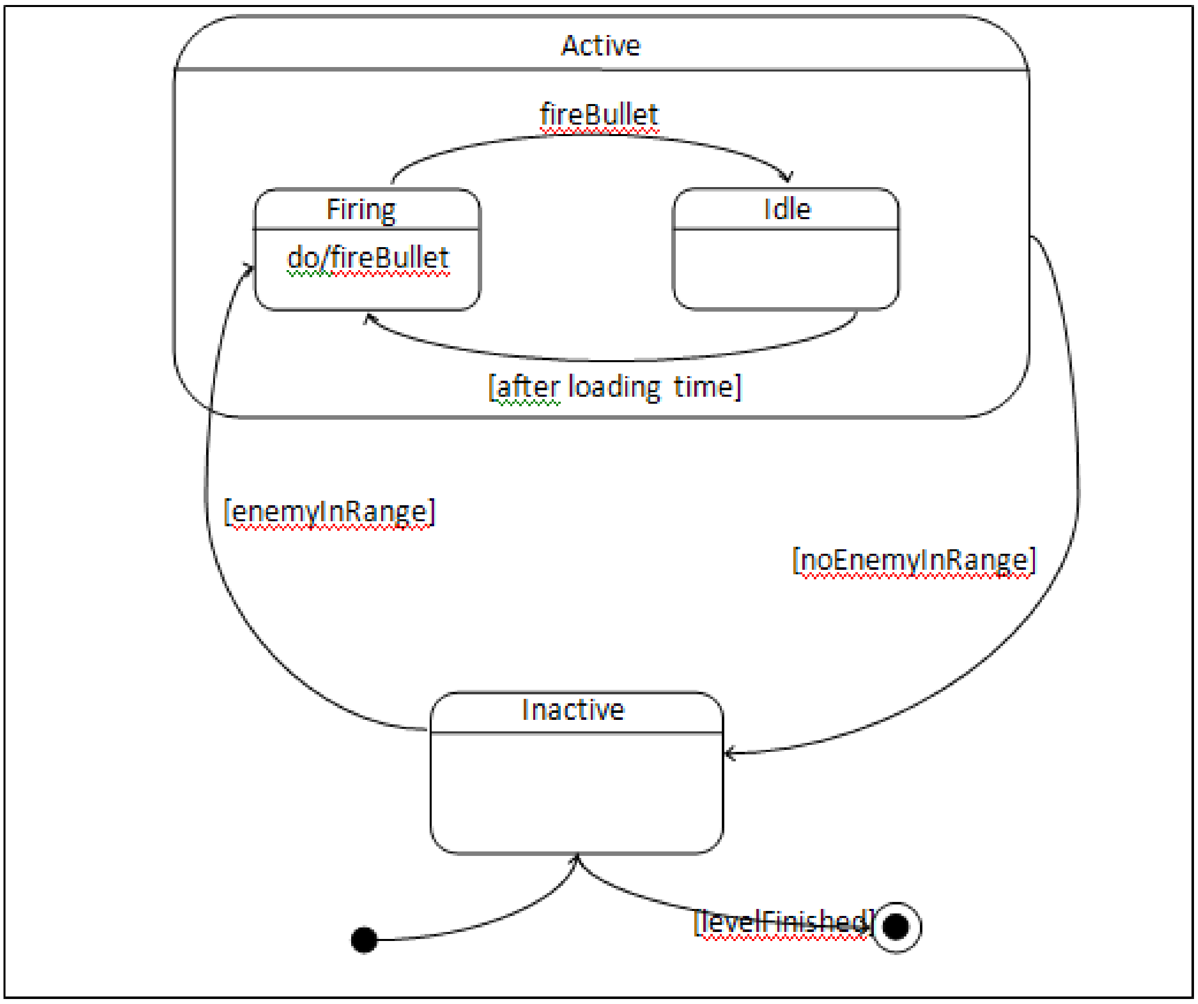


Figure 4.3: State Chart Diagram for Tower Object

In figure 4.3, states of Tower object have been diagrammed. There are two main states of Tower object; the Active and Inactive. When a level started, Tower starts in Inactive state. When there is an enemy in range it passes to the Firing state, which is a sub-state of Active state. In Firing state, Tower object calls fireBullet method and, whenever it calls fireBullet method, it passes to the Idle state. In Idle state, it waits for a “loading time”, which is predetermined by the properties of the Tower, and returns to the Firing state. Whenever in Firing or Idle state- i.e. in Active state- if there is no enemy in the range, Tower returns to the Inactive state. In Inactive state, when the level finishes, Tower exits state machine.

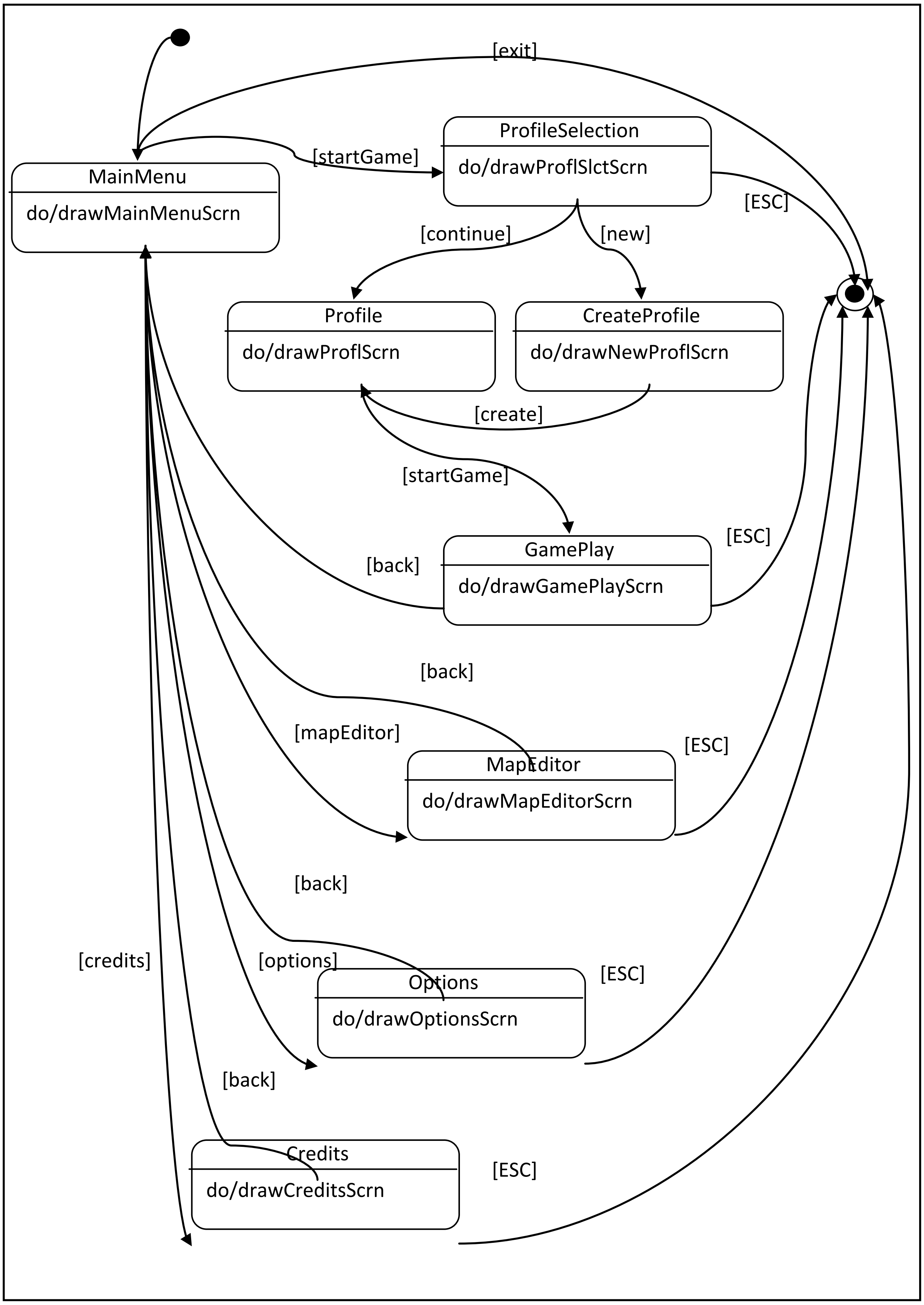


Figure 4.4: State Chart Diagram for GameScreenManager Object

In figure 4.4, the states of GameScreenManager object is diagrammed. GameScreenManager is the object that decides what is to be drawn on the screen. It starts with a MainMenu state, in which it draws the main menu.

In MainMenu state

- If start game button is pressed, GameScreenManager object passes to ProfileSelection state.
- If map editor button is pressed, GameScreenManager object passes to the MapEditor state.
- If options button is pressed, GameScreenManager object passes to the Options state.
- If credits button is pressed, GameScreenManager object passes to the Credits state.
- If exit button is pressed, GameScreenManager object finishes the state machine.

In ProfileSelection, GamePlay, MapEditor, Options and Credits states, if ESC button on the keyboard is pressed, GameScreenManager object passes to the final state.

In ProfileSelection state, If one of the profiles are selected and continue button is pressed, GameScreenManager object passes to the Profile state. If new button is pressed, GameScreenManager object passes CreateProfile state.

In CreateProfile state, if create button is pressed, GameScreenManager returns to Profile state if back button is pressed. GameScreenManager object returns to Profile state.

In Profile state if start game button is pressed, GameScreenManager passes to GamePlay state if back button is pressed. GameScreenManager returns to MainMenu state.

In Profile, GamePlay, MapEditor, Options and Credits states, if back button is pressed, GameScreenManager object returns to MainMenu state.

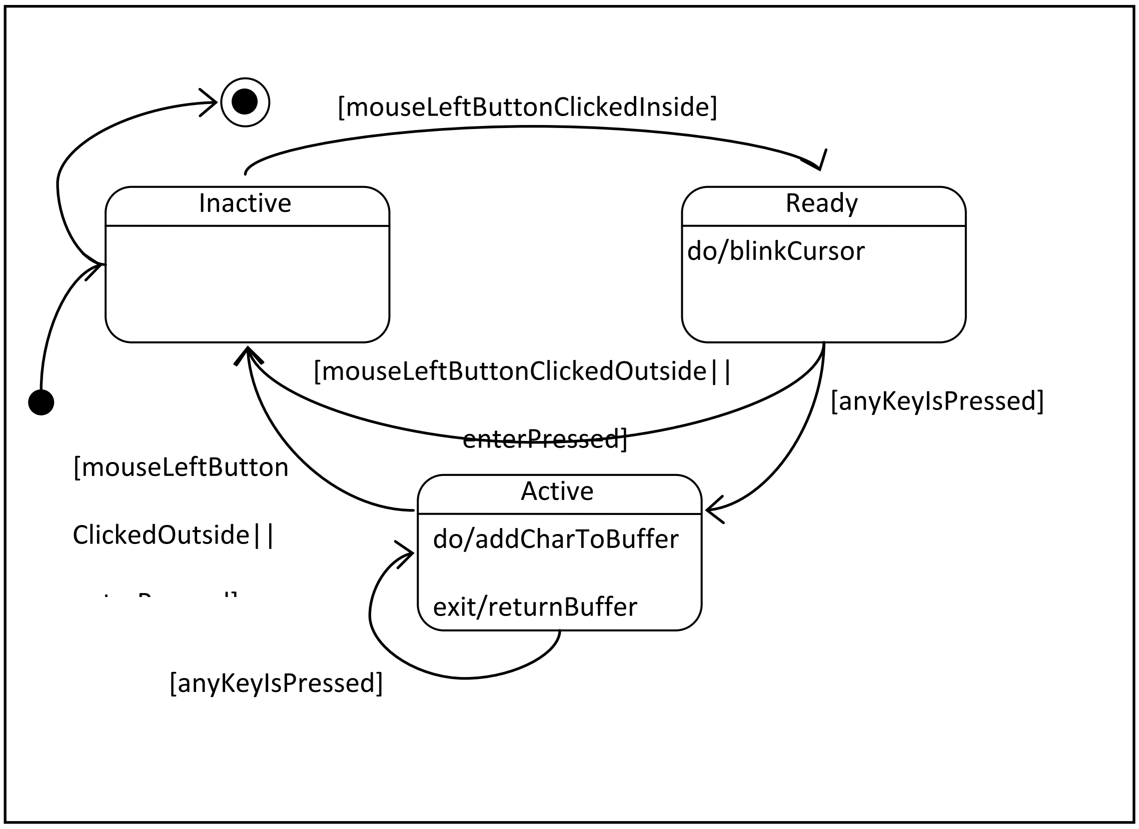


Figure 4.5: State Chart Diagram for InputField Object

In figure 4.5 , states of the InputField object is diagrammed. InputField object starts in Inactive state. When the left button of the mouse is clicked inside the InputField, it passes to the Ready state. In Ready state, the curser blinks in the InputField. When any key on the keyboard is pressed, the InputField passes to the Active state. In the Active state, any key pressed on the keyboard is stored in the buffer and when InputFields exits from the Active state, the buffer is returned. If the left button of the mouse is clicked outside the InputField or enter key is pressed, InputField passes to the Inactive state.

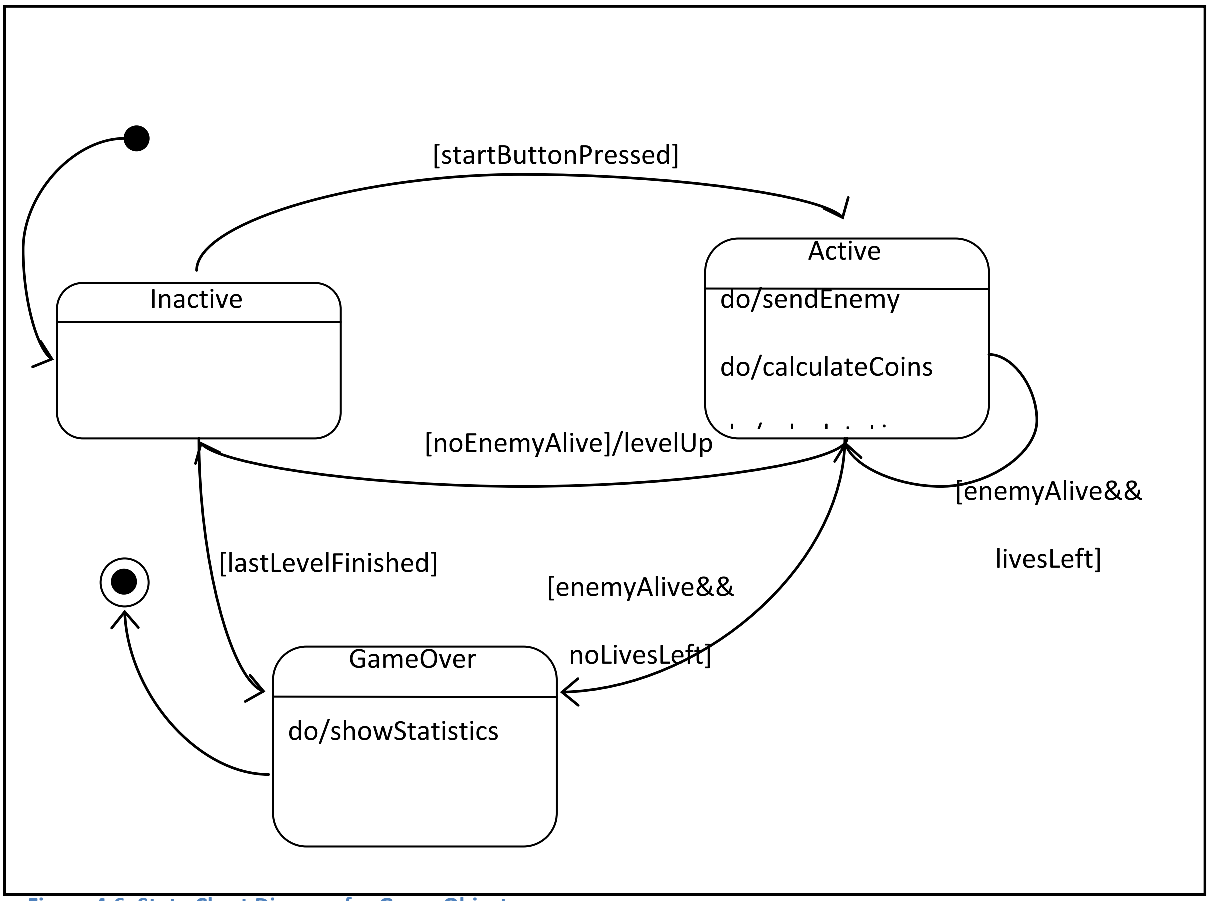
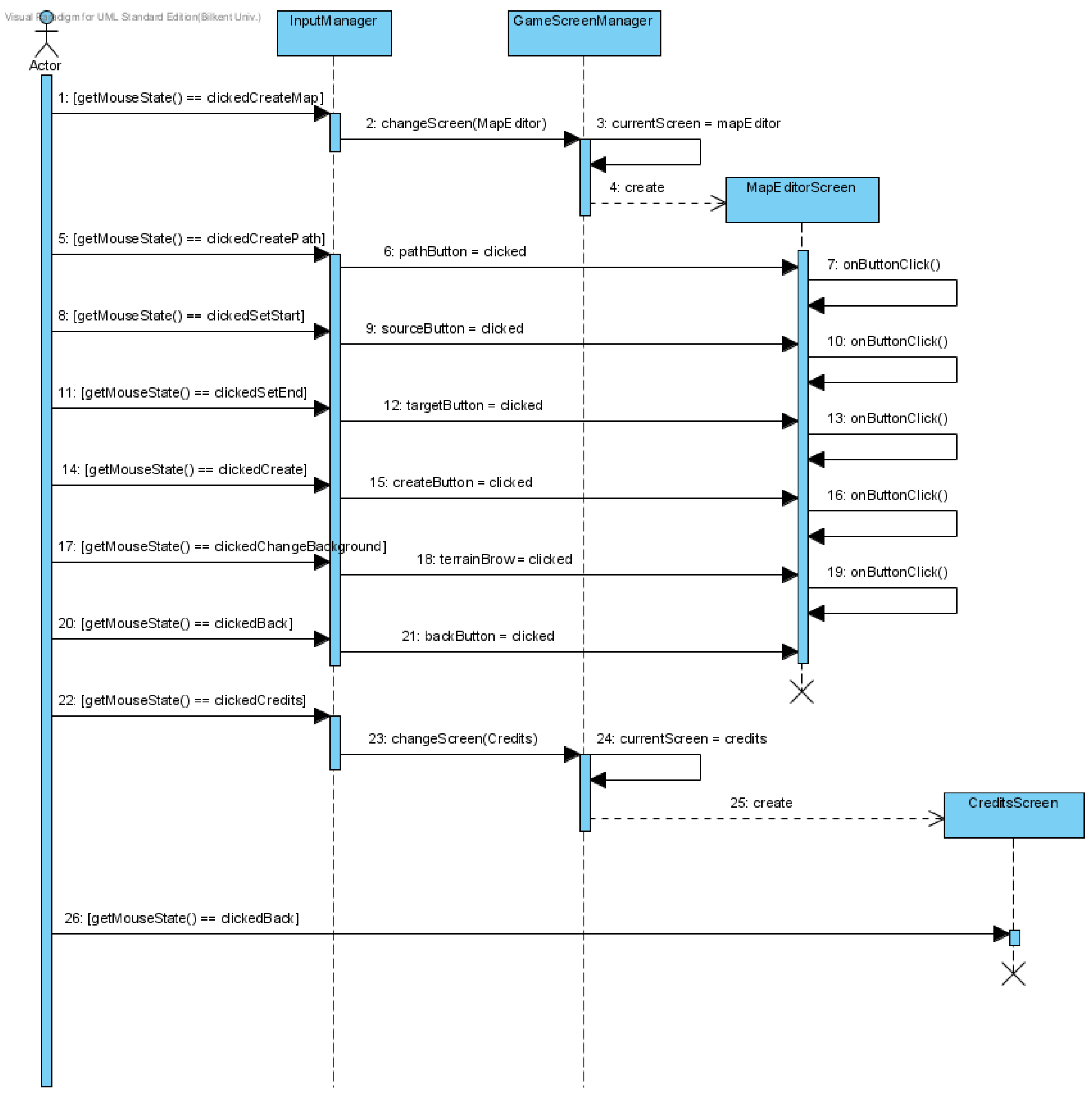


Figure 4.6: State Chart Diagram for Game Object

In figure 4.6, states of the Game object are diagrammed. Game object starts in Inactive state. When start button is pressed the Game passes to Active state. In Active state, Game sends the enemies in pre-defined numbers according to level, calculates the coins and lives. If last enemy is dead and there are lives left in Active state, Game passes to Inactive state and increments level. If there are enemies alive and there are no lives left in Active state or last level is reached in Inactive state, Game passes to GameOver state. In GameOver state Game shows the statistics and finishes the state machine.

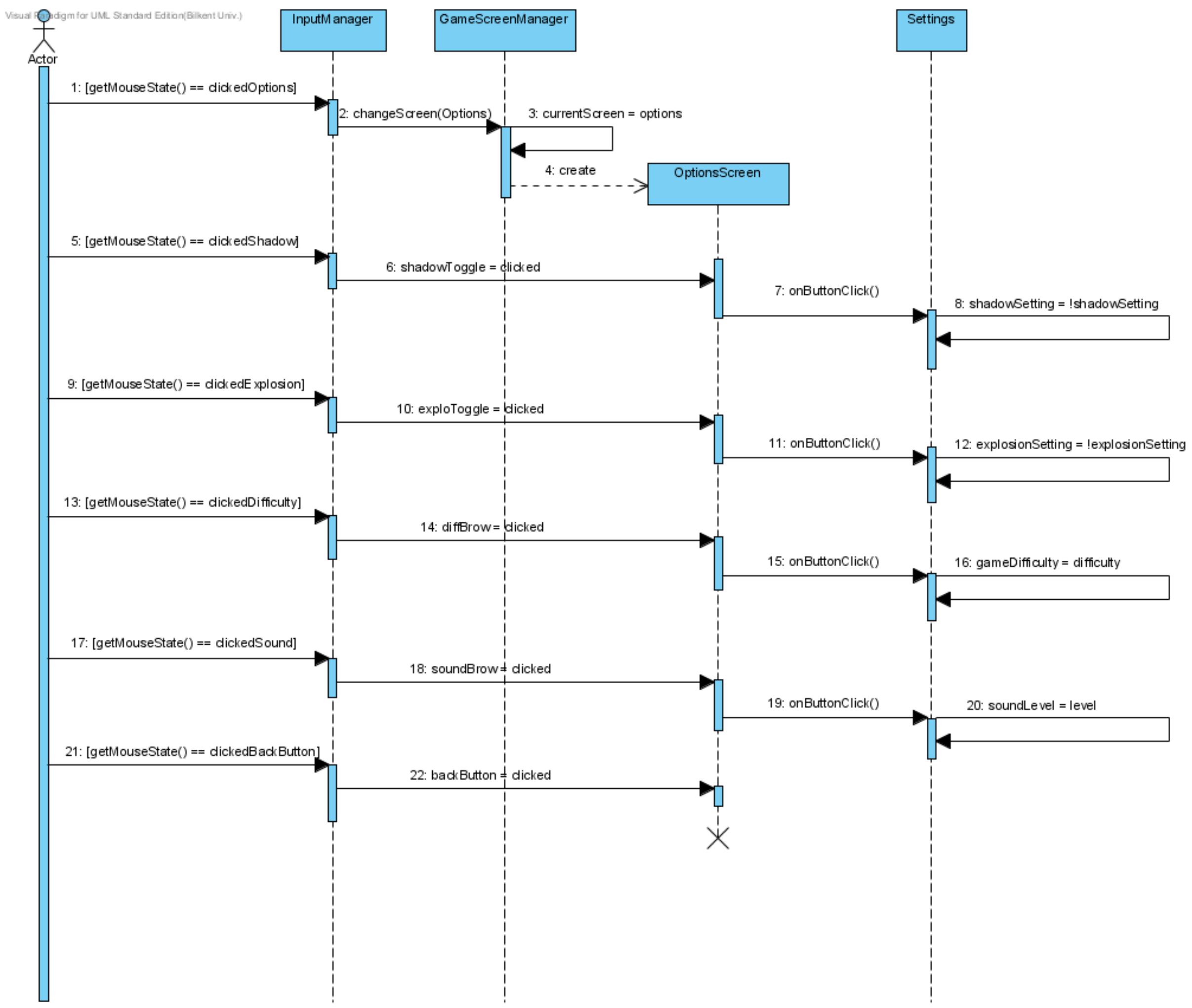
4.2.2 Sequence Diagrams



Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Actor
InputManager
GameScreenManager
1: [getMouseState() == clickedCreateMap]
2: changeScreen(MapEditor)
3: currentScreen = mapEditor
4: create
MapEditorScreen
5: [getMouseState() == clickedCreatePath]
6: pathButton = clicked
7: onButtonClick()
8: [getMouseState() == clickedSetStart]
9: sourceButton = clicked
10: onButtonClick()
11: [getMouseState() == clickedSetEnd]
12: targetButton = clicked
13: onButtonClick()
14: [getMouseState() == clickedCreate]
15: createButton = clicked
16: onButtonClick()
17: [getMouseState() == clickedChangeBackground]
18: terrainBrow= clicked
19: onButtonClick()
20: [getMouseState() == clickedBack]
21: backButton = clicked
22: [getMouseState() == clickedCredits]
23: changeScreen(Credits)
24: currentScreen = credits
25: create
CreditsScreen
26: [getMouseState() == clickedBack]

Figure 5.1: Sequence Diagram for Use Case Model of the System

In Figure 5.1, user's interaction to reach and use the Map Editor Screen is explained in detail. InputManager controls the input devices that user commands, and by following their states and interactions on user interface elements, onButtonClick() event listener is called. Corresponding operations are done, as it is requested by the user. This set of possible actions is corresponding with given scenarios.



Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Actor
InputManager
GameScreenManager
Settings
OptionsScreen
1: [getMouseState() == clickedOptions]
2: changeScreen(Options)
3: currentScreen = options
4: create
5: [getMouseState() == clickedShadow]
6: shadowToggle = clicked
7: onButtonClick()
8: shadowSetting = !shadowSetting
9: [getMouseState() == clickedExplosion]
10: exploToggle = clicked
11: onButtonClick()
12: explosionSetting = !explosionSetting
13: [getMouseState() == clickedDifficulty]
14: diffBrow = clicked
15: onButtonClick()
16: gameDifficulty = difficulty
17: [getMouseState() == clickedSound]
18: soundBrow = clicked
19: onButtonClick()
20: soundLevel = level
21: [getMouseState() == clickedBackButton]
22: backButton = clicked

Figure 5.2: Sequence Diagram for Use Case Model of the System (cont.)

In Figure 5.2, user's possible interactions to reach and use options menu from main menu is explained in detail. As in previous case, user's actions are controlled by managers and interface listeners. This set of possible actions is corresponding with given scenarios.

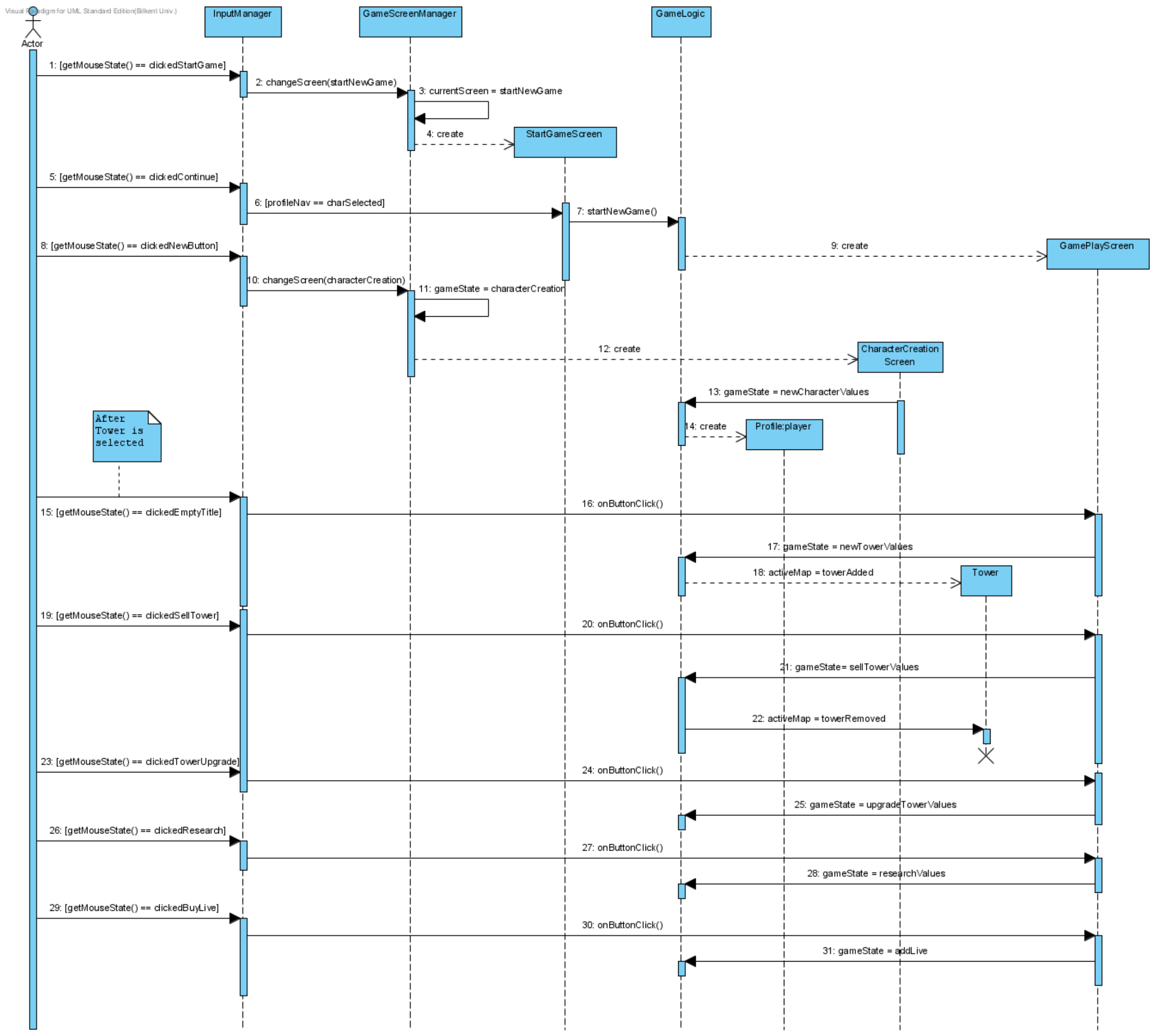

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Actor
InputManager
GameScreenManager
GameLogic
1: [getMouseState() == clickedStartGame]
2: changeScreen(startNewGame)
3: currentScreen = startNewGame
4: create
StartGameScreen
5: [getMouseState() == clickedContinue]
6: [profileNav == charSelected]
7: startNewGame()
8: [getMouseState() == clickedNewButton]
9: create
GamePlayScreen
10: changeScreen(characterCreation)
11: gameState = characterCreation
12: create
CharacterCreation Screen
13: gameState = newCharacterValues
14: create
Profile:player
After Tower is selected
15: [getMouseState() == clickedEmptyTitle]
16: onButtonClick()
17: gameState = newTowerValues
18: activeMap = towerAdded
Tower
19: [getMouseState() == clickedSellTower]
20: onButtonClick()
21: gameState= sellTowerValues
22: activeMap = towerRemoved
23: [getMouseState() == clickedTowerUpgrade]
24: onButtonClick()
25: gameState = upgradeTowerValues
26: [getMouseState() == clickedResearch]
27: onButtonClick()
28: gameState = researchValues
29: [getMouseState() == clickedBuyLive]
30: onButtonClick()
31: gameState = addLive

Figure 5.3: Sequence Diagram for Use Case Model of the System (cont.)

More detailed than others, Figure 5.3 shows how user can play the game, interacting with the user interface, following a set of actions. This set of possible actions is corresponding with given scenarios.

5. System Design

5.1 Design Goals

As in all engineering solutions, the system to be developed should also have prioritized goals, and should optimize its resources to achieve them. Since the game to be developed is a personal computer game, system should actually “entertain” casual computer users, who do not necessarily have good knowledge (or patience) to deal with difficulties that the system can create. Also, as in all games, the system should focus on a good gamin experience. Since this is a class project aimed to be an example of good software engineering, best methods of development should be followed.

Reliability: The system should be reliable. Since reliability is one of the fundamental design goals it must be reached in Tower Defense Game, too. The system should run consistently and no data loss should occur. Even though it is not a life or death situation but a game for fun; it is still frustrating to get different results each time a set of inputs are applied. For example, a tower should fire and hit the enemies in its range each time. Otherwise, winning or losing will be left to pure chance and there is no meaning in playing. Reliability requires a strong implementation and testing process but it is aimed to allocate enough time and recourses to implementation and testing to reach reliability.

Modifiability: In order to provide different gaming experience to players, modifiability is a key factor. With high degree of modifiability, it is granted that developers can add new content easily. It is important for our project since it is a game and requires lots of content and modification of this content with game play testing. For example, when a particular tower is extremely strong that makes game very easy or a level that is extremely hard, developers should be able to modify particular objects to make it balanced.

High Performance: The system distinguishes from other contemporary games by being a stand-alone desktop application, rather than limited application that is supposed work with other programs. Thus, the system should be able to use this advantage and be able to benefit from computer resources allocated for it. The system will be designed to use modern game frameworks which provide rich auditory and visionary experience, by not sacrificing much from the cost. A high performance is especially important for computer games; since they usually demand more resources to be used than other programs, and users quickly recognize a lack of performance in a game and are usually bothered with this. Since the whole idea is allowing users to have fun; a high performance is a must in a game. Thus, the system aims to provide at least a standardized 60 frames per second, and should be accelerated by XNA Framework and DirectX libraries.

Good Documentation: Documentation is aimed to be well organized and thorough, because it it important for users to have a guidebook or manual before playing the game. Also, it is very crucial to have a detailed documentation of the analysis, design and development processes for other developers that like to continue the project. For these purposes, in each step time and man power is allocated for the documentation of project. As well as necessary and crucial documentation, we spared some time to make a background and a story for the game. It is fun and a quality that is looked by dedicated players to have a manual explaining the backgrounds and properties of items in the game.

Minimum Number of Errors: One of our main design goals was reliability. In order to develop a reliable system, number of errors should be kept at minimum. Errors may result in loss of game data and unreliable gameplay; therefore it is vital to eliminate errors among system. Most games are often unplayable due to errors and users often prefer other alternatives. To prevent this, project must be free of errors and therefore must be tested thoroughly during development stage.

Ease of Learning: The use of options is very explicit. It does have complex structure of options menu. At the beginning of the game, tutorial will be given to user. If user needs, he will read and utilize from this documentation. The logic of game will be well-understood. Afterwards, it does not require extra explanation to guide user. It is self explanatory. User clicks with right button while buying any tower, he will see the function of it.

Ease of Use: The interface of game is very well understood. User can recognize   which buttons are used . User clicks with right button while   buying any tower, he will see the function of it. The buttons` names point to their functions. The design of graphics is very well representative. The parts of screen   manager are divided logically.

Trade-Offs

Naturally, all optimization problems come with drawbacks. Although we want to minimize them, expected sacrifices are as below

Cost (High Performance, Reliability): The system should be designed as a stand-alone game application, which benefits from high level frameworks and modern technologies. These features, however, are in conflict with the cost-effectiveness of the system, and as an engineering practice it is decided that cost may be sacrificed. In computer games, however, it is not unusual to sacrifice a system's cost to its reliability or high performance; since these two are usually more important for gamers than systems cost- gamers prefer to have costly systems in terms of both hardware and software. However, the system will use XNA Framework and DirectX libraries (both provided by Microsoft), which will assist designers to reach these two goals under a managed code, by not sacrificing much from the cost effectiveness. In addition, although high performance and reliability are usually in conflict with each other, using corresponding external sources to assist these goals will minimize any such conflict that may arise.

Functionality (Ease of Use, Ease of Learning): The system is designed to be addressing all user groups, from novice to experts. For this very reason, it is inevitable to sacrifice some possible extra features for the sake of usage and learning ease. In such designs, a system is either too complex with many functions, or plain and simple but relatively basic. Since this system is designed for primarily entertainment purposes it is only natural to choose the latter alternative. A simple user interface and explanatory guidelines are used instead of developing new functions. Furthermore, this choice overlaps with the existing conventions of such tower defense games. To follow these conventions, the system avoids having confusing new functions that may cause any difficulties for the user.

Rapid Development (Good Documentation, Minimum Number of Errors): Although the designated implementation stage is relatively short and requires quick coding in our case; this stage cannot be considered as the whole "development" of the system. The development of a program starts from its initial design steps to the last testing. Thus, any effort that may directly or indirectly extend this time interval will be conflicting with rapid development of a program. The system follows a top-down hierarchical design paradigm, aiming to follow good software engineering methods. A rapid development, however, focuses more on the implementation rather than the organization. A good documentation that aims to minimize ambiguity both in project requirements and design decisions are directly conflicting with this goal. In addition, the system will try to be robust and effective and it is planned to spend a good deal of effort in terms of error minimization, from design to testing. Thus, this goal design will result a natural trade off.

5.2 Sub-System Decomposition

Our system consists of following subsystems:

InputManagement Subsystem
Sound Management Subsystem
Viewer Subsystem
    Game Screen Management Subsystem
    Screen Elements Subsystem
Game Logic Subsystem
    Logic Management Subsystem
    Profile Management Subsystem
Data Management Subsystem

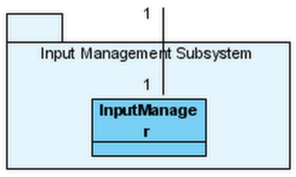


Figure 6.1: Input Management Subsystem

Input Management Subsystem: This subsystem's main function is to observe user actions, detect inputs and pass these inputs to the other subsystems. It records mouse movement, mouse position, mouse actions (click, drag etc.) and keyboard actions (which key is pressed or released etc.) Only class that is present in this subsystem is Input Manager class and is responsible for all input management.

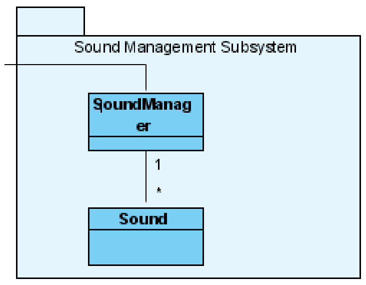


Figure 6.2: Sound Management Subsystem

Sound Management Subsystem: This system handles all kind of sound effects and music that will be played in the game. When notified, this system is responsible to play appropriate sound effect or music. In this subsystem, Sound class represents any kind of sound or music entity while SoundManager class specifies when and how these effects will be played.

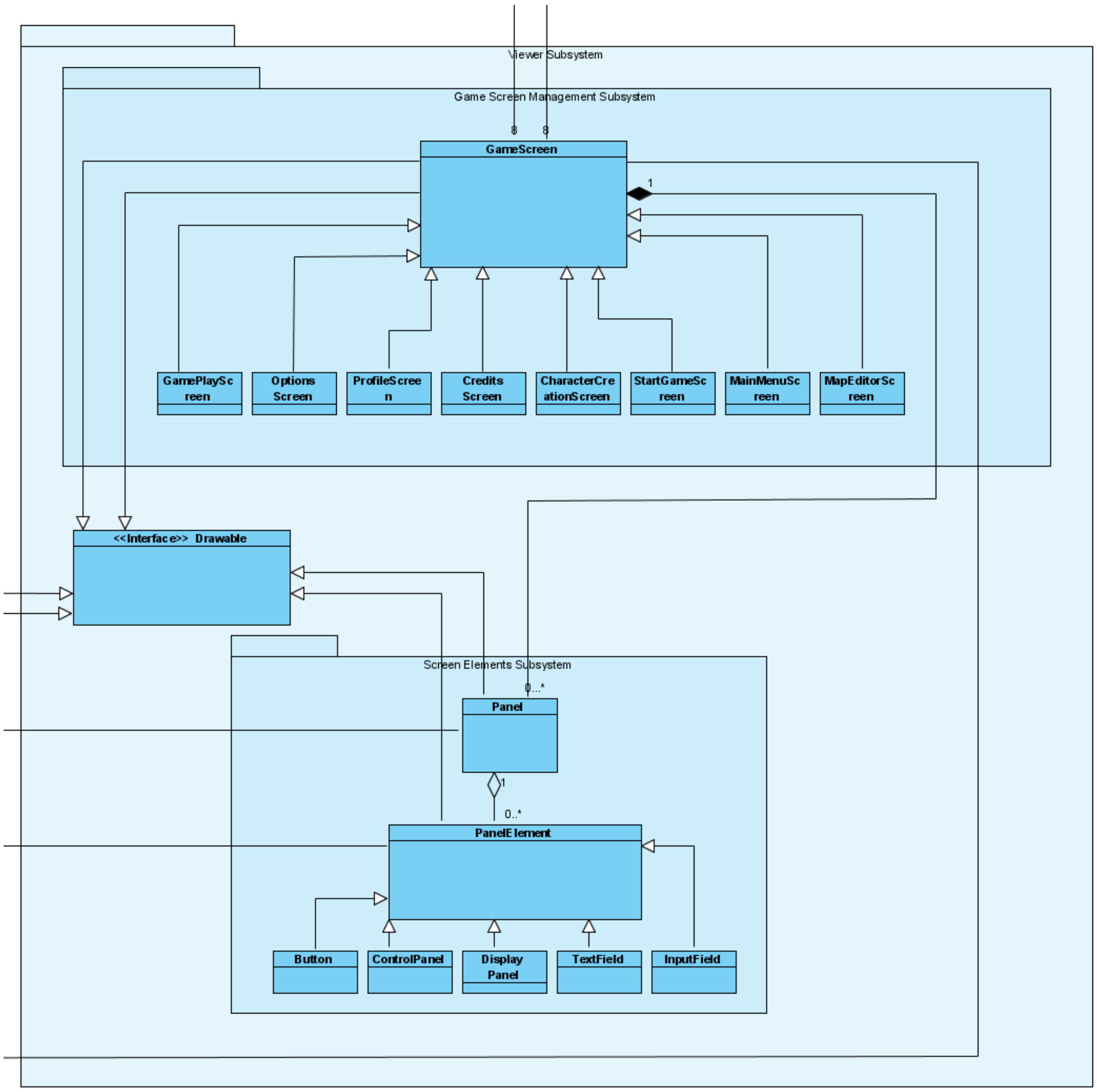


Figure 6.3: Viewer Subsystem

Viewer Subsystem: Viewer Subsystem's main concern is to present game model part of the game to the users with graphics. It contains all kind of things that is represented as graphical objects like images, buttons, panels, texts etc. It holds all game screens and manages transitions between them. In order to further specialize this system, it is divided into two subsystem named as Game Screen Management Subsystem and Screen Elements Subsystem.

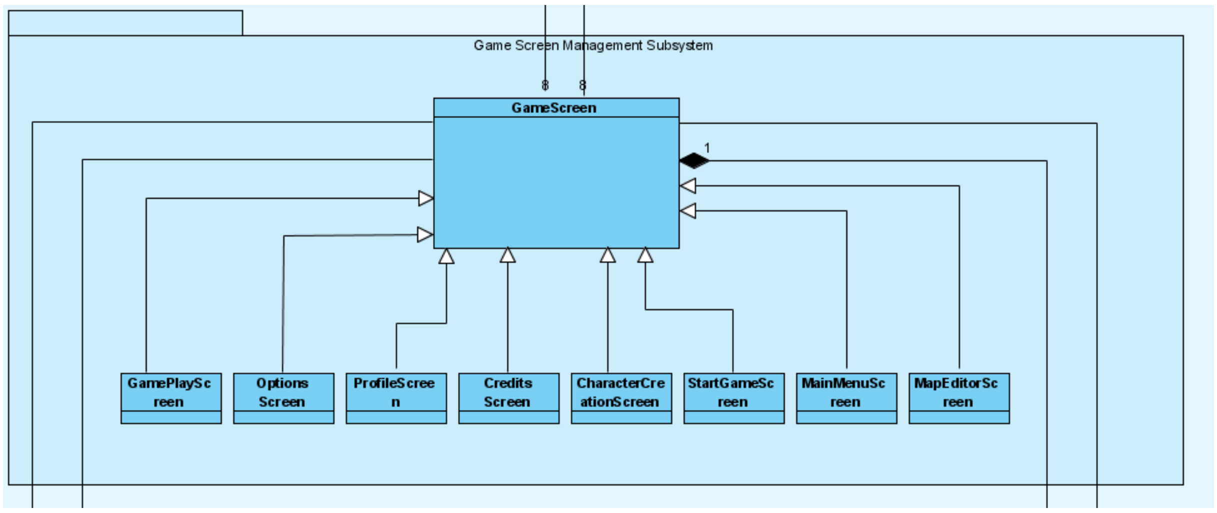


Figure 6.4: Game Screen Management Subsystem

Game Screen Management Subsystem: This system is responsible for managing game screens. It stores current state of the game screen (ex: active screen that is displayed on the screen) and with given inputs decides which game screen is to be displayed.

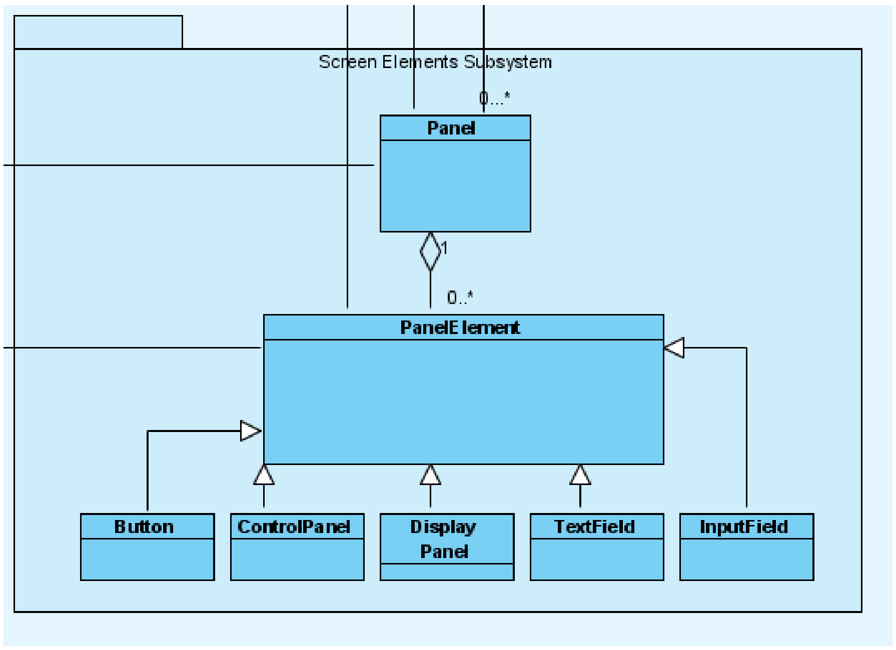


Figure 6.5: Screen Elements Subsystem

Screen Elements Subsystem: This system serves as a collection of all displayable elements that can be used in typical game screen. Some of these elements are buttons, textfields and input fields and can be added to any game screen to add functionality to these screens. These elements also respond to user inputs (ex: buttons).

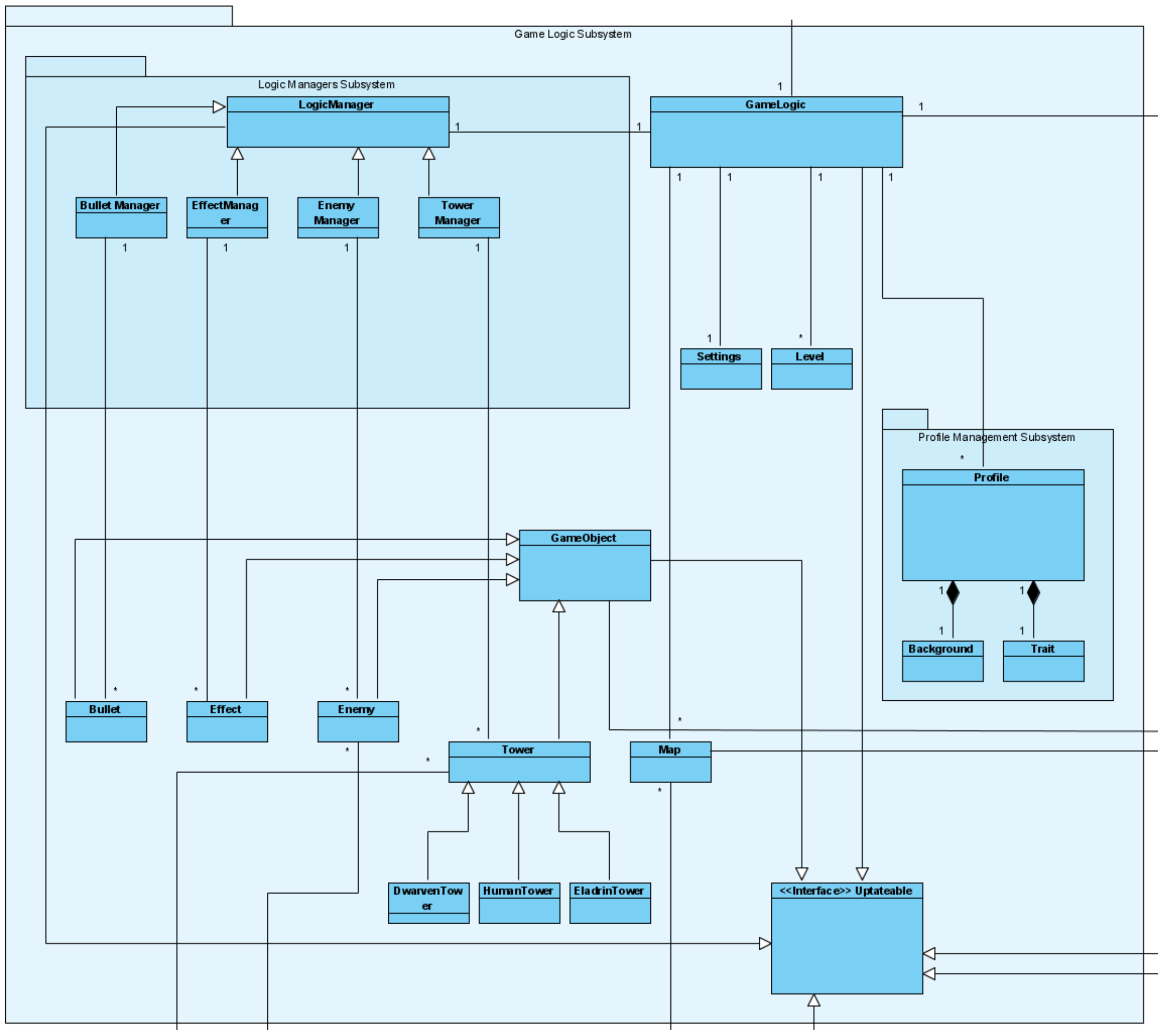


Figure 6.6: Game Logic Subsystem

Game Logic Subsystem: This subsystem represents model of our system that is abstracted from user. It stores and manipulates all kind of game data: Towers, enemies, bullets, maps, effect states, logic managers, profiles etc. With given inputs, it updates itself. All gameplay related calculations are done in this subsystem. Since it is complex system, it has two more subsystems named as Logic Management Subsystem and Profile Management Subsystem.

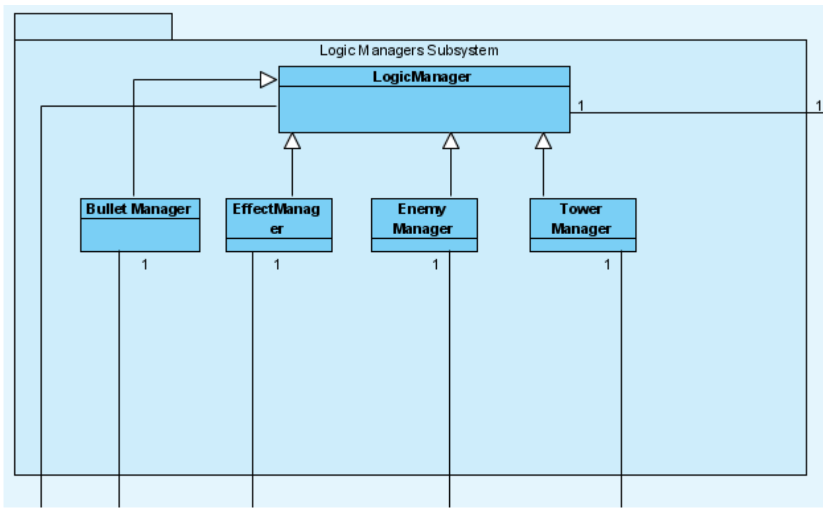


Figure 6.7: Logic Managers Subsystem

Logic Managers Subsystem: This subsystem contains manager classes namely Effect Manager, Bullet Manager, Enemy Manager, Tower Manager. These manager classes are responsible for storing and updating game entities like bullets, enemies, towers, effects. Logic Management Subsytem needs game objects to operate.

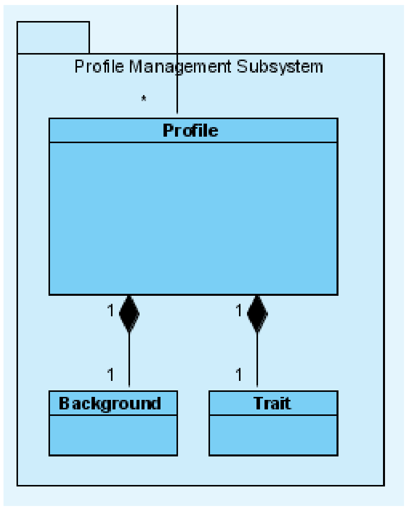


Figure 6.8: Profile Management Subsystem

Profile Management Subsystem: This subsystem contains Profile, Trait and Background classes which are directly related to profile specifications. It contains and manages all profile information that is used in the game.

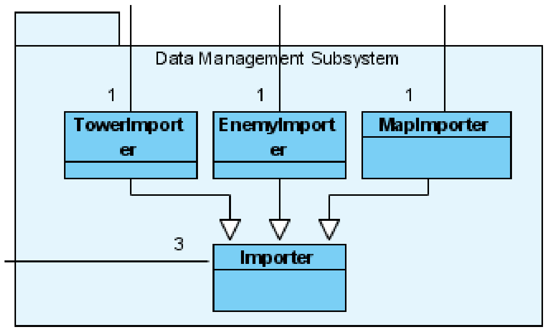


Figure 6.9 : Data Management Subsystem

Data Management Subsystem: Tower Defense Game stores some object properties externally in txt files. Towers, enemies, and maps are example of these. When game starts, all information in these file will be used to initialize these objects. Therefore, to handle all data interactions, Data Management Subsystem is responsible. It traverses all game data that is stored in files at the beginning of the game, loads them. When change is needed (ex: user created a new map), this subsystem is responsible for adding these contents to the specified files.

5.3 Architectural Patterns

The system architecture will be designed mainly following two methods, layer architecture and model view controller (MVC) architecture. The system to be developed has both hierarchical and functional separations amongst its classes, and these problems are addressed by these architectural choices relatively. So, design process of the system consisted of the examination, evaluation and adaptation of these options; benefiting from both but mainly depending on the MVC, since the system to be developed has many classes that are not strictly bounded to a specific layer and is defined in its place by its function, rather than its relative level.

LAYER PATTERN DESIGN

The system to be developed has a hierarchical relationship amongst related classes to a degree, and it is possible to specify layers to make this distinction. A layer would be a group of classes that have the same set of concerns and relative importance, and is closer to the user as it gets higher. The system's layers are group of reusable components that are reusable in similar circumstances.

To generalize, the highest level layers will be the classes that are in immediate interaction with the user: namely the interfaces, graphical and auditory supports and panels. These are handled by screen classes in our design (panels, labels, screens et cetera). Layers in middle classes are followed by the manager classes that process inputs, changes and state transformations of the game. Lowest level classes are actually conceptually more related to the domain; however, they are abstracted from the user. Their services are provided to the middle layers, and form the base of the layer pattern.

This architectural choice, although useful, would not satisfy the relationships between classes, especially for the controller class. Conceptually, user interface classes sit on the top of the layers, however, it should be noted that processed changes are also directed to these layers, meaning that there should also exist a bottom-up flow of information. Also, layers cannot be concretely separated as different branches of a layer tree. Thus, although has its own advantages and surely will influence the resulting architecture choice (such as pointing out a somewhat vague hierarchy); layer pattern limits the relationship between classes, even if an open layer architecture is used.

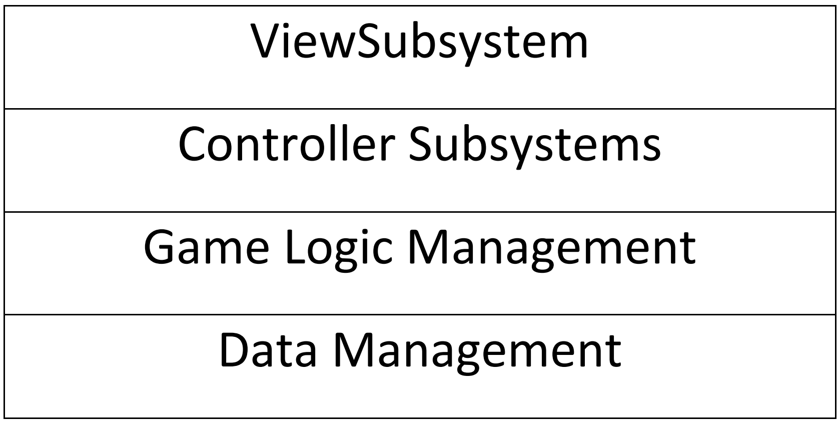


Figure 7.1 : Layer Pattern Design

MODEL VIEW CONTROLLER

The system to be developed consists of many agent objects that are used in the logic of the game, rapid usage and interaction with the user and an engine in behind to realize the application itself; because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere. High coupling makes classes difficult or impossible to reuse because they depend on many other classes. Adding new data views often requires re-implementing or cutting and pasting business logic code, which then requires maintenance in multiple places. Especially, considering the game being developed, it is predicted that there will be lots of agents that have their own attributes and operations; logic to maintain the relationship amongst these agents and the environment; and virtually all changes in the game should be reflected to the user. Thus, an architectural pattern choice to separate all these concern is needed.

The Model-View-Controller (MVC) design pattern solves these problems by decoupling game agents, game logic, and graphical presentation and user interaction. In the system design, game agent objects such as towers, creatures, bullets, and the map are models that represent the domain specific objects. These objects should be controlled, modified, which is the responsibility of the controller part of the architecture. Their modifications, although creates the domain knowledge, is not directly controlled by themselves, nor the user. Classes that form the game logic are responsible for this, which are in the controller part. Model objects also store vital information of themselves, needed by both controller and view classes.

The controller part of the architecture consist of the manager classes; game logic, tower manager, bullet manager, enemy manager, screen and sound manager. Also, inputs and outputs that effect the game logic are also controller classes; such as the input manager. These classes create the game logic, with a certain hierarchy amongst themselves. The controlling issues can be summarized as taking inputs either directly or from view classes, changin them to data that is directed to be processed and modified in the model and reflect these changes on the view, to the user.

The view part of the architecture directs the interactions between the system and the user. These interactions are mainly dealt in interface classes; which consist of all screen classes, panel and label elements. This part of the architecture uses much information from the models, however, is generally controlled and sends interactions to the controller.

MVC is often seen in game applications and game engines where the functions of view and control are specifically separated, and game content is preferred to be kept in the model part. Thus, it would be a good engineering practice to use the MVC design pattern in the system.

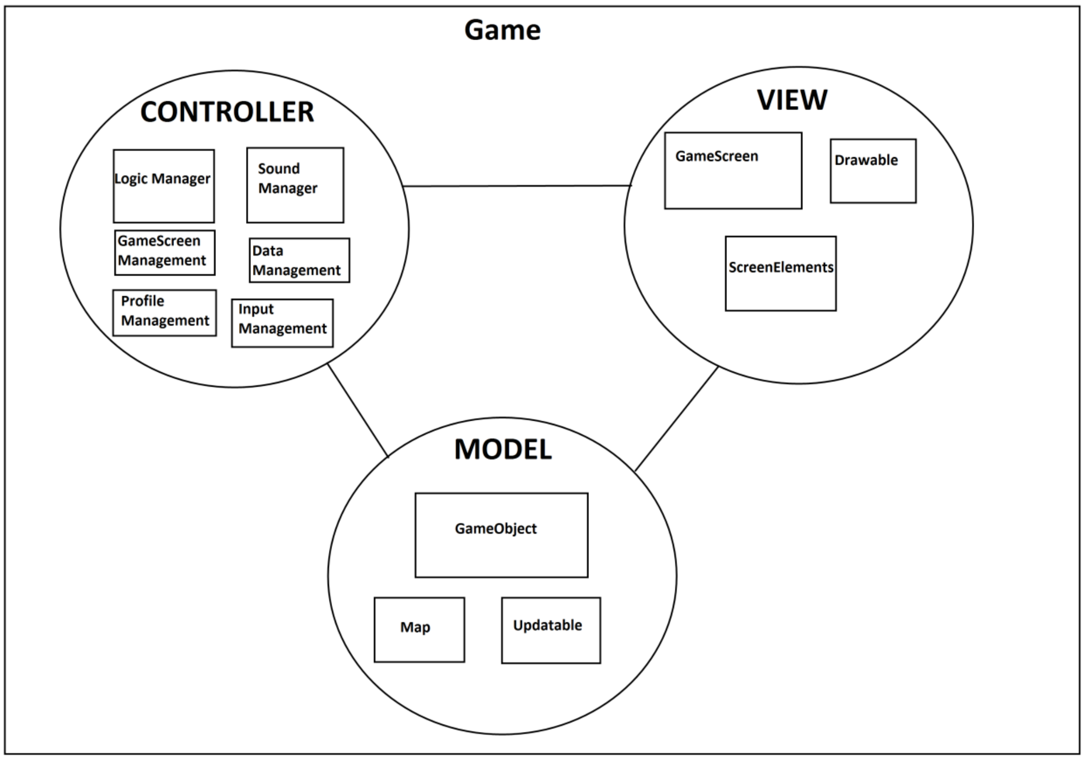


Figure 7.2: Model-View-Controller Pattern Design

5.4 Hardware/Software Mapping

Tower Defense Game is a standalone application, which will not require any kind of web or network system to operate. Only requirement is XNA Redistributable package that can be installed from web. In order to run Tower Defense Game, any standard desktop pc's configuration will be enough. Since game uses directX libraries, modern video card sith directX support is required.

In terms of memory management, Tower Defense is a lightweight game, memory will mostly used by game logic which is not demanding since all drawing – drawing related operations will be handled by GPU.

I/O demand for project is very low; there is no exhaustive access from hard drive or any other media. Hard drive access will be used in initialization time (ex: reading files to initialize enemy stats) and won't require much access anyway.

Being standalone application, once installed to hard drive, our projects subsystems will be mapped to hard drive, memory and video card. Before runtime, all software will reside on hard drive. During runtime, objects will be created and used in memory while data files (txt files for initialization or saving purposes) resides in hard drive. Other than that, graphical context (Textures, animations) will be executed in GPU which will be managed by XNA Framework.

Deployment Diagram: As a standalone application, one PC Machine will be enough to run once XNA Redistributable package is installed on system. This package can be downloaded from internet. Users can install Tower Defense Game from CD's or via downloading from internet. It will have a installation wizard like any modern software to help users to install our application easily.

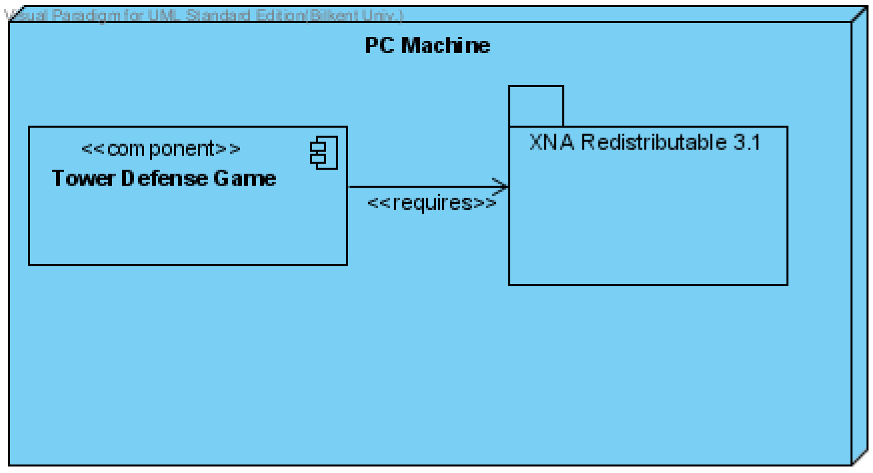


Figure 8: Deployment Diagram

5.5 Persistent Data Management

While managing persistent data, the file will be chosen. Because of their simplicity text files are commonly used for storage of information. They avoid some of the problems encountered with other file formats. Further, when data corruption occurs in a text file, it is often easier to recover and continue processing the remaining contents during the game. A disadvantage of occupation of more storage than is strictly necessary. Another reason of choosing file is that providing cheap, permanent storage. The operation of reading and writing are done during the game. In addition to this, data are kept only for short time. It does not require requesting queries. The system of the software does not depend on multiple users. Our system is for single user. Furthermore, our data are not changed during the game, only some data are added. Despite of the shortage of modification process, the system does not need database system. Multiple application programs do not access the data so the system of software does not require database system.

The organization of file system is explained as below:

Towers,enemy,bullet,Texture,Avatar,Effect,Sound,Background,Path,Menu,Images.Map

These files except Map File are accessed but not modified.

Towers: There are 6 towers for each race. The total number of the towers is 18. Each tower has attributions like name, id, tier, fire range, texture, bullet and percentage of upgrade

Enemy: Each enemy object has attributions like name, id, speed, hit point and texture

Profile: Each profile is used like name, id, speed, hit point and texture, statics information (such as number of creatures hit), avatar, id

Map: This file utilized from background, start point, end point, path, convenient,, height, width

Bullet: It uses Name, Damage Rate, Damage Type, Speed, Damage Area Texture Effect Attack profile classes, texture, Avatar

Sound: The audio files are held

Image: The pictures of races, towers, and profile pictures are held.

Effect: Image, Sound are found

Avatar: Image, Sound, Texture are found.

5.6 Access Control and Security

There is only one actor interacting with the system, which is the gamer -the person who plays the game-, so that there is no need for an access control strategy for the system. Another actor and associated use-cases may be created such as an administrator that can modify the dynamics of the game -like the frequency of bullets fired by a tower- but there is no need to separate those use-cases and prevent the gamer to access those functions. Any function that is offered by the system should be accessible and usable by the only actor- the gamer. There may be a password protection on these functions but this is also found unnecessary because there is no crucial data that can be corrupted by any user, all changes -except for deleting some profiles or maps created after the game is installed- are reversible. Since this system is after all a game and the convention for game systems is that all profiles are open to all gamers, there is no access control worth explaining. The one and only actor has all the privileges and access all the methods -some of them are implicitly invoked by other object but after all the user starts the chain of invocations.

5.7 Global Software Control

Global control flow is event-driven in Tower Defense Game. Consistently with its Model-View-Controller architecture, program waits for an event, in this case a key stroke on the keyboard or a mouse click inside the buttons; and updates the views. Input Management Subsystem is responsible for listening the input devices and delivering events to Logic Management Subsystem. Logic Management Subsystem is the Model part of the architecture and it calculates the input and determines any change in objects and views. Viewer Subsystem does make that change in the views related with the event. The control is more decentralized because different objects decide on the actions by evaluating different events. There is no "spider" or main controller that knows everything and controls the flow of data. This choice of control is explained by, again, the need for performance. Because it is an interactive system, the need for high performance is supported by distributing the control and avoiding a possible bottleneck.

5.8 Boundary Conditions

Initialization: At initialization, program is loaded into the memory and code is started to be executed. For some user interface elements -like textures and their calculations- are handled by the GPU (Graphics Processing Unit); some data is loaded into the GPU. At the beginning of the initialization, gamer profiles, maps, sounds, level information and various images (enemy, bullet, tower images etc.) are read from files and objects are created. The first screen presented to gamer is the opening animation of the developer studio logo. After this short opening animation; main menu screen is presented to the gamer. Main menu screen has a set of buttons associated with different functionalities of the system: Starting a game, creating a map, changing settings etc. The theme of interface is coherent with the story and the background of the game.

Termination: The system is terminated altogether, only a few subsystems are allowed to be terminated individually. Data Management subsystem is terminated after initialization finishes; it reads files and creates objects and terminates. The overall termination occurs when user quits the game. During termination, all objects read from the file system is written back to the files in case of a change. If the object, for example a user profile, is not changed during the game, then writing back to the file is an unnecessary operation; therefore does not occur.

Failure: There is no additional recovery function other than those MS Windows offers. The data of the current session is lost in case of a failure, because data update to the file system occurs at termination of overall system. However, older files are not changed so they may be assumed to be a kind of back up. If the failure occurs during the termination, this may cause the corruption of data files and the next execution of program can be faulty.
It may be frustrating and upsetting to have an unexpected termination or loss of data; this is not crucial considering the aim of the system is entertainment. Since a drawback in performance is far more upsetting than the loss of points gained in a session; it is chosen to have no regular write back operations or autosaving. Therefore, the next execution after a failure is similar to any execution of the program.

6. Object-Design

6.1 6. Object-Design

Template Method Pattern

Template method pattern is a behavioral design pattern. It requires the existence of one class with abstract and non-abstract methods and several classes that inherit that abstract method. Each subclass implements the abstract methods differently. So to speak, the parent class provides a "template" for its children but determines the overall algorithm. Every design decision with polymorphism naturally results in using template method pattern. This pattern allows the polymorphism by method overriding and also prevents duplicate code by template class controlling the overall algorithm. The aim of using template method pattern is to refine the algorithm in subclasses.

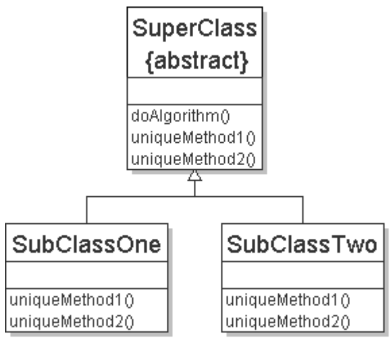


Figure 6.1: The UML representation of Template Method Pattern

In Tower Defense Game, *GameObject* class has a template method *update* which is inherited and implemented by *Tower*, *Enemy* and *Bullet* classes which are subclasses of *GameObject* class according to their own algorithms. GameObject ensures that all objects have the update method so that when *GameLogic* calls them they execute their respective algorithms.

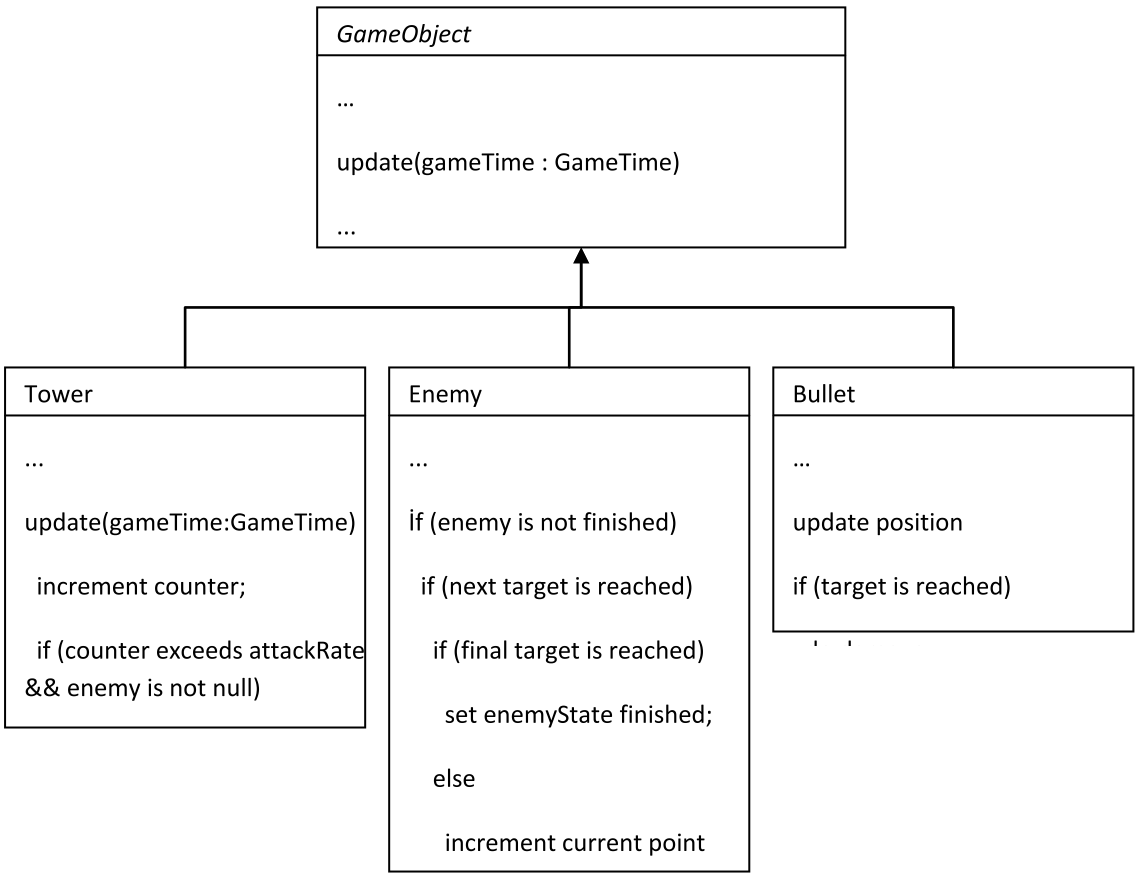

GameObject
...
update(gameTime : GameTime)
...
Tower
...
update(gameTime:GameTime)
increment counter;
if (counter exceeds attackRate && enemy is not null)
Enemy
...
if (enemy is not finished)
if (next target is reached)
if (final target is reached)
set enemyState finished;
else
increment current point
Bullet
...
update position
if (target is reached)

Figure 6.2: Template Method Pattern practice in Tower Defense Game

All *GameObject* s have the update button but they do different things in this method.

-*update* method in *Tower* object increments the counter until it reaches to a constant number –attack rate- then it restarts the counter. When counter is restarted, *Tower* checks if the bulletFired event is null or not, and if it is null *Tower* throws a bulletFired event do that the listener class –*GameLogic*- executes its related code.

-*update* method in *Enemy* object handles the movement of enemy through the target points –which are turning points on the road. It first checks if the *Enemy* object is reached the next target point or not. If it reached a target, it then checks if the target is the last target on the road, it means *Enemy* completed the road alive and sets the enemy state to finished. It the reached target is not the last target then it updates the current target –the next target on the road-, updates start point, updates direction, updates rotation.

-*update* method in *Bullet* object updates position by incrementing position by speed multiplied by direction. If the target is reached, it calls the *dealDamage()* method in enemy. Then removes the *Bullet* from the list kept in the manager.

Prototype Pattern

Prototype Pattern is a creational design pattern. In prototype pattern, an instance of an object is created only once in a class and whenever another instance of the same object is needed, instead of creating another instance, the instance created first is “cloned”. This method avoids the cost of creating new instances with “new” keyword. In Tower Defense Game, many instances of a class are needed, like whenever the user builds a tower, another *Tower* object should be created. Likewise, bullets are created many times by towers and enemies are created many times according to level. In order to be able to use prototype pattern, a *clone()* method is implemented in Tower, Bullet and Enemy classes.

TowerManager class has a method *addTower()* which is called whenever the user adds a tower to the game screen. However, instead of calling the constructor every time to create a new object, *addTower()* method calls *Clone()*method through the objects created in the constructor of *TowerManager*. The constructor of *TowerManager* has a prototype tower for each tower, which is used for cloning operation in the *addTower()* method of the class.

EnemyManager class has a method *update()* which is called by Game periodically when game is active state. However, instead of calling the constructor every time to create a new object, *update()* method calls *Clone()*method through the objects created in the constructor of *EnemyManager*. The constructor of *EnemyManager* has a prototype enemy for each enemy, which is used for cloning operation in the *update()* method of the class.

BulletManager class has a method *addBullet()* which is called whenever an enemy enters the range of a tower. However, instead of calling the constructor every time to create a new object, *addBulletr()* method calls *Clone()*method through the objects created in the constructor of *BulletManager*. The constructor of *BulletManager* has a prototype bullet for each bullet, which is used for cloning operation in the *addBullet()* method of the class.

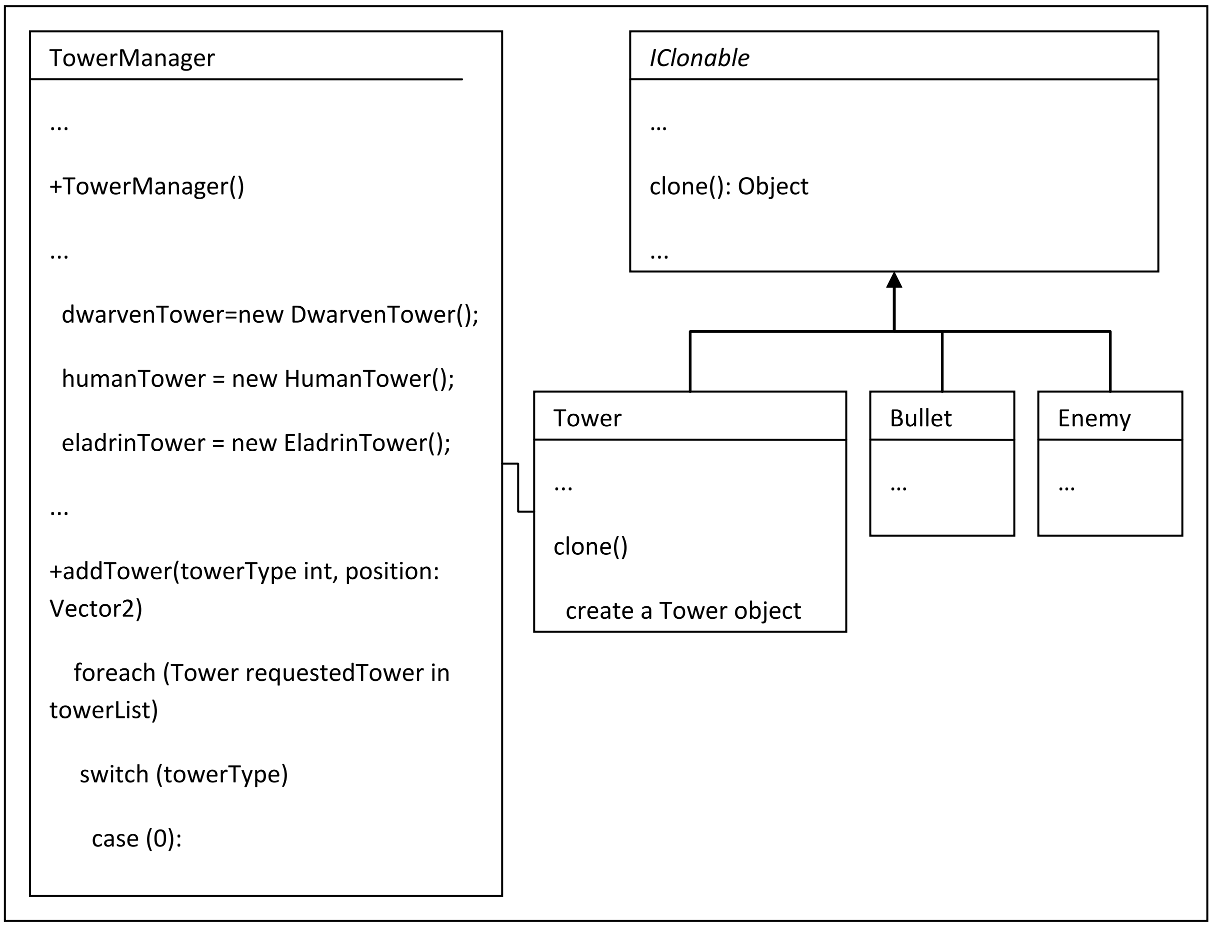

TowerManager
...
+TowerManager()
...
dwarvenTower=new DwarvenTower();
humanTower = new HumanTower();
eladrinTower = new EladrinTower();
...
+addTower(towerType int, position: Vector2)
foreach (Tower requestedTower in towerList)
switch (towerType)
case (0):
IClonable
...
clone(): Object
...
Tower
...
clone()
create a Tower object
Bullet
...
Enemy
...

Figure 6.3: Template Method Pattern practice on Tower object in Tower Defense Game

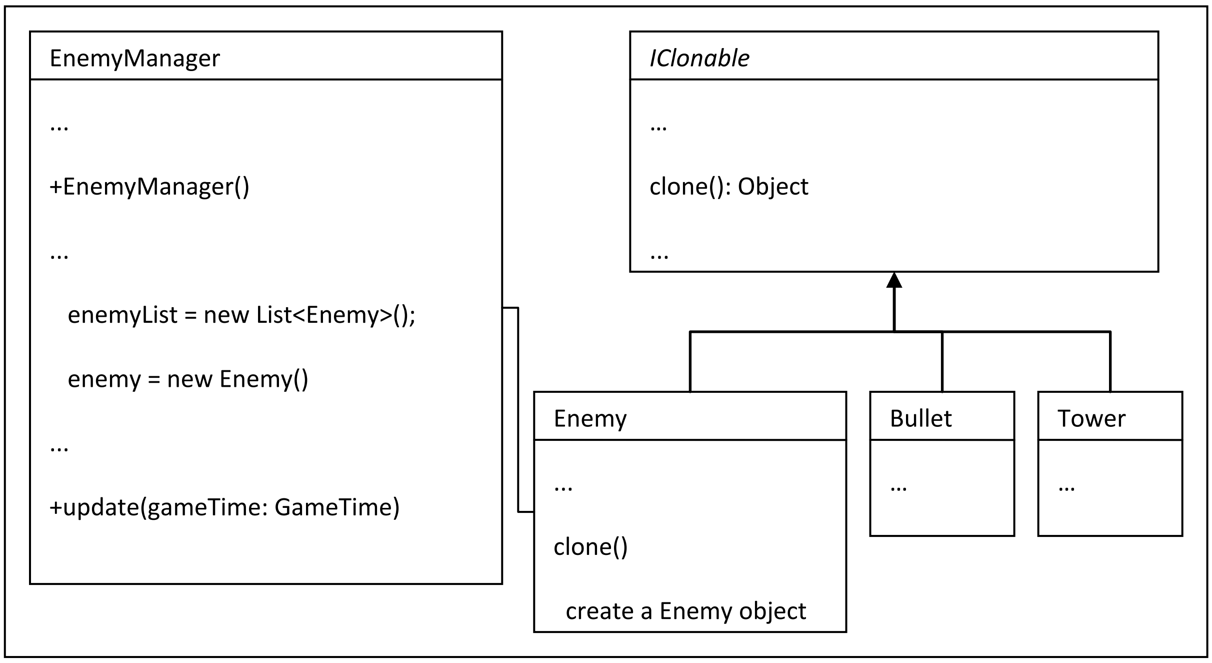


Figure 6.4: Template Method Pattern practice on Enemy object in Tower Defense Game

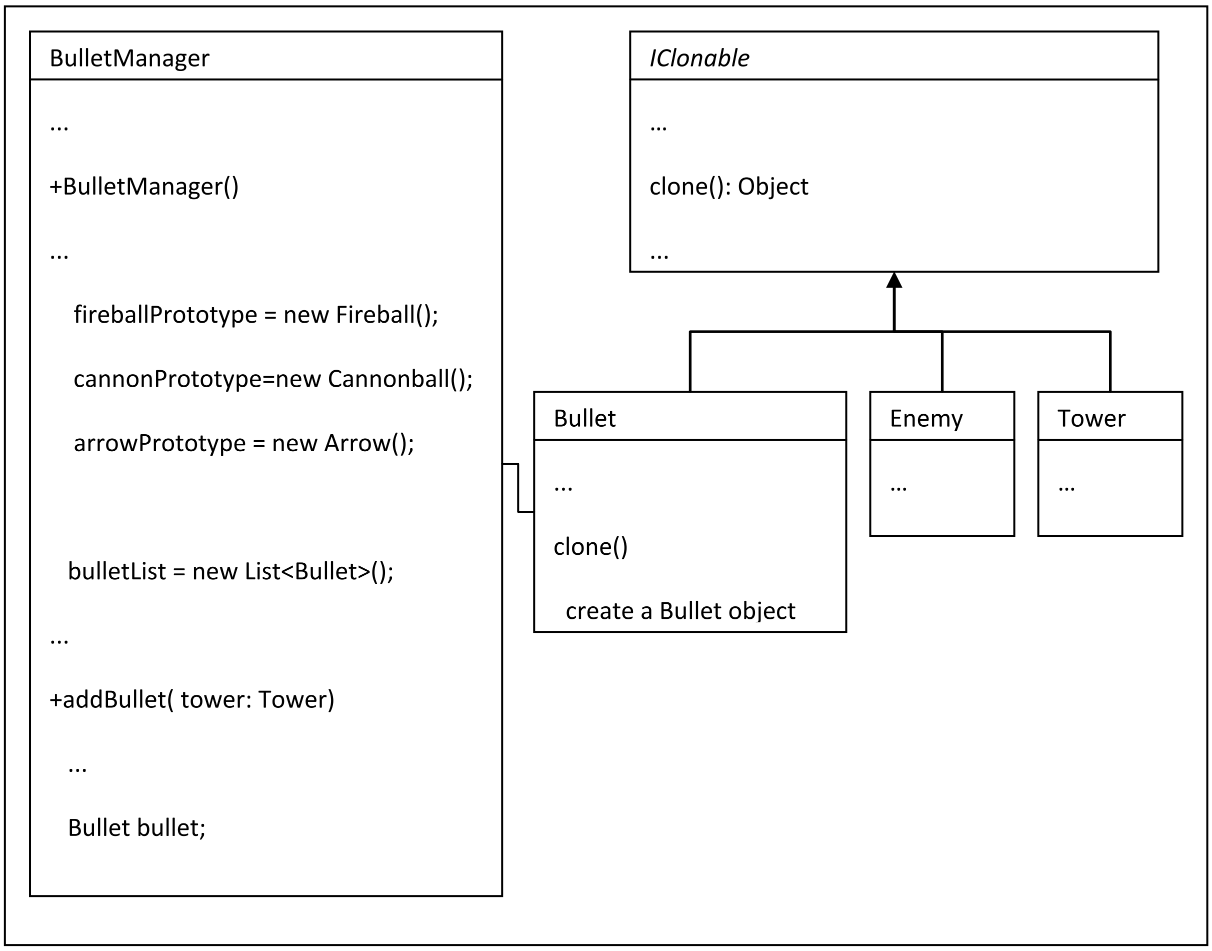
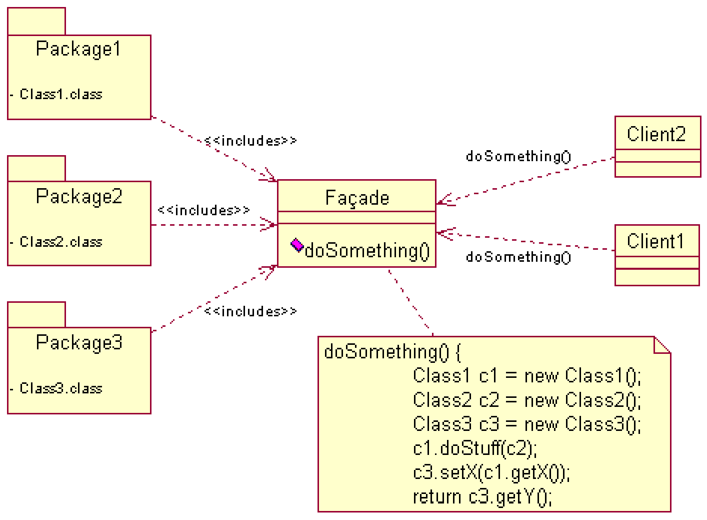

BulletManager
...
+BulletManager()
...
fireballPrototype = new Fireball();
cannonPrototype=new Cannonball();
arrowPrototype = new Arrow();
bulletList = new List<Bullet>();
...
+addBullet(tower: Tower)
...
Bullet bullet;
IClonable
...
clone(): Object
...
Bullet
...
clone()
create a Bullet object
Enemy
...
Tower
...

Figure 6.5: Template Method Pattern practice on Bullet object in Tower Defense Game

Façade Pattern



```
doSomething() {
        Class1 c1 = new Class1();
        Class2 c2 = new Class2();
        Class3 c3 = new Class3();
        c1.doStuff(c2);
        c3.setX(c1.getX());
        return c3.getY();
```

Façade Pattern is a structural design pattern. It provides a better interface - understandable and usable methods for the system- for a collection of class libraries or long and complex codes. It provides methods to the system that does what is expected by system, using the methods in the libraries and packages.

Figure 6.6: The UML representation of Façade Pattern

Drawer class in Tower Defense Gama is a Façade class for graphical classes in XNA package. It does simplify drawing operations that are supposed to be done with complex method provided by XNA. For example; *Drawer* class provides a *loadAllTexture2Dfiles().* It must be done by XNA's methods as follows:

```
public void loadAllTexture2DFiles()

{

     string[] filePaths = Directory.GetFiles(@"Content\\", "*.jpg");


                 for (int i = 0; i < filePaths.Length; i++)

                 addTexture2D(filePaths[i].Substring(9,
filePaths[i].Length - 9));
```

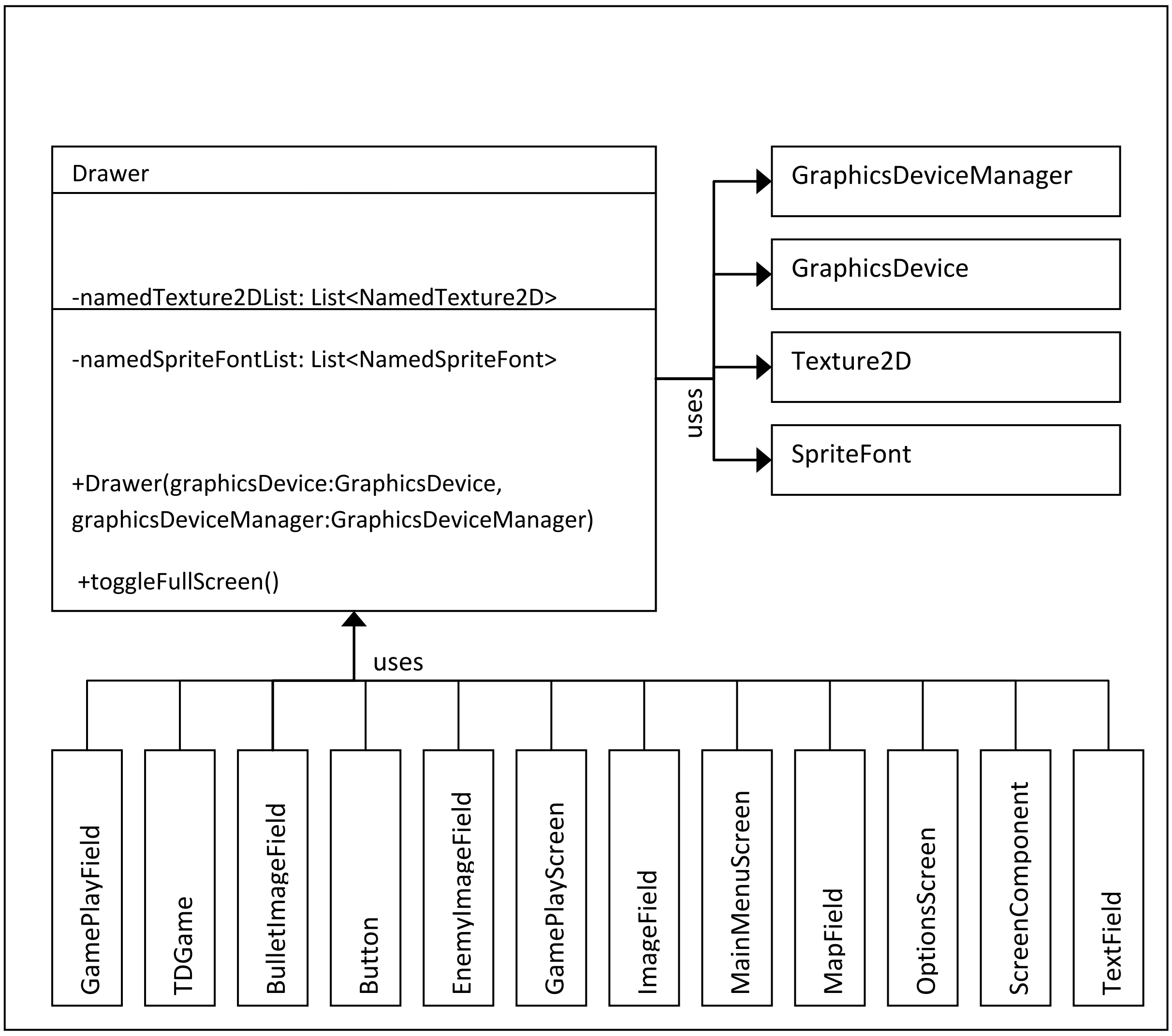


Figure 6.7: Façade Pattern practice on Drawer object in Tower Defense Game

6.2 Class Interfaces

Arrow
+Arrow(position: Vector2, enemy: Enemy, damage: int, bulletManager: BulletManager) + Clone():Arrow

<u>Arrow :</u>

Extends: Bullet

This class is used to create bullet object as arrow by using position, enemy, damage, bullet manager. It is derived from Bullet Class, and is cloneable.

Background
-backgroundType : BackgroundType

+Background(backgroundType: BackgroundType)
+setBackground(backgroundType: BackgroundType)
+ getBackgorundType():BackgroundType

<u>Background:</u>

This class is keeps the pre defined background values for created profiles.

Bullet
direction: Vector2 # targetPosition : Vector2 # startPosition : Vector2 # speed : float # damage : int # enemy : Enemy # bulletManager : BulletManager + bulletHit: FiredEvent

+ Bullet(position: Vector2, enemy: Enemy, damage: int, bulletManager: BulletManager)
+ FiredEvent(sender: Bullet)
+setDamage(damage: int)
+setEnemy(enemy: Enemy)
+update(gameTime: GameTime)
+Clone(): object

<u>Bullet</u>

Implements: ICloneable

Extends: GameObject

This class provides an abstraction to Bullet objects and updates damage power, enemy position and direction.

BulletManager
- bulletList: List<Bullet> - gameLogic: GameLogic - firingPositionOffset :Vector2 - fireballPrototype: Bullet - cannonballPrototype: Bullet - arrowPrototype: Bullet

+ BulletManager(gameLogic: GameLogic)
+addBullet(tower: Tower)
+removeBullet(bullet: Bullet)
+reset()
+update(gameTime: GameTime)
+getBulletList():List<Bullet>

BulletManager

Implements: Updateable

This class handles Bullet objects and keeps them in a list.

Cannonball
+Cannonball(position: Vector2, enemy: Enemy, damage: int, bulletManager: BulletManager) +Clone(): Cannonball

Cannonball

Extends: Bullet

This class is used to create bullet object subtype Cannonbal by using position, enemy, damage, bullet manager. It is derived from Bullet Class.

Cow
+ Cow(wayPoints: List<Vector2>, enemyManager: List<Vector2>) +Clone(): Cow

Cow

Extends: Enemy

This class is used to create Enemy object subtype Cow by using way points list and enemy manager. It is derived from Enemy Class.

DwarvenTower

+ DwarvenTower(position: Vector2, towerManager: TowerManager)
+Clone(): DwarvenTower
+calculateDamage()

DwarvenTower

Extends: Tower

This class is used to create tower object subtype DwarvenTower by using position, enemy, damage, bullet manager. It is derived from Tower Class. It creates DwarvenTower prototype object and subscribes the bullet to fire the event recursively, and is cloneable.

EladrinTower
+ EladrinTower (position: Vector2, towerManager: TowerManager) +Clone():EladrinTower +calculateDamage()

EladrinTower

Extends: Tower

This class is used to create tower subtype EladrinTower by using position, enemy, damage, bullet manager. It is derived from Tower Class. It creates EladrinTower prototype object and subscribes the bullet to fire the event recursively, and is cloneable.

Enemy
enemyState: EnemyState # hitPoint: int #maxHitPoint: int #currentPoint: int #speed: float #direction: Vector2 #targetPosition: Vector2 #startPosition;: Vector2 #wayPoints: List<Vector2> #enemyManager: EnemyManager #rotation: float +enemyKill: FiredEvent +enemySurvived: FiredEvent
+ Enemy(wayPoints: List<Vector2>, enemyManager: List<Vector2>) +FiredEvent(sender:object) -calculateRotation () +dealDamage(damage: int) +isInactive():boolean + isDead():boolean +update(gameTime: GameTime) +getRotation():float +getDirection():Vector2 +Clone():Enemy

Enemy

Implements: ICloneable

Extends: GameObject

This class abstracts all enemy objects. It holds all info about the enemy objects which are sheep and cow for now. It is clonable and enforces its subclasses to be clonable.

EnemyManager
- enemyList: List<Enemy> -currentLevel: Level - currentMap: Map -spawnTime: int -enemyCount: int -spawnController: int - sheepPrototype: Enemy -cowPrototype: Enemy

+EnemyManager(level: Level, map: Map)
+update(gameTime: GameTime)
+getSheepPrototype():Sheep
+getCowPrototype():Cow
+setCurrentLevel(level: Level)
+addEnemy(enemy: Enemy)
+removeEnemy(enemy: Enemy)
+getEnemyList():List<Enemy>
+reset()

Enemy

Implements: Updatable

This class is the controller class for the Enemy objects. It holds the all Enemies in a list. All other major controllers access Enemy objects through the instance of Enemy Manager.

Fireball
+ Bullet(position: Vector2, enemy: Enemy, damage: int, bulletManager: BulletManager) +Clone():Fireball

Fireball

Extends: Bullet

This class is used to create bullet object subtype Fireball by using position, enemy, damage, bullet manager. It is derived from Bullet Class.

GameLogic
- towerManager: TowerManager - bulletManager: BulletManager - enemyManager: EnemyManager - gameLogicState: GameLogicState - map: Map - levelList : List<Level> - tDGame: TDGame - enemiesKilled : int - startingGold : int - gold: int - currentLevel: int

+ GameLogic(tDGame: TDGame)
+ getTDGame():TDGame
+getNumberOfKills():int
+ getGold():int
+setGold(gold: int)
+activate()
+deActivate()
+setToBuildingMode()
+setToPassiveMode()
+update(gameTime: GameTime)
+ getTowerManager():TowerManager
+ getEnemyManager():EnemyManager
+ getBulletManager():BulletManager
+onGamePlayScreenClick(sender: object)
+onBulletFired(tower: Tower)
+onEnemyKill(sender: object)
+onEnemySurvived(sender: object)
+getLevelList():List<Level>
+getMap():Map

GameLogic

Implements: Updateable

This class creates state of game logic,map, tower and enemy managers,subcribes the events which fire the bullets and kill enemies by getting prototypes of the objects. Here, all managers are created and the objects of Tower,Enemy classes are added to managers. Game play screen components are activated or deactivated and   state of game logic is changed as active or building mode.Deactivation of components requires since all buttons woks even if game ends Further, Game logic updates states of and subscribes the events such as killing enemy, bullet firing and suviving enemy.

GameObject
#name:string #position:Vector2

+update(gameTime: GameTime)
+getPosition():Vector2
+setPosition(position:Vector2)

GameObject

Implements: Updateable

This class provides an abstraction to all GameObject subtypes and is Updateable. It is the parent of all models that are visible in the game.

GameScreenManager
-tDGame: TDGame -activeGameScreen: GameScreen -gameScreenList: List<GameScreen> -gameScreenState: GameScreenState

+GameScreenManager(tDGame: TDGame)
+addGameScreen(gameScreen: GameScreen)
+draw()
+changeGameScreen(gameScreenState:GameScreenState)

GameScreenManager

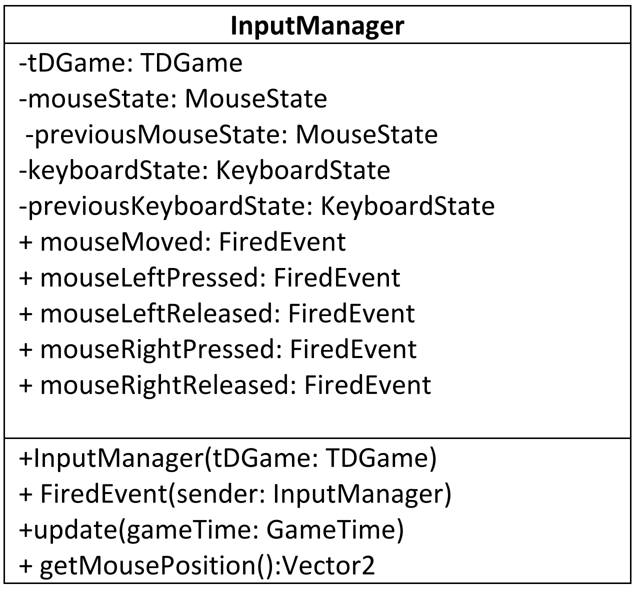
This class is the controller class of GameScreen objects and controls the state of the active screen. It also draws the active screen as its controlling unit, by calling related subroutines from the screen.

HumanTower
+ HumanTower (position: Vector2, towerManager: TowerManager) +Clone():EladrinTower +calculateDamage()

HumanTower

Extends: Tower

This class is used to create tower subtype human tower by using position, enemy, tower manager. It is derived from Tower Class. It creates the human tower object and subscribes the bullet to fire the event recursively.



InputManager

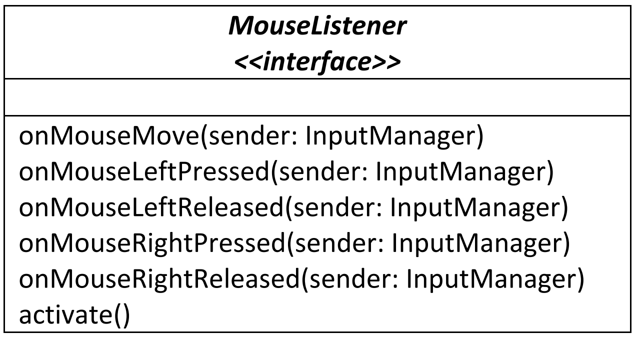
-tDGame: TDGame
-mouseState: MouseState
-previousMouseState: MouseState
-keyboardState: KeyboardState
-previousKeyboardState: KeyboardState
+ mouseMoved: FiredEvent
+ mouseLeftPressed: FiredEvent
+ mouseLeftReleased: FiredEvent
+ mouseRightPressed: FiredEvent
+ mouseRightReleased: FiredEvent

+InputManager(tDGame: TDGame)
+ FiredEvent(sender: InputManager)
+update(gameTime: GameTime)
+ getMousePosition():Vector2

<u>InputManager</u>

Implements: Updatable

This class handles user inputs like mouse and keyboard actions. Internally, it stores input states (mouse position, pressed keys etc.)

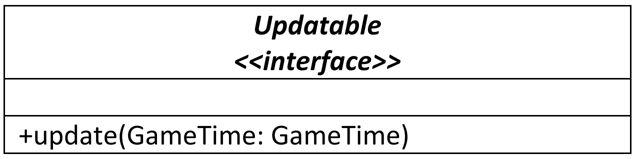


MouseListener
<<interface>>

onMouseMove(sender: InputManager)
onMouseLeftPressed(sender: InputManager)
onMouseLeftReleased(sender: InputManager)
onMouseRightPressed(sender: InputManager)
onMouseRightReleased(sender: InputManager)
activate()

<u>MouseListener</u>

This interface handles MouseEvents. Any class that implements this interface should take appropriate actions according to listed events in the interface.



Updatable
<<interface>>

+update(GameTime: GameTime)

<u>Updatable</u>

This interface makes sure that certain objects implemented update method by passing the GameTime object provided by XNA’s native libraries.

Level
+Level(levelNo: int, spawnTime: int, enemyCount: int, enemyType: int) + getEnemyCount(): int +getSpawnTime(): int +getLevelNo(): int +getEnemyType(): int

Level :

This class is used to enemy count which is valid for the existent level, the frequency of creation of the enemy in the determined spawn time, the existence of level and type of enemy is valid for existent level by using levelNo, spawnTime, enemyType and enemyCount variables.

LevelImporter
+ importLevel(): List<Level> + exportLevel(levelList: List<Level>)

LevelImporter :

This class is used to import levels from Levels.htk file and export levels to Levels.htk file.

Map
+Map() +Map(name: String, textureName: String, map: int[][]) + getWayPoints(): List<Vector2> +getMap(): int[][] +getMapSize(): int +getTextureName(): String +getName(): String

Map:

This class is used to represent maps in the game. It stores maps as a 2D int array in which numbers represent different types of roads. It calculates waypoints, which is needed by enemy objects.

MapImporter
+ importMap(): List<Map> + exportMapl(mapList: List<Map>)

MapImporter :

This class is used to import maps from Maps.htk file and export maps to Maps.htk file.

Profile

+Profile()
+getTrait(): Trait
+ setTrait(trait: Trait)
+getBackground(): Background
+setBackground(background: Background):

Profile:

Profile class stores profile information for a player. It contains Background and Trait classes for specialized profiles.

Sheep
+Sheep(waypoints: List<Vector2>, enemyManager: EnemyManager) +Clone(): Sheep

Sheep:

Extends: Enemy

Represent Sheep object in the game.

SoundManager

+SoundManager(tDGame: TDGame)
+register():
+playCannonSound(sender: object)
+playArrowSound(sender: object)
+playFireballSound(sender: object)
+update(gameTime: GameTime)

SoundManager:

Implements: Updateable

SoundManager is responsible for playing sound in the game. It subscribes to events that needs a sound effect and plays sound when event is fired.

TDGame

+TDGame()
#Initialize()
#LoadContent()
#UnloadContent()
#Update(gameTime: GameTime)
#Draw(gameTime: GameTime)
+getGameLogic(): GameLogic
+getGamePlayScreen(): GamePlayScreen

<u>TDGame:</u>

Extends: Game

This class is the general game class of the system that calls all related draw, update and load content methods firstly. Hierarchically on top of all, this class provides the main connection between the XNA Framework and the system, and maintains all initializations.

Tower
+getDamageType(): DamageType +getDamage(): int +getCost(): int +calculateDamage() +checkRange(enemyList: <List>Enemy) +getEnemy(): Enemy +Clone(): object +update(gameTime: GameTime)

<u>Tower:</u>

Implements: Icloneable,

Extends: GameObject

It is an abstract class that have methods to determine the damage type, calculate amount of damage power, try to determine any enemy is found on the view of the Tower or not, subcribe firing bullet event by depending on range of tower object. This class is cloneable.

Tower Manager

+TowerManager(map: Map, gameLogic: GameLogic)
+getDwarvenTowerPrototype(): DwarvenTower
+getHumanTowerPrototype(): HumanTower
+getEladrinTowerPrototype(): EladrinTower
+addTower(towerType: int, position: Vector2, gold: int)
+removeTower(tower: Tower)
+reset()
+update(gameTime: GameTime)
+getTowerList(): List<Tower>
+getGameLogic(): GameLogic

Tower Manager:

Implements: Updateable

This class is the controller class of Tower objects that are created and edited on runtime, according to prototype design pattern. Tower objects are controlled on the game map, thus tower manager should have an initial value as the map. Towers are created and edited as they are requested by the game logic, and tower manager keeps the track of number of enemies. It controls the existence of towers on a specific map, allows their controlled modification, and creates appropriate clones from specified pattern towers. This class is updatable, and while it updates on GameTime, it stores and process the list of towers and their state.

Trait

+Trait(traitType: TraitType)
+setTrait(traitType: TraitType)
+getTraitType(): TraitType

Trait:

This class is keeps the pre defined trait values for created profiles.

BulletImageField

+ BulletImagField(textureName: String, drawer: Drawer, bulletList: List<Bullet>)
+ draw(drawer: Drawer)
+setTexture(textureName: String)

<u>BulletImageField</u>

This class is the screen component derivation for Bullet models, which sets and draws the texture of bullets in the game. Member of view model, it is used to abstract the bullet drawings.

Button

+ Button(text: String, textureName: String, drawer:Drawer)
+ setText(text: String)
+getText(): text
+activate()
+deActivcate()
+setPosition(position: Vector2)
+setScale(scale: Vector2)
+recalcuateTextPosition()
+draw(drawer: Drawer)
+disabeText()
+enableText()
+onMouseMove(sender:InputManager)
+onMouseLeftPressed(sender:InputManager)
+onMouseLeftReleased(sender: InputManager)
+onMouseRightPressed(sender: InputManager)
+onMouseRightReleased(sender: InputManager)

<u>Button</u>

This class represents the user interaction buttons, and allows them to be drawn, updated and to notify events. Button derives from ScreenComponent, and notifications are allowed by MouseListener interface.

Drawable
+ draw(drawer: Drawer)

Drawable

This interface enforces drawable classes to have a draw method that takes Drawer class as the parameter

Drawer

+ Drawer(graphicsDevice: GraphicsDevice, graphicsDeviceManager: GraphicsDeviceManager)
+toggleFullScreen()
+loadAllTexture2DFiles()
+addTexture2D(name: String, texture2D Texture2D)
+addTexture2D(fileName: String)
+addSpriteFont(name: String, spriteFont: SpriteFont)
+getSpriteFont(name: String): SpriteFont
+getTexture2D(name: String): Texture2D

Drawer

This class is derived from default XNA SpriteBatch class, which is used to draw sprites on screen. In this derived method, as well as using other SpriteBatch methods, user can use pre-load content and call them by name. Also, other game specific methods are supported as well.

EnemyImageField

+ EnemyImageField(textureName: String, drawer: Drawer, enemyList: List<Enemy>)
+draw(drawer: Drawer)

EnemyImageField

This class is the screen component derivation for Enemy models, which sets and draws the texture of enemies in the game. Member of view model, it is used to abstract the enemy drawings.

GamePlayField

+ GamePlayFeld(textureName: String, drawer: Drawer)
+activate()
+deActivate()
+onMouseMove(sender: InputManager)
+onMouseLeftPressed(sender: InputManager)
+onMouseLeftReleased(sender: InputManager)
+onMouseRightPressed(sender: InputManager)
+onMouseRightReleased(sender: InputManager)
+draw(drawer: Drawer)

<u>GamePlayField</u>

This class is the subtype of ScreenComponent and implements MoseListener interface. It activates or deactivates of the button to subscribe or desubscribe the events such as moving the Mouse pressing left button of the mouse or releasing left button of the Mouse.

GamePlayField

+ GamePlayScreen(tDGame: TDGame, drawer: Drawer)
+draw()
+activate()
+deActivate()
+onButtonClick(sender: Button)
+onMouseMove(sender: InputManager)
+getGamePlayField(): GamePlayField
+getSelectedTowerType(): int
+setSelectedTowerType(selectedTowerType: int)

<u>GamePlayScreen</u>

This class derives from GameScreen and is the class which represents a game screen that is the active game playing. It forms a user interface and a view model, and allows event subscribers to be notified by specific events generated while user interaction.

GameScreen

+ addScreenComponent(screenComponent: ScreenComponent)
+getTDGame(): TDGame
+getGameScreenManager(): GameScreenManager
+setGameScreenManager(gameScreenManager: GameScreenManager)
+draw()
+activate()
+deactivate()

<u>GameScreen</u>

This class is the abstract interface for all game screens, which represent the view model and used to visualize states of the game. It allows general controller event notifier and drawer methods. Events can be activated, and deactivated.

ImageField

+ ImageField(textureName: String, drawer: Drawer)
+draw(drawer: Drawer)
+setTexture(textureName: String)

<u>ImageField</u>

This class is a derived screen component, which forms an image to be drawn on a game screen.

MainMenuScreen

+ MainMenuScreen(tDGame: TDGame, drawer: Drawer)
+onButtonClick(sender:Button)
+activate()

<u>MainMenuScreen</u>

This class derives from GameScreen and is the class which represents a game screen that is the active game playing. It forms a user interface and a view model, and allows event subscribers to be notified by specific events generated while user interaction.

MapField

+ MapField(textureName: String, drawer: Drawer, map: Map)
+draw(drawer: Drawer)

MapField

This class is the screen component derivation for the map model, which sets and draws the texture of map components in the game. Member of view model, it is used to abstract the map drawings.

OptionsScreen

+ OptionsScreen(tDGame: TDGame, drawer: Drawer)
+onButtonClick(sender:Button)
+activate()
+deactivate()

OptionsScreen

This class derives from GameScreen and is the class which represents a game screen that is the active game playing. It forms a user interface and a view model, and allows event subscribers to be notified by specific events generated while user interaction. Events can be activated, and deactivated.

ScreenComponent

+ setLayerIndex(layerIndex: float)
+getLayerIndex(): float
+setScale(scale: Vector2)
+setPosition(position: Vector2)
+setTexture(textureName: String)
+draw(drawer: Drawer)

ScreenComponent

This class is the abstract class for all screen components, and provides general default methods and attributes for them. It uses the Drawable interface, and is part of View model.

TextField

+ TextField(textField: String, drawer: Drawer)
+draw(drawer: Drawer)
+setText(text:String)

<u>TextField</u>

This class is the screen component derivation for the Text model, which sets and draws textual components in the game. Member of view model, it is used to abstract any textual visual.

6.3 Specifying Contracts using OCL

Tower Manager:

Context TowerManager.update(GameTime gameTime)

Pre:

(gameLogic.gameLogicState = GameLogicState.activeMode || gameLogic.gameLogicState = GameLogicState.buildingMode)

To update TowerManager, associated gameLogic must be either in active mode or building mode. If so, TowerManager is need to be updated in case building new tower or selling a tower etc.

Context TowerManager.addTower(int towerType, Vector2 position, int gold)

Post:

(gameLogic.getGold() < gameLogic.getGold()@pre)

(gameLogic.getGold() > 0)

After tower is built, particular amount of money should be reduced from the system. And resulting gold amount must be higher than zero.

Context TowerManager.reset()

Pre:

(gameLogic.gameLogicState = GameLogicState.buildingMode ||

gameLogic.gameLogicState = GameLogicState.passiveMode)

Post:

(self.towerList.size() = 0)

In order to reset a tower manager, gamelogic should be deactivated or game must be in a building mode (where there is no enemies running). After reset, all towers will be removed, towerList's size will become zero.

Enemy Manager:

Context EnemyManager.update(GameTime gameTime)

Pre:

(gameLogic.gameLogicState = GameLogicState.activeMode)

To update EnemyManager, associated gameLogic must be in activeMode since enemies will only be active and need to be updated when a gameLogic is in activeMode.

Context EnemyManager.addEnemy(Enemy enemy)

Pre:

(spawnController > spawnTime && enemyCount > 0)

Post:

(spawnController = 0 && enemyCount + 1 = enemyCount@pre)

To add a new enemy, enemyCount must be higher than zero. EnemyCount indicates how many enemies will be spawned in each level. Therefore there will be no spawn when enemyCount is zero.

When new enemy is added, enemyCount must be decremented by one.

Context EnemyManager.setCurrentLevel(Level leve)

Pre:

(gameLogic.gameLogicState = GameLogicState.buildingMode)

setCurrentLevel(Level level)

Post:

(self.currentLevel = level)

(self.spawnTime = self.currentLevel.getSpawnTime())

(self.enemyCount = self.currentLevel.getEnemyCount())

Setting a new level only occurs when gameLogic is in buildingMode. After new level is set currentLevel must be equal to level, spawnTime must be equal to currentLevel's spawnTime property and enemyCount must be equal to currentLevel's enemyCount property.

Context EnemyManager.reset()

Pre:

(gameLogic.gameLogicState = GameLogicState.passiveMode)

Post:

(self.enemyList.size() = 0)

When game is in passive mode, enemyManager can be reset. Therefore resulting in an enemyList with size zero.

Bullet Manager:

Context BulletManager.update(GameTime gameTime)

Pre:

(gameLogic.gameLogicState = GameLogicState.activeMode)

BulletManager needs to be updated only when gameLogic is in activeMode.

Context BulletManager.remove(Bullet bullet)

Pre:

((bullet.position - bullet.startPosition).Length() >

(bullet.targetPosition - bullet.startPosition).Length())

BulletManager needs to remove a bullet when a particular bullet reaches its target.

Context BulletManager.reset()

Pre:

(gameLogic.gameLogicState = GameLogicState.passiveMode)

Post:

(self.bulletList.size() = 0)

Bullet manager can be reset only when game is in passive mode. After reset all bullets will be removed, resulting in a bulletList of size zero.

Game Logic:

Context GameLogic.onEnemySurvived(object sender)

Post:

(enemiesKilled++)

(gold < gold@pre)

After a survival of an enemy, it is regarded as killed*, and since player missed that enemy particular amount of gold reduction has to be made.

Context GameLogic.onEnemyKill(object sender)

Post:

(enemiesKilled++)

(gold > gold@pre)

After a death of an enemy, it is regarded as killed, and since player killed that enemy particular amount of gold addition has to be made.

Context GameLogic.setToBuildingMode()

Post:

(self.gameLogicState = GameLogicState.buildingMode)

When a gameLogic's setToBuildingMode() operation is called, at the end, gameLogic must be in buildingMode.

Context GameLogic.setToActiveMode()

Post:

(self.gameLogicState = GameLogicState.activeMode)

GameLogic is set to activeMode when this operation is performed.

Context GameLogic.setToPassiveMode()

Post:

(currentLevel = 0)

(enemiesKilled = 0)

(gameLogicState = GameLogicState.passive)

When setToPassiveMode operation is called, currentLevel and enemiesKilled values are revert back to zero while gameLogic State changes to passive mode.

Context GameLogic.onGamePlayScreenClick(object sender)

Post:

(getGamePlayScreen().getSelectedTowerType() = -1)

When user clicks on gameplay area he builts tower if he selected one. And his building selection is set to -1 which means no particular tower. In order to build again, user needs to click tower buttons again.

Context GameLogic.activate()

Pre:

(sender.getText() = "Start New Game")

This operations precondition is satisfied when a sender object (Button in this case)'s text is "Start New Game" therefore activates gameLogic to initialize gameplay.

Context GameLogic.deActivate()

Pre:

(sender.getText() = "Back")

This operations precondition is satisfied when a sender object (Button in this case)'s text is "Back" therefore deactivates gameLogic.

Tower:

Context Tower.update(GameTime gameTime)

Post:

(lastAttacked – 1 = lastAttacked@pre)

After each update of a tower, its lastAttacked value must be incremented. lastAttacked value indicates when tower object fired a bullet.

Context Tower.bulletFired(Tower tower)

Pre:

(lastAttacked > attackRate)

(enemy != null)

A bullet is fired when tower has waited enough time (attackRate specifies how long tower waits until each firing) and it has an associated enemy in range.

Enemy:

Context Enemy.dealDamage(int damage)

Pre:

((position - startPosition).Length() >

(targetPosition - startPosition).Length())

Post:

self.hitPoint = self.hitPoint@pre – damage

When bullets positions go beyond its target point, it means, that bullet has hit the enemy. Therefore, particular amount of damage needs to be reduced from enemy.

Context Enemy.enemyKill(Enemy enemy)

Pre:

(hitPoint <= 0)

Post:

self.enemyState = EnemyState.dead

EnemyKill operation is called when enemies hit point is below zero. As a post condition, its state must be dead.

Context Enemy.enemySurvived(Enemy enemy)

Pre:

(self.enemyState = EnemyState.finished)

EnemySurvived operation is called when enemy state is finished. It means enemy has reached the final waypoint without being killed.

Bullet:

Context Bullet.update(GameTime gameTime)

Post:

(position = position@pre + direction * speed)

Each update call to a bullet adds velocity vector to its position vector representing bullet movement.

Context Bullet.setEnemy(Enemy enemy)

Post:

(self.enemy = enemy)

(self.startPosition = position)

(self.targetPosition = self.enemy.getPosition())

(direction = targetPosition – position)

When a bullet is associated with an enemy, associated enemy attribute is set to an enemy given by a parameter, its start position, target position and direction is initialized.

Context Bullet.bulletHit(Bullet bullet)

Pre:

((position - startPosition).Length() >

(targetPosition - startPosition).Length())

When bullets positions go beyond its target point, it means, that bullet has hit the enemy. Therefore this method is called when a condition above is satisfied.

Arrow:

Context Arrow

inv:

(speed = 40)

All instances of an Arrow should have same speed which is 40. (Means 40 pixel / sn, if these numbers are to be changed, arrows may not be seen properly)

DwarvenTower:

Context DwarvenTower

inv:

(attackRate = 150)

(baseCost = 170)

(range = 140)

(damageType = DamageType.cannon)

All instances of an Dwarven should have same values as specified above. These values will never change in a game.

Sheep:

Context Sheep

inv:

(speed = 1)

All instances of an Sheep should have same speed. Change of this value may result in unexpected results like going beyond waypoints in game screen etc.

Screen Component:

Context ScreenComponent

inv:

(layerIndex <= 1 && layerIndex > 0)

Layer index specifies drawing layer for any screen component. Components with higher layer indexes are drawn later than others. It should be a value between 0 and 1 otherwise it will cause an error.

7. Conclusion

The project is conducted with a great risk from the beginning: it aimed to improve a very-well known and admired genre of game, Tower Defense Game. The project had the risk of repeating what's already done and being discarded as a copycat. To achieve that, project team mostly focused on graphical elements and characteristics of game object to create a bond between the game and the gamer. The previous examples of the genre had been generally developed for web browsers and for this reason they lacked the graphical advantage and performance. The project team has chosen the platform as PCs for a better performance and improves graphics. Besides that, the design of towers, enemies and surroundings are studied in detail to catch an atmosphere that the gamer would like to return again. The algorithm of the game logic was hard to design since it must be a balance between hardcore and casual games. In this point, existing games are studied and their mechanisms are combined in a way that developer team believed that it is fun to play Tower Defense Game. Although it was a Greenfield engineering practice, many games had set some conventions and those must be followed without undergoing a total cloning operation.

The project was a team work practice but after 2 years of engineering education, it was a better application of work sharing, reporting, synchronization and communication. Because team formation was done with the realization of previously encountered problems –like irresponsibility against other members, missing meetings, handing incomplete work- the project had run problem-free in terms of team members.

The design process was an unfamiliar one, even though it was briefly introduced in other courses (like CS101-102), it was the first time practicing a strictly regulated design process for each team member. The programming experience was almost always about algorithms or learning key concepts. There was only one project as detailed as this one, but it was again about learning and practicing the theoretical knowledge (CS101-102). Because of these reasons, conducting a project from the scratch and following a development cycle were new experiences. The biggest challenge, in these terms, was to understand the concepts and applying them in implementation simultaneously, without any prior experience. It would be more convenient to exemplify the key concepts with real life project (coded and run).

Documentation and reports were useful but very difficult to achieve in deadlines. The verbal explanations and the necessity to design a software system on the paper without implementation were new concepts and also very compelling. Our conditioning in iterative design methods (implement some, test; then implement more.) was to be removed.

Some concepts learned in the class are proved to be useful and realistic. Especially, after starting coding, when we learned about object design patterns; we were surprised by the fact that we had already used the Prototype Pattern before naming it. It was a justification for the definition of the pattern; it was actually a generic solution to a set of problems that is encountered many times. Also, the architectural patterns were a perfect match for the system we designed. We had to configure our prior designs to make them fit into some architectural patterns (like separating managers from the game logic to have model and controller separately); however, it proven to be a better solution for event driven systems like a fully interactive game.

The project is delivered by a due date and the final product was incomplete, considering the system described at the beginning. First of all, system can be slightly modified (adding gamepad functionality instead of mouse) to run the system on Xbox game console. This was a plus when C# was chosen as the implementation language. Actually, the project was designed to be implemented in C# from the very first moment, but this is irrelevant because all the development cycle was independent of programming language choice. The system could be implemented in any other Object Oriented language by using the documentation provided until the implementation stage. Other future development might be the expansion of contents in the game. This is also very easy with the system developed. Any team that inherits the project can follow the reports and with a quick revision of code, they can add new contents to the game. Level count can be incremented as well as addition of enemy and towers.

The project is left incomplete in some parts, like the graphical interfaces of profile and map editor menus. Even though the infrastructure for map editor is ready, due to deadline issues, graphical interface (view part of that subsystem) is left to be finished later. By the same reasons, profile functionality is left unfinished. Because the project was done purely educational reasons, we believe these can be accomplished with a little more time.

After all, the project was a hard, challenging yet very educative experience. We learned project development cycle, individual processes and how it can be done with a team working together, the importance of documentation (both during the project and after the project), applying patterns to reach solutions easier, trading some design goals for a greater goal (performance for cost, portability for performance) and applying engineering principles from the beginning to the end. But, most importantly, we learnt communication between client and developer is crucial as well as that of among developers.

8. References

- Bruegge, Bernd. Allen H. Dutoit. Object Oriented Software Engineering: Using UML, Patterns, and Java. Pearson Prentice Hall. NJ. 2004. *A canon textbook of object oriented software engineering.*
- http://www.freewebarcade.com/tower-defense-games.php *A portal of free web based tower defense games*
- http://creators.xna.com/en-US/ *Offical Site of XNA Community*
- www.gamedev.net *The largest game development online community, both for beginners and seasoned programmers.*
- http://www.novelconcepts.co.uk/FlashElementTD/ *"The Flash Element", a tower defense game which influenced game design significantly*
- http://msdn.microsoft.com/en-us/library/bb198548.aspx *Official XNA Game Studio 3.1 Programming Manual by Microsoft*
- http://java.sun.com/blueprints/patterns/MVC.html *Application of Model View Controller pattern in Java*
- http://hillside.net/patterns/books/Siemens/abstracts.html *A discussion of frequently used architectural patterns in fostware design, by Siemens*

Appendices

A. Game Content

Backgrounds:

Dwarf Master Engineer:

Stout and stodgy, dwarves have an extremely solid appearance. Their small bodies are tightly packed with muscles and strong ligaments, and their hardy constitution sees them through some extremely harsh climates. To the outside world, dwarves are often seen as dour, even sour, people. They are extremely community oriented, and usually live in city fortresses dug inside mountains. They are extremely community oriented, and can be considered xenophobic by many. Most of the dwarf communities are located in Karrnath, and the biggest dwarven community known is the Sultanate of Mroar.

Dwarf cities are separated by clans, and they usually are defensive militaristic fortress cities. Although defensive, dwarves take pride in what they have and their greed to achieve glory and wealth. They are strong and legendary warriors, and gained numerous major victories against their enemies, eradicated many races, declared a somewhat successful jihad against dragons, and enslaved Gnomes, their long lasting allies (although many would argue that this enslavement was voluntarily). However, things are not that good for the dwarves, as their ambition and greed opened up the gates of abyss deep in their mines and full might of dragons decide to take the revenge of their fallen brethren. These drastic events forced dwarves to take an unexpected step: they were (somewhat) unified under a sultanate and began to retaliate to these dark creatures to defend their honor.

Culturally industrialized, dwarves take pride in crafting and smithing, and they excel at the mining of precious metals and stones. With the help of their gnome slaves, a constant rigorous desire to achieve perfectness by hardworking allowed them to improve their technologies beyond any measure. Combined with an ancestral militaristic mind, dwarves have created many technological weapons, weapons which other races may only see in their dreams, or in nightmares.

If player decides to create a dwarven character, s/he plays a seasoned combat engineer hailing from Clan Chromehammer of Mroar Holds, one of the best that Engineering Society of Mithral Corps can provide. Promoted to your new commanding area by the imperial order of your fierce and ambitious sultan, your character gathers his best soldiers and workers to do what he does best.

Towers:

Riflemen Sentry: The first line of defense, riflemen is the basic infantry units in dwarven armies. These sentry towers are built on outer borders of the empire, and allow a basic yet capable defense. Effect: Standard attack range, Standard attack speed, slightly increased damage, slightly more expansive. Type: Hit.

Abus Gun Tower: Riflemen who prove their honor in the battlefield are promoted a higher rank, allowed to have extra knots on their beards, and carry an ancestral abus gun. Abus guns resemble hand cannons, however they actually are rifles inherited from grand-grand-grandfather of a soldier. Of course in time, they had been modified with every possible thing that family had put their hands on it. Effect: Increased attack range, low attack speed, increased damage, more expansive. Type: Hit.

Dragon Tower: The holy jihad against dragons allowed dwarven researchers and engineers to conduct further experiments on the dragon anatomy and morphology, which allowed many breakthroughs in dwarven technology, and arsenal. To celebrate these results, Ministry of Experimental Science invested on the creation of Dragon Towers, monuments to celebrate the proud victories of dwarves; and to experiment the usage of learned technique of dragon breathe containment. Effect: Very short attack range, highly increased attack speed, highly increased damage. Type: Flame.

Warden of Honor: Since the legendary times, champions of Adamantium Order have always been the honor guards of historical and valuable values of dwarven folk. With the introduction of state-of-art cannons by the Engineering Society of Mithral Corps, these guardians are even more legendary, and dangerous. Effect: Slightly increased attack range, slow attack speed, highly increased damage. Type: Hit, Splash.

Sultan's Hammer: In the capital, imperial orders are written on calligraphic runes and put on to the hammer of the statue of sultan himself. To encourage the soldiers on the field, these statues have been copied and distributed to defensive formations. Some claim that lightning bolts striking from the hammer is the full anger of the sultan, others point out the sign of Nickel-Eye Tesla, the Minister of Experimental Science, carved right under the hammer. Effect: Standard attack range, fast attack speed, increased damage. Type: Hit, Lightning.

Import Gnomish Fireworks ZX-11: Gnomes may not know how to behave properly, study properly, even follow the common sense properly, however, they sure know how to blow things up. A common entertainment product for Gnome weddings and special occasions, all stocks and further productions of this device have been bought by the sultan, leaving his gnome subjects more than puzzled. Effect: Highly increased attack range, standard attack speed, highly increased damage. Type: Splash, Fire.

Traits:

Masters Degree of Civil Engineering and Structure Analysis: Being in love with stones and believing that a strong foundation is the most fundamental aspect of a character, you have completed your masters degree. Effect: Tower building costs less.

Senior Engineering of Arms Smiths and Gunnery Techniques: Years spent in workshops and laboratories have thought you many endless tricks of trades. However, contrary to popular beliefs, you do know that size matters. Effect: Tower attack damages are increased.

Philosophy Doctorate of Gnome Technologies and Thinking Methodology: Being raised as the only son of wealthy dwarven family, you had everything, including a house full of gnome servants and nannies. It was no wonder why you have chosen such an academic interest, but your parents' refusal of you is still a mystery to you. Effect: Tower upgrades cost less.

Human Aristocrat:

Humans come in a wide variety of heights, weights and color, and their attire varies wildly, depending of the environment and society in which they live. They are decisive and often rash. They consist the majority of civilized races, endlessly ambitious, and greedy, and more importantly, aggressive and energetic enough to grasp all their desires. Their curse, however, is their life span is very short, compared to other races. Their lack of time forces them to decide and act fast, making them one of the most unexpected race in the world. Their desires are only matched by their capabilities, they are intelligent and analytic, have a pragmatic mind to adapt and change its environment.

Human societies show no sign of singularity and highly heterogeneous. Humans tend to create big cities, but many of them live in small communities as well. This young race is the most active civilized race, both in political arena and in terms of economics. Humans tend to be pious, and find comfort in pantheons since death is always around the corner. Some however, is ready to give everything he has to beat the death.

If player decides to create a human character, s/he plays a human aristocrat landlord. As an enlightened intellectual gentleman, your curiosity about the secrets of the dark art opened the gates of a new life to you, a life where souls are immortal, enemies wither and die with black curses, and the sweat road to power is achieved only by making enough sacrifices from your soul, and opening up your mind to a new world. Your everlasting dominion, however, is disturbed by constant the raids of monsters and an aristocrat is nothing without its land, immortal or not. Raising enough mercenaries and peasants, and preparing the book of rituals, you have no desire to leave your dark dominion.

Towers:

Arrow Tower: Peasant archers trained in the fields may not be as glorious as knights in shining armors, but an arrow sent from a tower can still pierce their armor. Effect: Standard attack range, standard attacking speed, slightly less attacking damage, cheap. Type: Hit.

Ballista Tower: Most consider mercenary soldiers untrustworthy, however having bloodthirsty sell-swords on your side is much better than not having them on your side. Especially, if they know how to build war machines. Effect: High attack range, slightly slow attacking speed, slightly high attacking damage. Type: Hit, line.

Catapult Tower: When mercenary captain suggested that he can smuggle some catapults, you did not know what to do with a siege engine on a defensive fight. When mercenaries were installing the catapult on the top of the towers, you were uncomfortable with the image. However, you ceased all your protests when you see peasants are happily using the machine; to smash the bones of enemy creatures brutally. Effect: Standard attack range, slow attacking speed, high attacking damage. Type: Hit, splash damage.

Haunted Estate: There were many rumors about the sudden and agonized death of opposing families to your rule. Some even claimed you as being a monster to force their spirits to stay on the world of living. They, however, learn that every human soul, especially theirs, is corrupted when they are much rejoicing to see those bounded spirits violently kills raiding parties and invasion forces, and leach their blood, and paralyze them, and make them screen in agony. Effect: Decreased attack range, standard attacking speed, standard attacking damage. Type: Hit, curse, slow.

Bloodstone Pillar: Requested crystals from the mines of Moria did not suspect anyone, it was only a rumor that these crimson stones were used to store souls in it. Effect: Standard attack range, very fast attacking speed, low attacking damage. Type: curse, all in the range.

Dark Keep: Folks in other northern lands were happy to see that shackled skeletons and sarcophaguses of dark wizards were sent to other places. Your people, however, we not that happy when they learned that it is you which orders to build keeps upon those darkest souls of all times. Effect: Increased attack range, standard attacking speed, highly increased attacking damage. Type: Hit, curse.

Traits:

Crimson King: You are the son of the night, drinker of blood, master of shadows and gloomy mistresses. Your trade of soul and blood makes it much easier to replenish your life. Effect: Purchase of extra lives is cheaper.

Heir Apparent: Although the fact that your father is the king does not make you any less evil or your peasants happier; it makes officials in the capitol to be more understandable to your special needs, even if they are often bizarre and plain weird. Effect: Gains “letter of requisitions” more frequently.

Entrepreneur: Just when everyone get used to your obsession of studying on corpses, putting them on market, and actually selling them, is not really helping your image, as much as it helps your coffers. Effect: Gains more gold from enemy kills.

Eladrin Archmage:

Creatures of magic with strong ties to other dimensions, eladrin are not native to the world of Sies, they live in the twilight realm of Faerinaal. Their spiral towers cities lie close enough to the world that they sometimes cross over, appearing briefly in mountain valleys or deep forest glades before fading back into the Faerinaal.

Eladrin are of human height, they are slim and even the strongest simply look athletic rather than muscle-bound. They have the same range of complexions as humans, though they are more often fair than dark. Their straight fine hair is often white, silver, or pale gold, and they wear it long and loose.

Eladrins are long lived creatures that are detached from this world. In truth, this beautiful and astonishing race could not care less about this world and its habitants. However, they still have to protect their spiral cities from the assaulting creatures of darkness, before they can reach to their own world, or to one of the phasing spiral cities that happen to appear in this world.

If player decides to create an eladrin character, s/he plays an eladrin archmage. As the bearer of enchants, speaker of invokes and conjurer of elements, eladrin are a race which carries magical energy within their blood. You had the privilege of having much higher understanding and potential to master the magical energy both within and around, making you an archmage in the eternal city. However, as mandate comes to you, you should keep darkness at bay, for the sake of your brethren and for the beauty of goodness.

Towers:

Ranger Post: Physical Tower. Having centuries to practice, decades to wait silently, and millenniums of understanding the nature itself perfected eladrin rangers keen sense, and hunting skills. Effect: Increased attack range, slightly increased attacking speed, slightly increased attacking damage. Type: Hit.

Pavilion of Magus: Arcane Tower. Magic is only natural in for eladrin, however implementing magic in an alien world requires extra skill. A magus is always ready to answer the call of service, and can be distinguished by their sect's proud pavilions. Effect: Standard attack range, standard attacking speed, increased attacking damage. Type: Hit.

Frost Spiral: Elemental Tower. Summoned directly from the plane of elemental water and air, frost spirals are beautiful and deadly combinations of two elements. These spirals carry an open portal inside, which allows eladrin soldiers to crystallize enemies as beautiful as the tower itself. Effect: Standard attack range increased attacking speed, standard attacking damage. Type: Hit cold.

Minaret of Eternal Flame: Elemental Tower. Combined conjuration from earth elementals and fire elementals, minarets of eternal flames are highly unstable and requires a deep knowledge and understanding of arcane might, as well as the appreciation of those elements. If not enough, almost always blasted creatures appreciate enough to satisfy the needs. Effect: Increased attack range, standard attacking speed, highly increased attacking damage. Type: Splash, fire.

Hydra Nest: Physical Tower. Allies of eladrin in this world, tamed hydras support eladrin not as mounts, but also as poison splitting beasts. An eladrin riding a hydra always makes people remember beauty and beast, but that story was not ending with acidic cohesion. Effect: Standard attack range, increased attacking speed, increased attacking damage. Type: Splash, poison.

Jade Portal: Arcane Tower. Jade Portal allows full might of channelization to Eladrin armies, by concentrating eternal dreams to the darkest hearts of enemies. Don't always listen to your parents, and believe in the power of dreams. Effect: Highly Increased attack range, standard attacking speed, highly increased attacking damage. Type: Continuous, slow.

Traits:

Elemental Kindred: Your birth had been celebrated in all elemental dimensions, as you are the beacon of elemental powers themselves, and a champion of natural might. Effect: Elemental towers are more effective.

Arcane Magistrate Council Member: Your elegance in the court of high king when dealing with the art of magus had not been ignored by the elders of your race. Your witty thinking and self-sacrificing loyalty to your people allowed you to have higher responsibilities, and higher capacity to use and benefit from the cumulative knowledge of Arcanum. Effect: Arcane towers are more effective.

Fey Warlord: Many consider eladrin as being peaceful and calm. You are not. You have sworn your most sincere oath to defend the eternal city and fey wilds, and to purge darkness in every way you can find. Your fierce loyalty to the cause attracted many young warriors as well. Effect: Physical towers are more effective.

B. Source Code

In this section, we will provide the source code of the Tower Defense Game project.

Arrow.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Arrow : Bullet
    {
        //Constructor
        public Arrow(Vector2 position, Enemy enemy, int damage, 
BulletManager bulletManager)
            : base(position, enemy, damage, bulletManager)
        {
            base.speed = 40;
        }

        //implements clone method for ICloneable interface
        public new Arrow Clone()
        {
            Arrow bullet = new Arrow(position, enemy, damage, 
bulletManager);
            bullet.bulletHit = this.bulletHit;

            return bullet;
        }
    }
}
```

Background.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsGame1
{
    public class Background
    {
        public enum BackgroundType { imperium, dhacaan, karrnat };
        BackgroundType backgroundType;

        //Constructor
        public Background(BackgroundType backgroundType)
        {
            this.backgroundType = backgroundType;
        }

        //sets a new Background
        public void setBackground(BackgroundType backgroundType)
        {
            this.backgroundType = backgroundType;
        }

        //gets current Background
        public BackgroundType getBackgorundType()
        {
            return backgroundType;
        }
    }
}
```

Bullet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Bullet : GameObject, ICloneable
    {
        protected Vector2 direction, targetPosition, startPosition;
        protected float speed;
        protected int damage;
        protected Enemy enemy;
        protected BulletManager bulletManager;

        public delegate void FiredEvent(Bullet sender);
        public FiredEvent bulletHit;

        //Constructor
        public Bullet(Vector2 position, Enemy enemy, int damage, 
BulletManager bulletManager)
        {
            this.enemy = enemy;
            this.bulletManager = bulletManager;
            this.position = position;
            startPosition = position;

            this.damage = damage;

            speed = 10;
        }

        //sets damage of this bullet
        public void setDamage(int damage)
        {
            this.damage = damage;
        }

        //initializes an enemy for this bullet
        public void setEnemy(Enemy enemy)
        {
            this.enemy = enemy;
            startPosition = position;
            targetPosition = enemy.getPosition() + enemy.getDirection()
* 25;
            direction = targetPosition - position;
            direction.Normalize();
        }

        //implements update method of Updateable interface
        public override void update(GameTime gameTime)
        {
            position += direction * speed;
            if ((position - startPosition).Length() > (targetPosition -
startPosition).Length())
            {
                enemy.dealDamage(damage);

                if (bulletHit != null)
                    bulletHit(this);

                bulletManager.removeBullet(this);
            }
        }

        #region ICloneable Members

        //implements clone method of IClonable interface
        public object Clone()
        {
            Bullet bullet = new Bullet(position, enemy, damage,
bulletManager);

            return bullet;
        }

        #endregion
    }
}
```

BulletManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class BulletManager : Updateable
    {
        List<Bullet> bulletList;
        GameLogic gameLogic;
        Vector2 firingPositionOffset = new Vector2(24, 24);

        Bullet fireballPrototype, cannonballPrototype,
arrowPrototype;

        //Constructor
        public BulletManager(GameLogic gameLogic)
        {
            fireballPrototype = new Fireball(Vector2.Zero, null, 0,
this);
            cannonballPrototype = new Cannonball(Vector2.Zero, null,
0, this);
            arrowPrototype = new Arrow(Vector2.Zero, null, 0, this);

            this.gameLogic = gameLogic;
            bulletList = new List<Bullet>();
        }

        //Adds new bullet to the bullet collection
        public void addBullet(Tower tower)
        {
            Bullet bullet = null;

            if (tower.getDamageType() == Tower.DamageType.fireball)
                bullet = ((Fireball)fireballPrototype).Clone();
            else if (tower.getDamageType() ==
Tower.DamageType.cannon)
                bullet = ((Cannonball)cannonballPrototype).Clone();
            else if (tower.getDamageType() == Tower.DamageType.arrow)
                bullet = ((Arrow)arrowPrototype).Clone();

            if (bullet != null)
            {
                bullet.setPosition(tower.getPosition() +
firingPositionOffset);
                bullet.setDamage(tower.getDamage());
                bullet.setEnemy(tower.getEnemy());
                bulletList.Add(bullet);
            }
        }
        //removes bullet from collection
        public void removeBullet(Bullet bullet)
        {
            bulletList.Remove(bullet);
        }

        //Removes all bullets from collection
        public void reset()
        {
            bulletList.Clear();
        }

        //calls update methods for all bullets
        public void update(GameTime gameTime)
        {
            for (int i = 0; i < bulletList.Count; i++)
                bulletList[i].update(gameTime);
        }

        //returns bulletList (Drawers can use this method to draw
bullets)
        public List<Bullet> getBulletList()
        {
            return bulletList;
        }
    }
}
```

Cannonball.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Cannonball : Bullet
    {
        //Constructor
        public Cannonball(Vector2 position, Enemy enemy, int damage,
BulletManager bulletManager)
            : base(position, enemy, damage, bulletManager)
        {
            base.speed = 6;
        }

        //Implements clone method of ICloneable interface
        public new Cannonball Clone()
        {
            Cannonball bullet = new Cannonball(position, enemy,
damage, bulletManager);
            bullet.bulletHit = this.bulletHit;
            return bullet;
        }
    }
}
```

Cow.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Cow : Enemy
    {
        //Constructor to initialize enemy attributes
        public Cow(List<Vector2> wayPoints, EnemyManager
enemyManager) : base(wayPoints, enemyManager)
        {
            speed = 1;
            maxHitPoint = 30050;
            hitPoint = maxHitPoint;
        }

        //Implements clone method of ICloneable interface
        public new Cow Clone()
        {
            Cow cow = new Cow(wayPoints, enemyManager);
            cow.enemyKill = this.enemyKill;
            cow.enemySurvived = this.enemySurvived;

            return cow;
        }
    }
}
```

DwarvenTower.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

```
namespace WindowsGame1
{
    public class DwarvenTower : Tower
    {
        //Constructor to initialize tower attributes
        public DwarvenTower(Vector2 position, TowerManager
towerManager)
        {
            base.towerManager = towerManager;
            base.position = position;
            base.level = 0;
            base.towerState = TowerState.idle;
            base.range = 140;
            base.damageType = DamageType.cannon;
            base.attackRate = 150;
            base.baseCost = 170;
            calculateDamage();
        }

        //Implements clone method of ICloneable interface
        public override object Clone()
        {
            DwarvenTower dwarvenTower = new
DwarvenTower(Vector2.Zero, towerManager);
            dwarvenTower.bulletFired = this.bulletFired;

            return dwarvenTower;
        }

        //specifies how much damage will be given by bullets of this
tower
        public override void calculateDamage()
        {
            damage = 250;
        }
    }
}
```

EladrinTower.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class EladrinTower : Tower
    {
        //Constructor to initialize tower attributes
        public EladrinTower(Vector2 position, TowerManager
towerManager)
```

```
        {
            base.towerManager = towerManager;
            base.position = position;
            base.level = 0;
            base.towerState = TowerState.idle;
            base.range = 300;
            base.damageType = DamageType.arrow;
            base.attackRate = 50;
            base.baseCost = 150;
            calculateDamage();
        }

        //Implements clone method of ICloneable interface
        public override object Clone()
        {
            EladrinTower eladrinTower = new
EladrinTower(Vector2.Zero, towerManager);
            eladrinTower.bulletFired = this.bulletFired;

            return eladrinTower;
        }

        //specifies how much damage will be given by bullets of this
tower
        public override void calculateDamage()
        {
            damage = 50;
        }
    }
}
```

Enemy.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Enemy : GameObject, ICloneable
    {
        protected enum EnemyState { running, dead, finished }
        protected EnemyState enemyState;
        protected int hitPoint, maxHitPoint, currentPoint;
        protected float speed;
        protected Vector2 direction, targetPosition, startPosition;
        protected List<Vector2> wayPoints;
        protected EnemyManager enemyManager;
        protected float rotation;

        public delegate void FiredEvent(object sender);
        public FiredEvent enemyKill;
        public FiredEvent enemySurvived;
        //Constructor to initialize enemy with its path and its
manager object
        public Enemy(List<Vector2> wayPoints, EnemyManager
enemyManager)
        {
            enemyState = EnemyState.running;
            this.enemyManager = enemyManager;
            currentPoint = 0;
            speed = 4;
            this.wayPoints = wayPoints;
            startPosition = wayPoints[currentPoint];
            targetPosition = wayPoints[currentPoint + 1];
            position = startPosition;
            direction = targetPosition - startPosition;
            direction.Normalize();
            calculateRotation();
        }

        //calclates rotation of an enemy from its direction
        void calculateRotation ()
        {
            if (direction.X == 0)
                if (direction.Y > 0)
                    rotation = (float)(Math.PI * 0.5);
                else
                    rotation = (float)Math.PI * 1.5F;
            else
                if (direction.X > 0)
                    rotation = 0;
                else
                    rotation = (float)Math.PI;
        }

        //deals damage and reduces enemies hit point
        public void dealDamage(int damage)
        {
            if (enemyState != EnemyState.dead)
            {
                hitPoint -= damage;

                if (hitPoint <= 0)
                {

                    if (enemyKill != null)
                    {
                        enemyKill(this);
                    }

                    enemyState = EnemyState.dead;
                }
            }
        }

        //returns true if enemy is inActive otherwise false.
        public bool isInactive()
        {
            if (enemyState != EnemyState.running)
                return true;
            else
                return false;
        }

        //returns true if enemy is dead otherwise false.
        public bool isDead()
        {
            if (enemyState != EnemyState.dead)
                return true;
            else
                return false;
        }

        //Implements update method of Updateable Interface
        //Various calculations done here. Enemies state is checked,
its movement is done here.
        public override void update(GameTime gameTime)
        {
            if (enemyState != EnemyState.finished)
                if ((position - startPosition).Length() >
(targetPosition - startPosition).Length())
                {
                    if (currentPoint + 2 == wayPoints.Count)
                    {
                        enemyState = EnemyState.finished;

                        if (enemySurvived != null)
                            enemySurvived(this);
                    }
                    else
                    {
                        currentPoint++;
                        startPosition = wayPoints[currentPoint];
                        targetPosition = wayPoints[currentPoint + 1];
                        position = startPosition;
                        direction = targetPosition - startPosition;
                        direction.Normalize();
                        calculateRotation();
                    }
                }

            if (enemyState == EnemyState.running)
                position += direction * speed;
        }

        //returns rotation of an enemy
        public float getRotation()
        {
            return rotation;
        }

        //return direction of an enemy
        public Vector2 getDirection()
        {
            return direction;
        }

        #region ICloneable Members

        //implements clone method for ICloneable Interface
        public object Clone()
        {
            Enemy enemy = new Enemy(wayPoints, enemyManager);
            enemy.enemyKill = this.enemyKill;

            return enemy;
        }

        #endregion
    }
}
```

EnemyManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class EnemyManager : Updateable
    {
        List<Enemy> enemyList;
        Level currentLevel;
        Map currentMap;
        int spawnTime, enemyCount;
        int spawnController = 0;
        Enemy sheepPrototype, cowPrototype;

        //Constructor to initialize new EnemyManager with given level
and map
        public EnemyManager(Level level, Map map)
        {
            currentMap = map;
            currentLevel = level;
            spawnTime = currentLevel.getSpawnTime();
            enemyCount = currentLevel.getEnemyCount();
            enemyList = new List<Enemy>();
            sheepPrototype = new Sheep(map.getWayPoints(), this);
            cowPrototype = new Cow(map.getWayPoints(), this);
        }

        //returns prototype Sheep object
        public Sheep getSheepPrototype()
        {
            return (Sheep)sheepPrototype;
        }

        //returns prototype Cow object
        public Cow getCowPrototype()
        {
            return (Cow)cowPrototype;
        }

        //sets New level for this manager
        public void setCurrentLevel(Level level)
        {
            currentLevel = level;
            spawnTime = currentLevel.getSpawnTime();
            enemyCount = currentLevel.getEnemyCount();
        }

        //Implements update method of Updateable Interface
        //New enemies are created according to specified spawn time
        //Updates all enemies in collection
        public void update(GameTime gameTime)
        {
            spawnController++;

            if (spawnController > spawnTime && enemyCount > 0)
            {
                if (currentLevel.getLevelNo() == 0)
                    addEnemy(((Sheep)sheepPrototype).Clone());
                else if (currentLevel.getLevelNo() == 1)
                    addEnemy(((Cow)cowPrototype).Clone());

                spawnController = 0;
                enemyCount--;
            }

            for (int i = 0; i < enemyList.Count; i++)
                enemyList[i].update(gameTime);
        }

        //adds enemy to the list
        public void addEnemy(Enemy enemy)
        {
            enemyList.Add(enemy);
        }

        //removes enemy from the list
        public void removeEnemy(Enemy enemy)
        {
            enemyList.Remove(enemy);
        }

        //returns enemyList
        public List<Enemy> getEnemyList()
        {
            return enemyList;
        }

        //removes all enemies from the list
        public void reset()
        {
            enemyList.Clear();
        }
    }
}
```

Fireball.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Fireball : Bullet
    {
        //Constructor
        public Fireball(Vector2 position, Enemy enemy, int damage,
BulletManager bulletManager): base(position, enemy, damage,
bulletManager)
        {
            base.speed = 10;
        }

        //implements clone method for ICloneable Interface
        public new Fireball Clone()
        {
            Fireball bullet = new Fireball(position, enemy, damage,
bulletManager);
            bullet.bulletHit = this.bulletHit;

            return bullet;
        }
    }
}
```

GameLogic.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class GameLogic : Updateable
    {
        TowerManager towerManager;
        BulletManager bulletManager;
        EnemyManager enemyManager;

        enum GameLogicState { passiveMode, buildingMode, activeMode }
        GameLogicState gameLogicState;

        Map map;
        List<Level> levelList = LevelImporter.importLevel();
        TDGame tDGame;

        int enemiesKilled = 0;
        int startingGold = 500;
        int gold;
        int currentLevel = 0;

        //Constructor to initialize a new game from stracth
        //maps, managers are kept here.
        //This class subscribes events of protoype game objects like
towers and enemies
        public GameLogic(TDGame tDGame)
        {
            gameLogicState = GameLogicState.buildingMode;

            map = MapImporter.importMap()[1];
            towerManager = new TowerManager(map, this);
            enemyManager = new EnemyManager(levelList[0], map);

            towerManager.getDwarvenTowerPrototype().bulletFired +=
new DwarvenTower.FiredEvent(onBulletFired);
            towerManager.getHumanTowerPrototype().bulletFired += new
HumanTower.FiredEvent(onBulletFired);
            towerManager.getEladrinTowerPrototype().bulletFired +=
new EladrinTower.FiredEvent(onBulletFired);

            enemyManager.getSheepPrototype().enemyKill += new
Sheep.FiredEvent(onEnemyKill);
            enemyManager.getCowPrototype().enemyKill += new
Cow.FiredEvent(onEnemyKill);

            enemyManager.getSheepPrototype().enemySurvived += new
Sheep.FiredEvent(onEnemySurvived);
            enemyManager.getCowPrototype().enemySurvived += new
Cow.FiredEvent(onEnemySurvived);

            bulletManager = new BulletManager(this);
            this.tDGame = tDGame;

            gold = startingGold;
        }

        //returns tDGame instance
        public TDGame getTDGame()
        {
            return tDGame;
        }

        //returns total number of kills
        public int getNumberOfKills()
        {
            return enemiesKilled;
        }

        //returns current gold amount
        public int getGold()
        {
            return gold;
        }

        //sets a new gold value
        public void setGold(int gold)
        {
            this.gold = gold;
        }

        //activates game logic, sets it to building mode, subscribes
to gameplayfield events.
        public void activate()
        {

this.getTDGame().getGamePlayScreen().getGamePlayField().mouseClick +=
new GamePlayField.FiredEvent(onGamePlayScreenClick);
            setToBuildingMode();
        }

        //deactivates game logic, sets it to passive mode,
unsubscribes from gameplayfield events.
        public void deActivate()
        {

this.getTDGame().getGamePlayScreen().getGamePlayField().mouseClick -=
onGamePlayScreenClick;
            setToPassiveMode();
        }

        //sets gamelogic to building mode in where no enemies are
spawned but towers can be built
        public void setToBuildingMode()
        {
            gameLogicState = GameLogicState.buildingMode;

            enemyManager.reset();
            enemyManager.setCurrentLevel(levelList[currentLevel]);
        }

        //sets gamelogic to active mode, enemies start to spawn
        public void setToActiveMode()
        {
            gameLogicState = GameLogicState.activeMode;
        }

        //terminates the game
        public void setToPassiveMode()
        {
            towerManager.reset();
            enemyManager.reset();
            bulletManager.reset();
            gold = startingGold;
            currentLevel = 0;
            enemiesKilled = 0;
            gameLogicState = GameLogicState.passiveMode;
        }

        #region Updateable Members
        //calls update method of all managers according to current
gamelogic state.
        public void update(GameTime gameTime)
        {
            if (gameLogicState == GameLogicState.activeMode)
            {
                towerManager.update(gameTime);
                bulletManager.update(gameTime);
                enemyManager.update(gameTime);

                if (enemiesKilled ==
levelList[currentLevel].getEnemyCount())
                {
                    currentLevel++;
                    enemiesKilled = 0;
                    setToBuildingMode();
                    bulletManager.reset();
                }
            }
            else if (gameLogicState == GameLogicState.buildingMode)
                towerManager.update(gameTime);
        }

        #endregion

        //returns associated towermanager
        public TowerManager getTowerManager()
        {
            return towerManager;
        }

        //specifies which actions to be taken on game play screen
click
        public void onGamePlayScreenClick(object sender)
        {

towerManager.addTower(tDGame.getGamePlayScreen().getSelectedTowerType
(), new Vector2(Mouse.GetState().X - Mouse.GetState().X % 48,
Mouse.GetState().Y - Mouse.GetState().Y % 48), gold);
            tDGame.getGamePlayScreen().setSelectedTowerType(-1);
        }

        //returns current map
        public Map getMap()
        {
            return map;
        }

        //returns associated bullet manager
        public BulletManager getBulletManager()
        {
            return bulletManager;
        }

        //returns associated enemy manager
        public EnemyManager getEnemyManager()
        {
            return enemyManager;
        }

        //adds a new bullet to a colletion on bullet Fired event
        public void onBulletFired(Tower tower)
        {
            bulletManager.addBullet(tower);
        }

        //returns level list
        public List<Level> getLevelList()
        {
            return levelList;
        }

        //on enemy kill increases gold amound, and number of killed
enemies
        public void onEnemyKill(object sender)
        {
            enemiesKilled++;
            gold += 20;
        }

        //on enemy survival decreases gold amound, and number of
killed enemies
        public void onEnemySurvived(object sender)
        {
            enemiesKilled++;
            gold -= 20;
        }
    }
}
```

GameObject.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public abstract class GameObject : Updateable
    {
        protected string name = "Game Object";
        protected Vector2 position;

        #region Updateable Members

        //abstract Update method for game objects
        public abstract void update(GameTime gameTime);

        //returns position of a GameObject
        public Vector2 getPosition()
        {
            return position;
        }

        //sets a new Position for this game object
        public void setPosition(Vector2 position)
        {
            this.position = position;
        }
        #endregion
    }
}
```

GameScreenManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class GameScreenManager
    {
        TDGame tDGame;
        GameScreen activeGameScreen;
        List<GameScreen> gameScreenList;
        public enum GameScreenState { mainMenuScreen, optionsScreen, 
gamePlayScreen };
        GameScreenState gameScreenState;

        //Constructor
        public GameScreenManager(TDGame tDGame)
        {
            this.tDGame = tDGame;
            gameScreenList = new List<GameScreen>();
            gameScreenState = GameScreenState.mainMenuScreen;
        }

        //adds a new game screen to the collection
        public void addGameScreen(GameScreen gameScreen)
        {
            gameScreenList.Add(gameScreen);
            gameScreen.setGameScreenManager(this);
        }

        //draws active game screen
        public void draw()
        {
            if (activeGameScreen is GamePlayScreen)
                ((GamePlayScreen)activeGameScreen).draw();
            else
                activeGameScreen.draw();
        }

        //changes game screen with given gameScreenState
        public void changeGameScreen(GameScreenState gameScreenState)
        {
            this.gameScreenState = gameScreenState;

            if (activeGameScreen != null)
                activeGameScreen.deActivate();
            switch (gameScreenState)
            {
                case (GameScreenState.mainMenuScreen):
                {
                    foreach (GameScreen gameScreen in gameScreenList)
                        if (gameScreen is MainMenuScreen)
                        {
                            activeGameScreen = gameScreen;

((MainMenuScreen)activeGameScreen).activate();
                        }
                } break;
                case (GameScreenState.optionsScreen):
                {
                    foreach (GameScreen gameScreen in gameScreenList)
                        if (gameScreen is OptionsScreen)
                        {
                            activeGameScreen = gameScreen;

((OptionsScreen)activeGameScreen).activate();
                        }
                } break;
                case (GameScreenState.gamePlayScreen):
                {
                    foreach (GameScreen gameScreen in gameScreenList)
                        if (gameScreen is GamePlayScreen)
                        {
                            activeGameScreen = gameScreen;

((GamePlayScreen)activeGameScreen).activate();
                        }
                } break;
            }
        }
    }
}
```

HumanTower.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class HumanTower : Tower
    {
        //Constructor to initialize tower attributes
        public HumanTower(Vector2 position, TowerManager
towerManager)
        {
            base.towerManager = towerManager;
            base.position = position;
            base.level = 0;
            base.towerState = TowerState.idle;
            base.range = 220;
            base.damageType = DamageType.fireball;
            base.attackRate = 80;
            base.baseCost = 110;
            calculateDamage();
        }

        //Implements clone method of ICloneable interface
        public override object Clone()
        {
            HumanTower humanTower = new HumanTower(Vector2.Zero, 
towerManager);
            humanTower.bulletFired = this.bulletFired;

            return humanTower;
        }

        //specifies how much damage will be given by bullets of this 
tower
        public override void calculateDamage()
        {
            damage = 100;
        }
    }
}
```

InputManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class InputManager : Updateable
    {
        TDGame tDGame;

        MouseState mouseState, previousMouseState;
        KeyboardState keyboardState, previousKeyboardState;

        public delegate void FiredEvent(InputManager sender);
        public FiredEvent mouseMoved;
        public FiredEvent mouseLeftPressed;
        public FiredEvent mouseLeftReleased;
        public FiredEvent mouseRightPressed;
        public FiredEvent mouseRightReleased;

        //Constructor
        public InputManager(TDGame tDGame)
        {
            this.tDGame = tDGame;
        }

        #region Updateable Members

        //Updates input state 60 times in 1 second and fires
appropriate events according to input states
        public void update(GameTime gameTime)
        {
            previousMouseState = mouseState;
            previousKeyboardState = keyboardState;

            mouseState = Mouse.GetState();
            keyboardState = Keyboard.GetState();

            //if (previousMouseState.X != mouseState.X ||
previousMouseState.Y != mouseState.Y)
                if (mouseMoved != null)
                    mouseMoved(this);

            if (previousMouseState.LeftButton == ButtonState.Pressed
&& mouseState.LeftButton == ButtonState.Released)
                if (mouseLeftReleased != null)
                    mouseLeftReleased(this);

            if (previousMouseState.LeftButton == ButtonState.Released
&& mouseState.LeftButton == ButtonState.Pressed)
                if (mouseLeftPressed != null)
                    mouseLeftPressed(this);

            if (previousMouseState.RightButton == ButtonState.Pressed
&& mouseState.RightButton == ButtonState.Released)
                if (mouseRightReleased != null)
                    mouseRightReleased(this);

            if (previousMouseState.RightButton ==
ButtonState.Released && mouseState.RightButton ==
ButtonState.Pressed)
                if (mouseRightPressed != null)
                    mouseRightPressed(this);
        }

        //returns mouse position
        public Vector2 getMousePosition()
        {
            return new Vector2(mouseState.X, mouseState.Y);
        }

        #endregion
    }
}
```

Level.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace WindowsGame1
{
    public class Level
    {
        int levelNo, spawnTime, enemyCount, enemyType;

        //Constructor to initalize new level with given parameters
        public Level(int levelNo, int spawnTime, int enemyCount, int
enemyType)
        {
            this.levelNo = levelNo;
            this.spawnTime = spawnTime;
            this.enemyCount = enemyCount;
            this.enemyType = enemyType;
        }

        //returns enemyCount
        public int getEnemyCount()
        {
            return enemyCount;
        }

        //returns spawnTime
        public int getSpawnTime()
        {
            return spawnTime;
        }

        //returns levelNo
        public int getLevelNo()
        {
            return levelNo;
        }

        //returns enemyType
        public int getEnemyType()
        {
            return enemyType;
        }
    }
}
```

LevelImporter.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class LevelImporter
    {
        static String levelFileAddress = "Content\\Levels.htk";

        //static method to import level specifications from
Levels.htk file.
        public static List<Level> importLevel()
        {
            TextReader textReader = new
StreamReader(levelFileAddress);

            String levelLine = textReader.ReadLine();

            List<Level> levelList = new List<Level>();
            int count = 0;
            while (levelLine != null)
            {

                String[] tokens = levelLine.Split(' ');

                levelList.Add(new Level(count, int.Parse(tokens[0]),
int.Parse(tokens[1]), int.Parse(tokens[2])));

                levelLine = textReader.ReadLine();
                count++;
            }

            textReader.Close();

            return levelList;
        }

        //static method to export level specifications to Levels.htk
file.
        public static void exportLevel(List<Level> levelList)
        {
            TextWriter textWriter = new
StreamWriter(levelFileAddress);

            foreach (Level level in levelList)
            {
                textWriter.WriteLine(level.getSpawnTime()+"
"+level.getEnemyCount()+" "+level.getEnemyType());
            }
            textWriter.Close();
        }

    }
}
```

Map.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Map
    {
        int[,] map;
        String name;
        String textureName;
        const int mapSize = 9;
        const int tileSize = 96;

        //Constructor
        public Map()
        {
            name = "Default Map";
            textureName = "Default";

            map = new int[mapSize, mapSize]{{0,7,0,0,0,0,0,0,0},
                                            {0,2,0,0,0,0,3,1,8},
                                            {0,2,0,0,0,0,2,0,0},
                                            {0,2,0,0,0,0,2,0,0},
                                            {0,2,0,0,0,0,2,0,0},
                                            {0,4,1,1,5,0,2,0,0},
                                            {0,0,0,0,4,1,6,0,0},
                                            {0,0,0,0,0,0,0,0,0},
                                            {0,0,0,0,0,0,0,0,0}};
        }

        //Constructor to initialize a map manually
        public Map(String name, String textureName, int[,] map)
        {
            this.name = name;
            this.textureName = textureName;
            this.map = map;
        }

        //returns waypoint list from map array.
        public List<Vector2> getWayPoints()
        {
            Vector2 entry, exit;
            entry = Vector2.Zero;
            exit = Vector2.Zero;
            List<Vector2> wayPoints = new List<Vector2>();

            int currentX = 0;
            int currentY = 0;
            int previousX = 0;
            int previousY = 0;

            for (int i = 0; i < mapSize; i++)
            {
                for (int j = 0; j < mapSize; j++)
                {
                    if (map[i, j] == 7)
                    {
                        entry = new Vector2(j * tileSize, i *
tileSize);
                        currentX = j;
                        currentY = i;
                        previousX = j;
                        previousY = i;
                    }
                    if (map[i, j] == 8)
                        exit = new Vector2(j * tileSize, i *
tileSize);
                }
            }

            wayPoints.Add(entry);

            if (currentX == 0)
            {
                currentX++;
            }
            else if (currentY == 0)
            {
                currentY++;
            }
            else if (currentX == mapSize - 1)
            {
                currentX--;
            }
            else if (currentY == mapSize - 1)
            {
                currentY--;
            }

            int dirX, dirY;
            dirX = currentX - previousX;
            dirY = currentY - previousY;

            while (map[currentY, currentX] != 8)
            {

                int tile = map[currentY, currentX];

                if (tile == 3)
                {
                    if (dirX == -1)
                    {
                        dirX = 0;
                        dirY = 1;
                    }
                    else // dirY == 1
                    {
                        dirX = 1;
                        dirY = 0;
                    }

                }
                else if (tile == 4)
                {
                    if (dirX == -1)
                    {
                        dirX = 0;
                        dirY = -1;
                    }
                    else // dirY == -1
                    {
                        dirX = 1;
                        dirY = 0;
                    }

                }
                else if (tile == 5)
                {
                    if (dirX == 1)
                    {
                        dirX = 0;
                        dirY = 1;
                    }
                    else // dirY == -1
                    {
                        dirX = -1;
                        dirY = 0;
                    }

                }
                else if (tile == 6)
                {
                    if (dirX == 1)
                    {
                        dirX = 0;
                        dirY = -1;
                    }
                    else // dirY == 1
                    {
                        dirX = -1;
                        dirY = 0;
                    }

                }

                if (tile < 7 && tile > 2)
                    wayPoints.Add(new Vector2(currentX * tileSize,
currentY * tileSize));

                currentX = currentX + dirX;
                currentY = currentY + dirY;
            }

            wayPoints.Add(exit);

            return wayPoints;

        }

        //returns map array
        public int[,] getMap()
        {
            return map;
        }

        //returns mapsize
        public int getMapSize()
        {
            return mapSize;
        }

        //returns map texture name
        public String getTextureName()
        {
            return textureName;
        }

        //returns map name
        public String getName()
        {
            return name;
        }
    }
}
```

MapImporter.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class MapImporter
    {
        static String mapFileAddress = "Content\\Maps.htk";
        static int mapSize = 9;

        //static method to import map specifications from Map.htk
file.
        public static List<Map> importMap()
        {
            TextReader textReader = new StreamReader(mapFileAddress);

            String mapName = textReader.ReadLine();

            List<Map> mapList = new List<Map>();

            while (mapName != null)
            {

                String textureName = textReader.ReadLine();
                String mapInfoLine = textReader.ReadLine();
                int[,] mapArray = new int[mapSize,mapSize];
                for(int i=0; i<mapSize; i++)
                {
                    for(int j=0; j<mapSize; j++)
                    {
                        mapArray[i,j] = int.Parse(
mapInfoLine.Substring(((i*mapSize)+j)*2,1) );
                    }
                }
                mapList.Add(new Map(mapName, textureName, mapArray));

                mapName = textReader.ReadLine();
            }

            textReader.Close();

            return mapList;
        }

        //static method to export map specifications to Map.htk file.
        public static void exportMap(List<Map> mapList)
        {
            TextWriter textWriter = new StreamWriter(mapFileAddress);

            foreach (Map map in mapList)
            {
                textWriter.WriteLine(map.getName());
                textWriter.WriteLine(map.getTextureName());
                for (int i = 0; i < mapSize; i++)
                {
                    for (int j = 0; j < mapSize; j++)
                    {
                        textWriter.Write(map.getMap()[i, j] + ",");
                    }
                }
                textWriter.Write(textWriter.NewLine);
            }
            textWriter.Close();
        }
    }
}
```

MouseListener.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsGame1
{
    interface MouseListener
    {
        //interface for mouse events

        void onMouseMove(InputManager sender);
        void onMouseLeftPressed(InputManager sender);
        void onMouseLeftReleased(InputManager sender);
        void onMouseRightPressed(InputManager sender);
        void onMouseRightReleased(InputManager sender);
        void activate();
    }
}
```

Profile.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsGame1
{
    public class Profile
    {
        Background background;
        Trait trait;

        //Constructor to initialize a new profile with default 
characteristics
        public Profile()
        {
            trait = new Trait(Trait.TraitType.warlord);
            background = new 
Background(Background.BackgroundType.dhacaan);
        }

        //returns current trait object
        public Trait getTrait()
        {
            return trait;
        }

        //sets new trait
        public void setTrait(Trait trait)
        {
            this.trait = trait;
        }

        //returns current background object
        public Background getBackground()
        {
            return background;
        }

        //sets new background
        public void setBackground(Background background)
        {
            this.background = background;
        }
    }
}
```

Program.cs

```
using System;

namespace WindowsGame1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (TDGame game = new TDGame())
            {
                game.IsMouseVisible = true;
                game.Run();
            }
        }
    }
}
```

Sheep.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Sheep : Enemy
    {
        //Constructor to initialize enemy attributes
        public Sheep(List<Vector2> wayPoints, EnemyManager
enemyManager) : base(wayPoints, enemyManager)
        {
            speed = 1;
            maxHitPoint = 8050;
            hitPoint = maxHitPoint;
        }

        //Implements clone method of ICloneable interface
        public new Sheep Clone()
        {
            Sheep sheep = new Sheep(wayPoints, enemyManager);
            sheep.enemyKill = this.enemyKill;
            sheep.enemySurvived = this.enemySurvived;

            return sheep;
        }
    }
}
```

SoundManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class SoundManager : Updateable
    {
        AudioEngine audioEngine;
        WaveBank waveBank;
        SoundBank soundBank;
        TDGame tDGame;

        //Constructor
        public SoundManager(TDGame tDGame)
        {
            this.tDGame = tDGame;
            audioEngine = new AudioEngine("Content/xactProject.xgs");
            waveBank = new WaveBank(audioEngine,
"Content/myWaveBank.xwb");
            soundBank = new SoundBank(audioEngine,
"Content/mySoundBank.xsb");
        }

        //subscribe to the events that needs a sound effect
        public void register()
        {

tDGame.getGameLogic().getTowerManager().getDwarvenTowerPrototype().bu
lletFired += new Tower.FiredEvent(playCannonSound);

tDGame.getGameLogic().getTowerManager().getEladrinTowerPrototype().bu
lletFired += new Tower.FiredEvent(playArrowSound);

tDGame.getGameLogic().getTowerManager().getHumanTowerPrototype().bull
etFired += new Tower.FiredEvent(playFireballSound);
        }

        //plays cannon sound
        public void playCannonSound(object sender)
        {
            soundBank.PlayCue("flamestrike");
        }

        //plays arrow sound
        public void playArrowSound(object sender)
        {
            soundBank.PlayCue("Arrow");
        }

        //plays fireball sound
        public void playFireballSound(object sender)
        {
            soundBank.PlayCue("Fireball");
        }
        #region Updateable Members

        //implement updateable method of Updateable interface
        public void update(GameTime gameTime)
        {
            audioEngine.Update();
        }

        #endregion
    }
}
```

TDGame.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class TDGame : Game
    {
        public GraphicsDeviceManager graphics;
        Drawer drawer;
        SpriteFont defaultFont;

        GameLogic gameLogic;

        SoundManager soundManager;
        public InputManager inputManager;
        GameScreenManager gameScreenManager;

        GameScreen mainManuScreen, optionsScreen, gamePlayScreen;

        //Initializes a new game, sets content directory, specifies
screen size.
        public TDGame()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            graphics.PreferredBackBufferHeight = 768;
            graphics.PreferredBackBufferWidth = 1024;
        }

        //initializes game
        protected override void Initialize()
        {
            base.Initialize();
        }
        //load graphical content (fonts, textures).
        //instantianate drawer, manager classes and screens
        protected override void LoadContent()
        {
            drawer = new Drawer(GraphicsDevice, graphics);

            defaultFont = Content.Load<SpriteFont>("DefaultFont");
            drawer.addSpriteFont("Default Font",defaultFont);

            gameLogic = new GameLogic(this);

            soundManager = new SoundManager(this);
            soundManager.register();

            inputManager = new InputManager(this);

            gameScreenManager = new GameScreenManager(this);

            mainManuScreen = new MainMenuScreen(this, drawer);
            optionsScreen = new OptionsScreen(this, drawer);
            gamePlayScreen = new GamePlayScreen(this, drawer);

            gameScreenManager.addGameScreen(mainManuScreen);
            gameScreenManager.addGameScreen(optionsScreen);
            gameScreenManager.addGameScreen(gamePlayScreen);

gameScreenManager.changeGameScreen(GameScreenManager.GameScreenState.
mainMenuScreen);
        }

        protected override void UnloadContent(){}

        //Main update loop of a game
        protected override void Update(GameTime gameTime)
        {
            inputManager.update(gameTime);
            gameLogic.update(gameTime);
            soundManager.update(gameTime);

            base.Update(gameTime);
        }

        //calls draw method of gameScreenManager
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.Black);

            drawer.Begin(SpriteBlendMode.AlphaBlend,
SpriteSortMode.FrontToBack, SaveStateMode.None);
            gameScreenManager.draw();
            drawer.End();
        }

        //returns gameLogic
        public GameLogic getGameLogic()
        {
            return gameLogic;
        }

        //returns gamePlayScreen
        public GamePlayScreen getGamePlayScreen()
        {
            return (GamePlayScreen)gamePlayScreen;
        }
    }
}
```

Tower.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public abstract class Tower : GameObject, ICloneable
    {
        protected enum TowerState { idle , loading, shot};
        public enum DamageType { cannon, fireball, arrow };

        protected TowerState towerState;
        protected DamageType damageType;

        protected TowerManager towerManager;

        protected float range, attackRate;
        protected int level, damage, baseCost ,lastAttacked = 0;
        protected Enemy enemy;

        public delegate void FiredEvent(Tower sender);
        public FiredEvent bulletFired;

        //returns damageType of a tower
        public DamageType getDamageType()
        {
            return damageType;
        }

        //returns amount of damage of this tower
        public int getDamage()
        {
            return damage;
        }

        //returns building cost of this tower
        public int getCost()
        {
            return baseCost;
        }

        //abstract method to calculate damage of a tower
        public abstract void calculateDamage();

        //check ranges, if there is, sets a new target enemy
        public void checkRange(List<Enemy> enemyList)
        {
            if (enemy != null)
                if (enemy.isInactive() || (position -
enemy.getPosition()).Length() > range )
                    enemy = null;

            if (enemy == null)
            for (int i = 0; i < enemyList.Count; i++)
            {
                if ((position - enemyList[i].getPosition()).Length()
< range && !enemyList[i].isInactive())
                {
                    enemy = enemyList[i];
                    i = enemyList.Count + 1;
                }
            }
        }

        //returns current target enemy
        public Enemy getEnemy()
        {
            return enemy;
        }

        #region ICloneable Members

        //abstarct method for clone
        public abstract object Clone();

        #endregion

        //updates tower, if reloading time has passes, and enemy is
in range, fires a bullet
        public override void update(GameTime gameTime)
        {
            lastAttacked++;

            if (lastAttacked > attackRate && enemy!=null)
            {
                lastAttacked = 0;

                if (bulletFired != null)
                    bulletFired(this);
            }
        }
    }
}
```

TowerManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class TowerManager : Updateable
    {
        Tower dwarvenTowerPrototype, humanTowerPrototype,
eladrinTowerPrototype;
        List<Tower> towerList;
        Map map;
        GameLogic gameLogic;

        //Construcutor
        public TowerManager(Map map, GameLogic gameLogic)
        {
            this.gameLogic = gameLogic;
            this.map = map;
            towerList = new List<Tower>();

            dwarvenTowerPrototype = new DwarvenTower(Vector2.Zero,
this);
            humanTowerPrototype = new HumanTower(Vector2.Zero, this);
            eladrinTowerPrototype = new EladrinTower(Vector2.Zero,
this);
        }

        //returns Dwarven tower prototype
        public DwarvenTower getDwarvenTowerPrototype()
        {
            return (DwarvenTower)dwarvenTowerPrototype;
        }

        //returns Human tower prototype
        public HumanTower getHumanTowerPrototype()
        {
            return (HumanTower)humanTowerPrototype;
        }

        //returns Eladrin tower prototype
        public EladrinTower getEladrinTowerPrototype()
        {
            return (EladrinTower)eladrinTowerPrototype;
        }

        //method to add tower: check availability of building process
by doing some controls: checks gold, checks map (place is occupied?
etc)
```

```
        public void addTower(int towerType, Vector2 position, int
gold)
        {
            Tower tower = null;
            bool available = true;

            foreach (Tower requestedTower in towerList)
            {
                if (requestedTower.getPosition() == position)
                    available = false;
            }

            if (map.getMap()[((int)position.Y + 48) / 96,
((int)position.X + 48) / 96] != 0)
            {
                available = false;
            }

            if (available)
            {
                switch (towerType)
                {
                    case (0):
                        tower =
(DwarvenTower)dwarvenTowerPrototype.Clone();
                        break;
                    case (1):
                        tower =
(HumanTower)humanTowerPrototype.Clone();
                        break;
                    case (2):
                        tower =
(EladrinTower)eladrinTowerPrototype.Clone();
                        break;
                    default:
                        tower =
(EladrinTower)eladrinTowerPrototype.Clone();
                        break;

                }

                if (getGameLogic().getGold() >= tower.getCost())
                {
                    tower.setPosition(position);
                    towerList.Add(tower);
                    getGameLogic().setGold(getGameLogic().getGold() -
tower.getCost());
                }
            }
        }

        //method to remove a tower from list
        public void removeTower(Vector2 position)
        {
            towerList.RemoveAt(0);
        }

        //removes all towers from list
        public void reset()
        {
            towerList.Clear();
        }

        #region Updateable Members

        //updates all towers in a list, and makes them check their
range to find enemies if they don't have
        public void update(GameTime gameTime)
        {
            foreach (Tower tower in towerList)
            {

tower.checkRange(getGameLogic().getEnemyManager().getEnemyList());
                tower.update(gameTime);
            }
        }

        #endregion

        //returns towerList
        public List<Tower> getTowerList()
        {
            return towerList;
        }

        //returns gameLogic
        public GameLogic getGameLogic()
        {
            return gameLogic;
        }
    }
}
```

Trait.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsGame1
{
    public class Trait
    {
        public enum TraitType { archmage, warlord, engineer };
        TraitType traitType;

        //Constructor
        public Trait(TraitType traitType)
        {
            this.traitType = traitType;
        }

        //sets a new trait
        public void setTrait(TraitType traitType)
        {
            this.traitType = traitType;
        }

        //gets current trait
        public TraitType getTraitType()
        {
            return traitType;
        }
    }
}
```

Updateable.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    //interface for updateable game elements
    interface Updateable
    {
        //Any object that implenets this method can be updated
    }
}
```

BulletImageField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class BulletImageField : ScreenComponent
    {
        List<Bullet> bulletList;
        Vector2 pivot;


        //Constructor
        public BulletImageField(String textureName, Drawer drawer,
List<Bullet> bulletList)
        {
            this.bulletList = bulletList;
            base.drawer = drawer;
            base.texture2D = base.drawer.getTexture2D(textureName);
            pivot = new Vector2(base.texture2D.Width / 2,
base.texture2D.Height / 2);
            destinationRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);
            sourceRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);
        }

        //Draws all bullets in bulletList using drawer object
        public override void draw(Drawer drawer)
        {
            for (int i = 0; i < bulletList.Count; i++ )
            {
                if (bulletList[i] is Fireball)
                    setTexture("fireball");
                else if (bulletList[i] is Cannonball)
                    setTexture("cannonball");
                else if (bulletList[i] is Arrow)
                    setTexture("arrow");

                destinationRectangle.X =
(int)bulletList[i].getPosition().X;
                destinationRectangle.Y =
(int)bulletList[i].getPosition().Y;
                drawer.Draw(texture2D, destinationRectangle,
sourceRectangle, Color.White, 0.0f, pivot, SpriteEffects.None,
layerIndex);
            }
        }

        //Used to set a new texture for this component
        public void setTexture(String textureName)
        {
            base.setTexture(textureName);
            pivot = new Vector2(base.texture2D.Width / 2,
base.texture2D.Height / 2);
        }


    }
}
```

Button.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Button : ScreenComponent, MouseListener
    {
        enum ButtonState { idle, mouseOver, mouseClick, disabled };
        ButtonState buttonState;
        String text;
        Vector2 textPosition;
        Vector2 pressedPosition;
        SpriteFont spriteFont;
        Color color;
        bool textIsVisible = true;

        public delegate void FiredEvent(Button sender);
        public FiredEvent buttonClick;


        //Constructor
        public Button(String text, String textureName, Drawer drawer)
        {
            color = Color.White;
            spriteFont = drawer.getSpriteFont("Default Font");
            this.text = text;
            buttonState = ButtonState.idle;
            base.texture2D = drawer.getTexture2D(textureName);
            destinationRectangle = new Rectangle(0, 0,
base.texture2D.Width , base.texture2D.Height / 3);
            sourceRectangle = new Rectangle(0, 0,
base.texture2D.Width , base.texture2D.Height / 3);
            pressedPosition = new Vector2(0, 0);
            textPosition = new Vector2((destinationRectangle.X +
(destinationRectangle.Width - spriteFont.MeasureString(text).X) / 2),
(destinationRectangle.Y + (destinationRectangle.Height -
spriteFont.MeasureString(text).Y) / 2));
        }
        //Changes text of a button
        public void setText(String text)
        {
            this.text = text;
        }

        //Returns current text of a button
        public String getText()
        {
            return text;
        }

        //Initializes button to idle state and subscribes to keyboard
events
        public void activate()
        {
            buttonState = ButtonState.idle;
            gameScreen.getTDGame().inputManager.mouseMoved += new
InputManager.FiredEvent(onMouseMove);
            gameScreen.getTDGame().inputManager.mouseLeftPressed +=
new InputManager.FiredEvent(onMouseLeftPressed);
            gameScreen.getTDGame().inputManager.mouseLeftReleased +=
new InputManager.FiredEvent(onMouseLeftReleased);
        }

        //Sets button to disabled state and unsubscribes from
keyboard events
        public void deActivate()
        {
            gameScreen.getTDGame().inputManager.mouseMoved -=
onMouseMove;
            gameScreen.getTDGame().inputManager.mouseLeftPressed -=
onMouseLeftPressed;
            gameScreen.getTDGame().inputManager.mouseLeftReleased -=
onMouseLeftReleased;
            buttonState = ButtonState.disabled;
        }

        //Sets button position and readjusts text position
        public void setPosition(Vector2 position)
        {
            base.setPosition(position);
            recalculateTextPosition();
        }

        //Scales button with given vector
        public void setScale(Vector2 position)
        {
            base.setScale(position);
            recalculateTextPosition();
        }

        //Aligns text to the center of the button
        void recalculateTextPosition()
        {
            textPosition = new Vector2((destinationRectangle.X +
(destinationRectangle.Width - spriteFont.MeasureString(text).X) / 2),
(destinationRectangle.Y + (destinationRectangle.Height -
spriteFont.MeasureString(text).Y) / 2));
```

```
        }

        //Draws button to the screen according to its state
        public override void draw(Drawer drawer)
        {
            switch (buttonState)
            {
                case (ButtonState.disabled):
                {
                        sourceRectangle.Y = 0;
                        color = Color.DarkGray;
                        break;
                }
                case(ButtonState.idle):
                {
                    sourceRectangle.Y = 0;
                    color = Color.White;
                    break;
                }
                case(ButtonState.mouseOver):
                {
                    sourceRectangle.Y = texture2D.Height / 3;
                    break;
                }
                case (ButtonState.mouseClick):
                {
                    sourceRectangle.Y = texture2D.Height / 3 * 2;
                    break;
                }
            }
            drawer.Draw(texture2D, destinationRectangle,
sourceRectangle, color, 0.0f, Vector2.Zero, SpriteEffects.None,
layerIndex);
            if (textIsVisible)
                drawer.DrawString(spriteFont, text, textPosition,
color, 0.0f, Vector2.Zero, Vector2.One, SpriteEffects.None,
(layerIndex + 0.001f));
        }

        //When called, text of a button won't be displayed
        public void disableText()
        {
            textIsVisible = false;
        }

        //Enables display of a text on button
        public void enableText()
        {
            textIsVisible = true;
        }

        #region MouseListener Members

        //Specifies which actions to be done when mouse is over this
button
        public void onMouseMove(InputManager sender)
        {
            if (new Rectangle((int)sender.getMousePosition().X,
(int)sender.getMousePosition().Y, 1,
1).Intersects(destinationRectangle))
            {
                if (buttonState == ButtonState.mouseClick)
                    buttonState = ButtonState.mouseClick;
                else
                    buttonState = ButtonState.mouseOver;
            }
            else
                buttonState = ButtonState.idle;
        }

        //Specifies which actions to be done when left button of a 
mouse is pressed
        public void onMouseLeftPressed(InputManager sender)
        {
                if (buttonState == ButtonState.mouseOver)
                    buttonState = ButtonState.mouseClick;

                pressedPosition = sender.getMousePosition();
        }

        //Specifies which actions to be done when left button of a 
mouse is released
        public void onMouseLeftReleased(InputManager sender)
        {
            if (new Rectangle((int)sender.getMousePosition().X, 
(int)sender.getMousePosition().Y, 1, 
1).Intersects(destinationRectangle))
            {
                buttonState = ButtonState.mouseOver;

                if (new Rectangle((int)pressedPosition.X, 
(int)pressedPosition.Y, 1, 1).Intersects(destinationRectangle))
                    if (buttonClick != null)
                        buttonClick(this);
            }
            else
                buttonState = ButtonState.idle;

        }

        //Specifies which actions to be done when right button of a 
mouse is pressed
        public void onMouseRightPressed(InputManager sender)
        {
            //throw new NotImplementedException();
        }

        //Specifies which actions to be done when right button of a 
mouse is released
        public void onMouseRightReleased(InputManager sender)
        {
            //throw new NotImplementedException();
        }

        #endregion
    }
}
```

Drawable.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    //Interface to provide drawing methods to objects that want to
draw itself
    interface Drawable
    {
        //Any object that implenets this method can draw itself on
the screen
        void draw(Drawer drawer);
    }
}
```

Drawer.cs

```
using System.IO;
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Drawer : SpriteBatch
    {
        public GraphicsDeviceManager graphics;

        //Inner class to store textures with names
        private class NamedTexture2D
        {
            //Constructor
            public NamedTexture2D(String name, Texture2D texture2D)
            {
                this.name = name;
                this.texture2D = texture2D;
            }
            public String name;
            public Texture2D texture2D;
        }

        //Inner class to store fonts with names
        private class NamedSpriteFont
        {
            //Constructor
            public NamedSpriteFont(String name, SpriteFont 
spriteFont)
            {
                this.name = name;
                this.spriteFont = spriteFont;
            }

            public String name;
            public SpriteFont spriteFont;
        }

        List<NamedTexture2D> namedTexture2DList = new 
List<NamedTexture2D>();
        List<NamedSpriteFont> namedSpriteFontList = new 
List<NamedSpriteFont>();

        //Constructor
        public Drawer(GraphicsDevice graphicsDevice, 
GraphicsDeviceManager graphicsDeviceManager) : base(graphicsDevice)
        {
            loadAllTexture2DFiles();
            this.graphics = graphicsDeviceManager;
        }

        //Switch between fullscreen and windowed mode
        public void toggleFullScreen()
        {
            graphics.ToggleFullScreen();
        }

        //Used to load all textures (*.png, *.jpg) to the memory in 
content folder
        public void loadAllTexture2DFiles()
        {
            string[] filePaths = Directory.GetFiles(@"Content\\", 
"*.jpg");

            for (int i = 0; i < filePaths.Length; i++)
                addTexture2D(filePaths[i].Substring(9, 
filePaths[i].Length - 9));

            filePaths = Directory.GetFiles(@"Content\\", "*.png");

            for (int i = 0; i < filePaths.Length; i++)
                addTexture2D(filePaths[i].Substring(9, 
filePaths[i].Length - 9));
        }

        //Adds a texture to a texture collection with given name
        public void addTexture2D(String name, Texture2D texture2D)
        {
            namedTexture2DList.Add(new NamedTexture2D(name, 
texture2D));
```

```
        }

        //Adds a texture to a texture collection with given name from
file
        public void addTexture2D(String fileName)
        {
            Texture2D texture2D = Texture2D.FromFile(GraphicsDevice,
@"Content\\" + fileName);
            addTexture2D(fileName.Substring(0, fileName.Length - 4),
texture2D);
        }

        //Adds a font to a font collection with given name
        public void addSpriteFont(String name, SpriteFont spriteFont)
        {
            namedSpriteFontList.Add(new NamedSpriteFont(name,
spriteFont));
        }

        //returns font with given name
        public SpriteFont getSpriteFont(String name)
        {
            SpriteFont spriteFont = null;

            foreach (NamedSpriteFont namedSpriteFont in
namedSpriteFontList)
            {
                if (name == namedSpriteFont.name)
                    spriteFont = namedSpriteFont.spriteFont;
            }

            return spriteFont;
        }

        //returns texture with given name
        public Texture2D getTexture2D(String name)
        {
            Texture2D texture2D = null;

            foreach (NamedTexture2D namedTexture2D in
namedTexture2DList)
            {
                if (name == namedTexture2D.name)
                    texture2D = namedTexture2D.texture2D;
            }

            return texture2D;
        }
    }
}
```

EnemyImageField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class EnemyImageField :ScreenComponent
    {
        List<Enemy> enemyList;
        Vector2 pivot;
        Texture2D healthBar;
        Rectangle healthBarRectangle, healthBarBackgroundRectangle;

        //Constructor
        public  EnemyImageField(String textureName, Drawer drawer,
List<Enemy> enemyList)
        {
            healthBar = drawer.getTexture2D("Health Bar");
            this.enemyList = enemyList;
            base.drawer = drawer;
            base.texture2D = base.drawer.getTexture2D(textureName);
            pivot = new Vector2(base.texture2D.Width / 2,
base.texture2D.Height / 2);
            destinationRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);
            sourceRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);

            healthBarBackgroundRectangle = new Rectangle(0, 0,
healthBar.Width, healthBar.Height / 2);
        }

        //Implements draw method for this component. Draws all
enemies in an enemyList.
        public override void draw(Drawer drawer)
        {
            for (int i = 0; i < enemyList.Count; i++ )
            {
                if (!enemyList[i].isInactive())
                {
                    if (enemyList[i] is Sheep)
                        setTexture("sheep");
                    else if (enemyList[i] is Cow)
                        setTexture("cow");

                    destinationRectangle.X =
(int)enemyList[i].getPosition().X;
                    destinationRectangle.Y =
(int)enemyList[i].getPosition().Y;
```

```
                    drawer.Draw(texture2D, destinationRectangle,
sourceRectangle, Color.White, enemyList[i].getRotation(), pivot,
SpriteEffects.None, layerIndex);
                }
            }
        }
    }
}
```

GamePlayField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class GamePlayField : ScreenComponent, MouseListener
    {
        public delegate void FiredEvent(object sender);
        public FiredEvent mouseClick;

        //Constructor
        public GamePlayField(String textureName, Drawer drawer)
        {
            base.texture2D = drawer.getTexture2D(textureName);
        }

        //Subscribe to mouse events
        public void activate()
        {
            gameScreen.getTDGame().inputManager.mouseMoved += new
InputManager.FiredEvent(onMouseMove);
            gameScreen.getTDGame().inputManager.mouseLeftPressed +=
new InputManager.FiredEvent(onMouseLeftPressed);
            gameScreen.getTDGame().inputManager.mouseLeftReleased +=
new InputManager.FiredEvent(onMouseLeftReleased);
        }

        //Unsubscribe from mouse events
        public void deActivate()
        {
            gameScreen.getTDGame().inputManager.mouseMoved -=
onMouseMove;
            gameScreen.getTDGame().inputManager.mouseLeftPressed -=
onMouseLeftPressed;
            gameScreen.getTDGame().inputManager.mouseLeftReleased -=
onMouseLeftReleased;
        }

        #region MouseListener Members
        public void onMouseMove(InputManager sender)
        {
            //throw new NotImplementedException();
        }

        //fires mouseClick event when left mouse button is pressed
        public void onMouseLeftPressed(InputManager sender)
        {
            if (mouseClick != null && new
Rectangle((int)sender.getMousePosition().X,
(int)sender.getMousePosition().Y, 1, 1).Intersects(new Rectangle(0,
0, 768, 768)))
                mouseClick(this);
        }

        public void onMouseLeftReleased(InputManager sender)
        {
            //throw new NotImplementedException();
        }

        public void onMouseRightPressed(InputManager sender)
        {
            throw new NotImplementedException();
        }

        public void onMouseRightReleased(InputManager sender)
        {
            throw new NotImplementedException();
        }

        #endregion


        public override void draw(Drawer drawer)
        {
        }
    }
}
```

GamePlayScreen.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class GamePlayScreen : GameScreen
    {
        TextField titleField, goldField;
        ImageField imageField, towerImage, rightPanel;
        EnemyImageField sheepImage;
        BulletImageField bulletImage;
        Button startButton, backButton, pauseButton,
dwarvenTowerButton, humanTowerButton, eladrinTowerButton;
        GamePlayField gamePlayField;
        MapField mapField;
        int selectedTowerType = -1;

        //Constructor to initialize Screen Components
        public GamePlayScreen(TDGame tDGame, Drawer drawer)
        {
            base.tDGame = tDGame;
            base.screenComponentList = new List<ScreenComponent>();
            base.drawer = drawer;

            gamePlayField = new GamePlayField("void", drawer);
            addScreenComponent(gamePlayField);

            goldField = new TextField("", drawer);
            addScreenComponent(goldField);
            goldField.setPosition(new Vector2(800, 20));

            dwarvenTowerButton = new Button("D", "Build Button",
drawer);
            dwarvenTowerButton.disableText();
            dwarvenTowerButton.setPosition(new Vector2(774, 200));
            addScreenComponent(dwarvenTowerButton);

            humanTowerButton = new Button("H", "Build Button 2",
drawer);
            humanTowerButton.disableText();
            humanTowerButton.setPosition(new Vector2(856, 200));
            addScreenComponent(humanTowerButton);

            eladrinTowerButton = new Button("E", "Build Button 3",
drawer);
            eladrinTowerButton.disableText();
            eladrinTowerButton.setPosition(new Vector2(938, 200));
            addScreenComponent(eladrinTowerButton);

            startButton = new Button("START!", "Fancy Button Right
Panel", drawer);
            startButton.setPosition(new Vector2(772, 666));
            addScreenComponent(startButton);

            backButton = new Button("Back", "Fancy Button 2",
drawer);
            backButton.setPosition(new Vector2(772, 732));
            addScreenComponent(backButton);

            pauseButton = new Button("Pause", "Fancy Button 2",
drawer);
            pauseButton.setPosition(new Vector2(897, 732));
            addScreenComponent(pauseButton);

            startButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            backButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            pauseButton.buttonClick += new
Button.FiredEvent(onButtonClick);
```

```
            dwarvenTowerButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            humanTowerButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            eladrinTowerButton.buttonClick += new
Button.FiredEvent(onButtonClick);

            imageField = new ImageField("Default Terrain", drawer);
            imageField.setLayerIndex(0.01f);
            addScreenComponent(imageField);

            rightPanel = new ImageField("Right Panel", drawer);
            rightPanel.setPosition(new Vector2(768, 0));
            rightPanel.setLayerIndex(0.05f);
            addScreenComponent(rightPanel);

            towerImage = new ImageField("tower", drawer);
            towerImage.setLayerIndex(0.2f);

            sheepImage = new EnemyImageField("cow", drawer,
tDGame.getGameLogic().getEnemyManager().getEnemyList());
            bulletImage = new BulletImageField("plasma", drawer,
tDGame.getGameLogic().getBulletManager().getBulletList());
            bulletImage.setLayerIndex(0.3f);
            bulletImage.setScale(new Vector2(0.5f, 0.5f));

            mapField = new MapField("roads", drawer,
tDGame.getGameLogic().getMap());
            mapField.setLayerIndex(0.02f);
            addScreenComponent(mapField);
        }

        //Draw method to draw all gameplay objects to the screen
(Enemies, bullets etc.)
        public new void draw()
        {
            base.draw();

            goldField.setText("Gold: " +
getTDGame().getGameLogic().getGold().ToString());

            sheepImage.draw(drawer);
            bulletImage.draw(drawer);

            foreach (Tower tower in
tDGame.getGameLogic().getTowerManager().getTowerList())
            {
                towerImage.setPosition(tower.getPosition());

                if (tower is DwarvenTower)
                    towerImage.setTexture("tower");
                if (tower is HumanTower)
                    towerImage.setTexture("tower 2");
                if (tower is EladrinTower)
                    towerImage.setTexture("tower 3");

                towerImage.draw(drawer);
            }
        }

        //On button click specifies actions to be taken
        public void onButtonClick(Button sender)
        {
            if (sender.getText() == "Back")
            {

gameScreenManager.changeGameScreen(GameScreenManager.GameScreenState.
mainMenuScreen);
                tDGame.getGameLogic().deActivate();
            }
            if (sender.getText() == "START!")
            {
                tDGame.getGameLogic().setToActiveMode();
            }
            if (sender.getText() == "Pause")
            {
            }
            if (sender.getText() == "D")
            {
                selectedTowerType = 0;
            }
            if (sender.getText() == "H")
            {
                selectedTowerType = 1;
            }
            if (sender.getText() == "E")
            {
                selectedTowerType = 2;
            }
        }

        //Activates all screen components in this screen
        public void activate()
        {
            base.activate();
            gamePlayField.activate();
        }

        //Deactivates all screen components in this screen
        public void deActivate()
        {
            base.deActivate();
            gamePlayField.deActivate();
        }

        //returns gamePlayField object
        public GamePlayField getGamePlayField()
        {
            return gamePlayField;
        }

        //returns selected tower type
        public int getSelectedTowerType()
        {
            return selectedTowerType;
        }

        //sets selected tower type
        public void setSelectedTowerType(int selectedTowerType)
        {
            this.selectedTowerType = selectedTowerType;
        }
```

```
    }
}
```

GameScreen.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public abstract class GameScreen
    {
        protected String name = "Game Screen";
        protected TDGame tDGame;
        protected List<ScreenComponent> screenComponentList;
        protected Drawer drawer;
        protected GameScreenManager gameScreenManager;

        //Adds screen component to the game screen
        public void addScreenComponent(ScreenComponent
screenComponent)
        {
            screenComponentList.Add(screenComponent);
            screenComponent.setGameScreen(this);
        }

        //returns tDGame object
        public TDGame getTDGame()
        {
            return tDGame;
        }

        //returns associated GameScreenManager
        public GameScreenManager getGameScreenManager()
        {
            return gameScreenManager;
        }

        //sets a new GameScreenManager
        public void setGameScreenManager(GameScreenManager
gameScreenManager)
        {
            this.gameScreenManager = gameScreenManager;
        }

        #region Drawable Members

        //Draws each ScreenComponent that is added to the screen
        public void draw()
        {
            foreach (ScreenComponent screenComponent in
screenComponentList)
                screenComponent.draw(drawer);
        }

        #endregion

        //Activates ScreenComponents
        public void activate()
        {
            foreach (ScreenComponent screenComponent in
screenComponentList)
                if (screenComponent is Button)
                    ((Button)screenComponent).activate();
        }

        //Deactivates ScreenComponents
        public void deActivate()
        {
            foreach (ScreenComponent screenComponent in
screenComponentList)
                if (screenComponent is Button)
                    ((Button)screenComponent).deActivate();
        }
    }
}
```

ImageField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class ImageField : ScreenComponent
    {
        //Constructor
        public ImageField(String textureName, Drawer drawer)
        {
            base.drawer = drawer;
            base.texture2D = base.drawer.getTexture2D(textureName);
            destinationRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);
            sourceRectangle = new Rectangle(0, 0,
base.texture2D.Width, base.texture2D.Height);
        }

        //Draws this ImageField to the screen
        public override void draw(Drawer drawer)
        {
            drawer.Draw(texture2D, destinationRectangle,
sourceRectangle, Color.White, 0.0f, Vector2.Zero, SpriteEffects.None,
layerIndex);
        }

        //Sets texture of this ImageField
        public void setTexture(String textureName)
        {
           base.texture2D = drawer.getTexture2D(textureName);
        }
    }
}
```

MainMenuScreen.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class MainMenuScreen : GameScreen
    {
        TextField titleField;
        ImageField backgroundImage;
        Button startGameButton, quickBattleButton, mapEditorButton,
optionsButton, creditsButton, exitButton;

        //Constructor to initialize ScreenComponents
        public MainMenuScreen(TDGame tDGame, Drawer drawer)
        {
            base.tDGame = tDGame;
            base.screenComponentList = new List<ScreenComponent>();
            base.drawer = drawer;

            backgroundImage = new ImageField("Default Terrain",
drawer);
            backgroundImage.setScale(new Vector2(1024 / 768.0f, 768 /
768));
            backgroundImage.setLayerIndex(0.01f);
            addScreenComponent(backgroundImage);

            startGameButton = new Button("Start New Game", "Fancy
Button", drawer);
            startGameButton.setPosition(new Vector2(384, 256));
            addScreenComponent(startGameButton);

            quickBattleButton = new Button("Quick Battle", "Fancy
Button", drawer);
            quickBattleButton.setPosition(new Vector2(384, 322));
            addScreenComponent(quickBattleButton);
            mapEditorButton = new Button("Map Editor", "Fancy
Button", drawer);
            mapEditorButton.setPosition(new Vector2(384, 388));
            addScreenComponent(mapEditorButton);

            optionsButton = new Button("Options", "Fancy Button",
drawer);
            optionsButton.setPosition(new Vector2(384, 454));
            addScreenComponent(optionsButton);

            creditsButton = new Button("Credits", "Fancy Button",
drawer);
            creditsButton.setPosition(new Vector2(384, 520));
            addScreenComponent(creditsButton);

            exitButton = new Button("Exit", "Fancy Button", drawer);
            exitButton.setPosition(new Vector2(384, 586));
            addScreenComponent(exitButton);

            startGameButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            quickBattleButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            mapEditorButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            optionsButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            creditsButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            exitButton.buttonClick += new
Button.FiredEvent(onButtonClick);
        }

        //On button Click specifies actions to be taken
        public void onButtonClick(Button sender)
        {
            if (sender.getText() == "Exit")
                tDGame.Exit();
            if (sender.getText() == "Options")

gameScreenManager.changeGameScreen(GameScreenManager.GameScreenState.
optionsScreen);
            if (sender.getText() == "Start New Game")
            {

gameScreenManager.changeGameScreen(GameScreenManager.GameScreenState.
gamePlayScreen);
                tDGame.getGameLogic().activate();
            }
        }

        //Activates ScreenComponents
        public void activate()
        {
            base.activate();
            quickBattleButton.deActivate();
        }
    }
}
```

MapField.cs
```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class MapField : ScreenComponent
    {
        Map map;

        //Constructor
        public  MapField(String textureName, Drawer drawer, Map map)
        {
            this.map = map;
            base.drawer = drawer;
            base.texture2D = base.drawer.getTexture2D(textureName);
            destinationRectangle = new Rectangle(0, 0,
base.texture2D.Width / 3, base.texture2D.Height / 2);
            sourceRectangle = new Rectangle(0, 0,
base.texture2D.Width / 3, base.texture2D.Height / 2);
        }

        //According to Map objects specifications, draws roads (takes
turning points into account etc.)
        public override void draw(Drawer drawer)
        {
            int roadType;

            for (int i = 0; i < map.getMapSize(); i++)
            {
                for (int j = 0; j < map.getMapSize(); j++)
                {
                    destinationRectangle.X = j * 96 -48;
                    destinationRectangle.Y = i * 96 - 48;
                    roadType = map.getMap()[i, j];

                    switch (roadType)
                    {
                        case (1):
                            {
                                sourceRectangle.X = 0;
                                sourceRectangle.Y = 0;
                            } break;

                        case (2):
                            {
                                sourceRectangle.X = 0;
                                sourceRectangle.Y = 96;
                            } break;
                        case (3):
                            {
                                sourceRectangle.X = 96;
                                sourceRectangle.Y = 0;
                            } break;
                        case (4):
                            {
                                sourceRectangle.X = 96;
                                sourceRectangle.Y = 96;
                            } break;
                        case (5):
                            {
                                sourceRectangle.X = 192;
                                sourceRectangle.Y = 0;
                            } break;
                        case (6):
                            {
                                sourceRectangle.X = 192;
                                sourceRectangle.Y = 96;
                            } break;
                    }

                    if (roadType != 0)
                    drawer.Draw(texture2D, destinationRectangle, 
sourceRectangle, Color.White, 0.0f, Vector2.Zero, SpriteEffects.None, 
layerIndex);
                }
            }
        }
    }
}
```

OpionsScreen.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class OptionsScreen : GameScreen
    {
        TextField titleField;
        ImageField backgroundImage;
        Button graphicsButton, soundButton, backButton;

        //Initializes ScreenComponents
        public OptionsScreen(TDGame tDGame, Drawer drawer)
        {
            base.tDGame = tDGame;
            base.screenComponentList = new List<ScreenComponent>();
            base.drawer = drawer;
            backgroundImage = new ImageField("Default Texture",
drawer);
            backgroundImage.setScale(new Vector2(1024 / 800.0f, 768 /
600.0f));
            backgroundImage.setLayerIndex(0.01f);
            addScreenComponent(backgroundImage);

            graphicsButton = new Button("Graphics", "Fancy Button",
drawer);
            graphicsButton.setPosition(new Vector2(384, 256));
            addScreenComponent(graphicsButton);

            soundButton = new Button("Sound", "Fancy Button",
drawer);
            soundButton.setPosition(new Vector2(384, 322));
            addScreenComponent(soundButton);

            backButton = new Button("Back", "Fancy Button", drawer);
            backButton.setPosition(new Vector2(2, 702));
            addScreenComponent(backButton);

            graphicsButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            soundButton.buttonClick += new
Button.FiredEvent(onButtonClick);
            backButton.buttonClick += new
Button.FiredEvent(onButtonClick);
        }

        //On button click specifies which actions to be taken
        public void onButtonClick(Button sender)
        {
            if (sender.getText() == "Back")

gameScreenManager.changeGameScreen(GameScreenManager.GameScreenState.
mainMenuScreen);
            if (sender.getText() == "Graphics")
                drawer.toggleFullScreen();
        }

        //Activates ScreenComponents
        public void activate()
        {
            base.activate();
        }

        //Deactivates ScreenComponents
        public void deActivate()
        {
            base.deActivate();
        }
    }
}
```

ScreenComponent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public abstract class ScreenComponent : Drawable
    {
        protected Rectangle destinationRectangle, sourceRectangle;
        protected Texture2D texture2D;
        protected GameScreen gameScreen;

        protected Drawer drawer;

        protected int textureIndex;
        protected float layerIndex = 0.1f;

        //sets layer index of this ScreenComponent
        public void setLayerIndex(float layerIndex)
        {
            if (layerIndex <= 1 && layerIndex >= 0)
                this.layerIndex = layerIndex;
        }

        //returns layer index of this component
        public float getLayerIndex()
        {
            return layerIndex;
        }

        //scales ScreenComponent with given vector
        public void setScale(Vector2 scale)
        {
            destinationRectangle.Width = (int)(scale.X *
destinationRectangle.Width);
            destinationRectangle.Height = (int)(scale.Y *
destinationRectangle.Height);
        }

        //sets position of this component with given Position
        public void setPosition(Vector2 position)
        {
            destinationRectangle.X = (int)position.X;
            destinationRectangle.Y = (int)position.Y;
        }

        //sets a new texture for this component
        public void setTexture(String textureName)
        {
            texture2D = drawer.getTexture2D(textureName);
            destinationRectangle = new Rectangle(0, 0,
texture2D.Width, texture2D.Height);
            sourceRectangle = new Rectangle(0, 0, texture2D.Width,
texture2D.Height);
        }

        //sets a new GameScreen for this component
        public void setGameScreen(GameScreen gameScreen) {
this.gameScreen = gameScreen; }

        #region Drawable Members

        //abstract method for drawing purposes
        public abstract void draw(Drawer drawer);

        #endregion
    }
}
```

TextField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class TextField : ScreenComponent
    {
        String textField;
        SpriteFont spriteFont;
        //Constructor
        public TextField(String textField, Drawer drawer)
        {
            this.textField = textField;
            spriteFont = drawer.getSpriteFont("Default Font");
        }

        //Draws textField to the screen
        public override void draw(Drawer drawer)
        {
            drawer.DrawString(spriteFont, textField, new
Vector2(destinationRectangle.X, destinationRectangle.Y), Color.White,
0.0f, Vector2.Zero, Vector2.One, SpriteEffects.None, layerIndex);
        }
        //Sets text of a textField
        public void setText(String text)
        {
            textField = text;
        }
    }
}
```