AETD - Arch-Enemy Tower Defense
Group 6

Olof Ol-Mårs
Erik Nordenhök
Felix Wallén
Johan Gustafson
Jonas Hellgren

Contents

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Scope . . . 4
- 1.3 Intended audience . . . 4
- 1.4 Version history . . . 4
- 1.5 Reference . . . 4
- 1.6 Glossary . . . 5
- 1.7 Summary . . . 7

2 System Overview **8**
- 2.1 General Description . . . 8
  - 2.1.1 Singleplayer mode . . . 8
  - 2.1.2 Multiplayer mode . . . 8
- 2.2 Overall Architecture Description . . . 9
- 2.3 Detailed Architecture . . . 10
  - 2.3.1 Dataflow for user input . . . 10
  - 2.3.2 Input manager control flow . . . 11
  - 2.3.3 Game engine control flow . . . 11
  - 2.3.4 Client network manager control flow . . . 12

3 Design considerations **13**
- 3.1 Assumptions and considerations . . . 13
- 3.2 General constraints . . . 13
  - 3.2.1 Hardware . . . 13
  - 3.2.2 Software . . . 13
  - 3.2.3 Network communication . . . 14

4 Graphical User Interface **15**
- 4.1 Overview . . . 15
  - 4.1.1 Flowchart . . . 18
- 4.2 Detailed descriptions . . . 19
  - 4.2.1 Main menu . . . 19
  - 4.2.2 Join game . . . 20
  - 4.2.3 Game-lobby . . . 21
  - 4.2.4 In-game . . . 23
  - 4.2.5 Settings menu . . . 26

5 Design Details **27**
5.1 Class Responsibility Collaboratior (CRC) Cards 27
5.2 Class Diagram 31
5.3 State Charts 32
5.4 Interaction Diagrams (Collaboration diagrams) 33
5.5 Detailed Design 37
5.5.1 Class: AETD 37
5.5.2 Class: Game 39
5.5.3 Class: Player 42
5.5.4 Class: Entity 43
5.5.5 Class: Projectile 44
5.5.6 Class: Button 45
5.5.7 Class: Tower 46
5.5.8 Class: Monster 49
5.5.9 Class: Graphics 51
5.5.10 Class: Chat 52
5.5.11 Class: Server 54
5.5.12 Class: Client 56
5.5.13 Class: InputParser 58
5.6 Package Diagram 59

6 Functional Test Cases **60**
6.0.1 Download the program 60
6.0.2 Install the program 61
6.0.3 Configure game settings 62
6.0.4 Play singleplayer game 63
6.0.5 Host multiplayer game 64
6.0.6 Join multiplayer game 65
6.0.7 Build Towers 66
6.0.8 Select a specific tower 67
6.0.9 Sell towers 68
6.0.10 Upgrade towers 69
6.0.11 View individual tower statistic 70
6.0.12 Monster levels 71
6.0.13 Monster movement 72
6.0.14 Kill monsters 73
6.0.15 Receive information about monster's health 74
6.0.16 Send monsters to other players 75
6.0.17 Chat with other players 76

1 Introduction

1.1 Purpose

The purpose of this document is to give the reader a good understanding of the design of the system. After reading this document, one should be able to implement the system as described in the Requirements Document.

1.2 Scope

This document contains descriptions of the classes and functions, and how they interact. It also contains information about how the interface will look like as well as test cases for the functionality to make sure the program will work as intended when completed

1.3 Intended audience

We expect the readers of this document to be those who are supposed to implement the system i.e. programmers. There is therefore necessary to have some programming knowledge to fully understand this document.

1.4 Version history

Date	Version	Summary
2008-01-28	0.1	Section 2.2 and 2.3 added.
2008-02-04	0.2	Section 4 added.
2008-02-11	0.3	Section 5.1 to 5.4 added.
2008-02-18	0.4	Section 5.5 and 5.6 added.
2008-02-25	0.5	Section 6 added.
2008-03-03	0.6	Section 1 added.
2008-03-05	0.7	Section 3 added.
2008-03-06	0.8	Section 2.1 added.
2008-03-10	1.0	First version of the document compiled.

1.5 Reference

A document that is related to this is the Requirements Document (version 1.0). If there is references to a RD, it is this Requirements Document that is to be referenced to.

The documents that result from this document is the Implementation Plan (IP) and the Final Documentation (FD).

1.6 Glossary

- **Actor**

  Someone or something that interacts in a use case.

- **AETD**

  Arch-Enemy Tower Defense is the name of our game.

- **AI**

  Artificial Intelligence, a computer system that acts as if it could think.

- **Blocking**

  A tower is blocking if it is placed in such a way that monsters can find no path between their current position and the goal.

- **C++**

  A programming language that we will use to build our system.

- **Client**

  One of the players in a multiplayer game that connects, rather than create, the game.

- **Client-computer**

  The computer of a client player.

- **Compiler**

  When compiling the computer translates the written code to a language that the computer can read.

- **Executable file**

  A program that can be executed on a computer.

- **Functional requirement**

  A function that will be included in the final product.

- **GUI**

  Graphical User Interface is the graphical interface that the system presents to the user.

- **Host**

  One of the players in a multiplayer game that creates, rather than connects to, a game. There is only one host for every multiplayer game.

- **Host-computer**

  The computer of the host player.

- **IP-address**

  An Internet Protocol-address is used for connecting computers over network. Much like how telephone numbers connects telephones.

- **Monster**

  Object moving from one side of the playing field towards the other.

- **OpenGL**

  Open Graphics Library is a free source graphical library that contains functions to implement a graphic environment.

- **Playing field**

  An area individual to each player where towers can be built and monsters move.

- **Socket**

  A socket is a communication end-point unique to a machine communicating on an Internet Protocol-based network.

- **Tower**

  A stationary object placed on a playing field that shoots at monsters in range.

- **UDP**

  User Datagram Protocol, a specific set of rules used in networking.

- **Use Case**

  A step-by-step description of a function of the system.

- **Web-server**

  A storage space for files on the Internet. It allows users to download data from the server.

1.7 Summary

The design document is a guide for a programmer to follow when constructing the game. It starts off with a general overview of the system and issues to be taken into consideration. It also contains descriptions of the graphical elements of the game as well as what functions they relate to. All classes in the game are described in detail, including major methods for each class and how the classes interact with each other. Finally there are test cases of how to test the functionality from the RD.

2 System Overview

2.1 General Description

Arch-Enemy Tower Defense is a strategic game which supports both singleplayer and multiplayer features. The general idea of the game is to stop a group of monsters from crossing a certain area and reaching their goal. For each monsters that manages to cross the area the player looses points and when a certain amount of monsters have passed the player has been defeated.

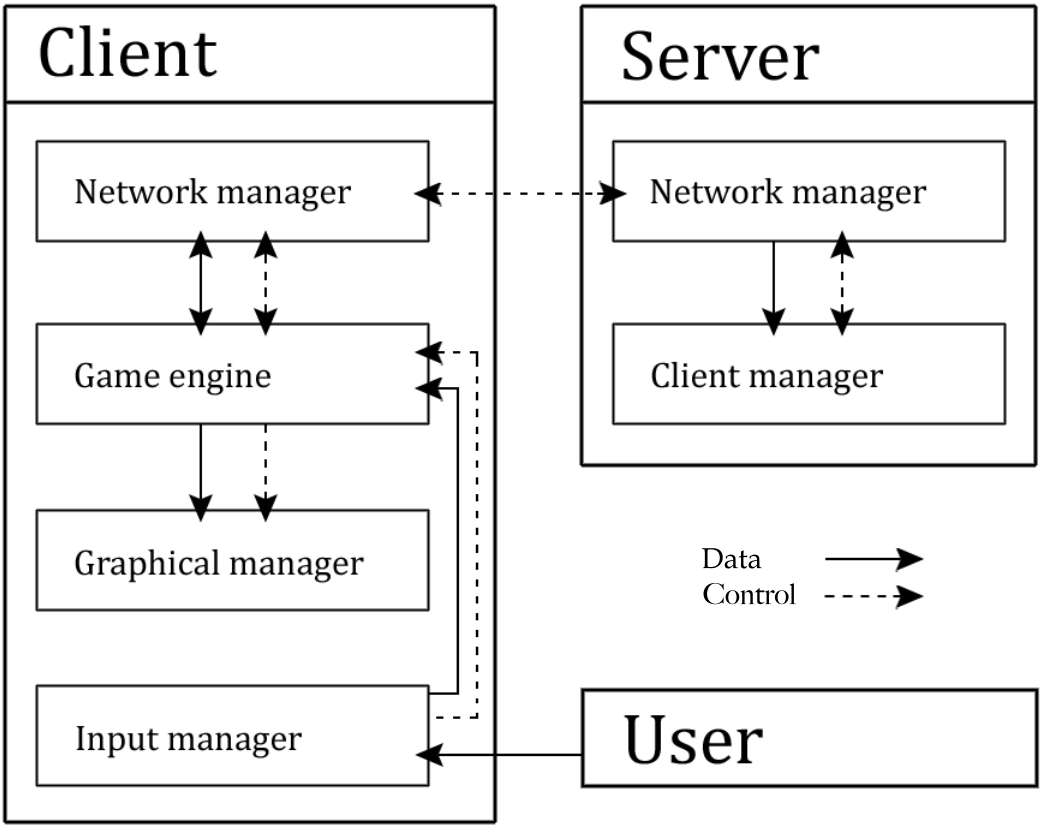
2.1.1 Singleplayer mode

In singleplayer mode the player tries to survive for as long as possible with monsters arriving at set intervals. With the gold earned from killing a monster the player can choose to build new towers or upgrade existing towers and thereby improving his defensive capabilities. When playing singleplayer mode the user has no need for an Internet connection.

2.1.2 Multiplayer mode

In the multiplayer mode two or more players are playing against each other. Players can see certain information concerning the progress of their opponents. They can then choose to spend their gold either on building and upgrading their towers, as in the singleplayer mode, or on sending monsters to their opponents. One of the players in a multiplayer game will be hosting by choosing the host option from AETDs main menu. This will allow other players to connect by choosing the join network game option and then specifying the IP-address of the host computer. In this mode players can also chat with each other.

2.2 Overall Architecture Description



Client

The clients job is to handle the game data for the particular user associated with it and to provide a graphical interface for said user. The Input manager takes input from the user and sends data to the Game engine. The Game engine then process the data and sends it to both the Graphical manager and the Network manager. The Graphical manager updates the graphical interface from the given data. The Network manager sends the data to the Network manager in the server.

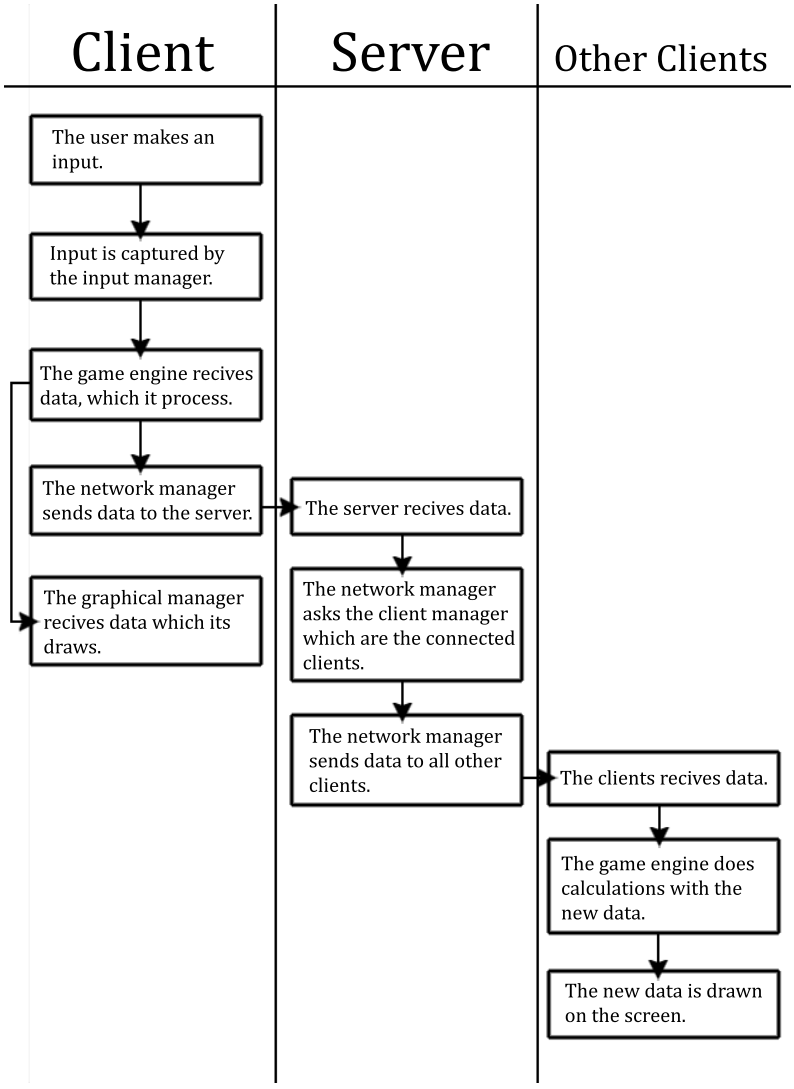
Server

The servers job is to manage the clients that are connected to it and to distribute information received from the individual clients to all others. The

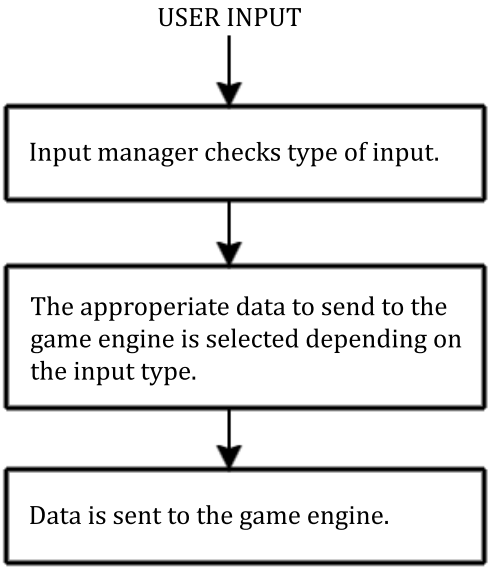
communication between the Network manager in the clients and the one in the server is made with the UDP protocol.

2.3 Detailed Architecture

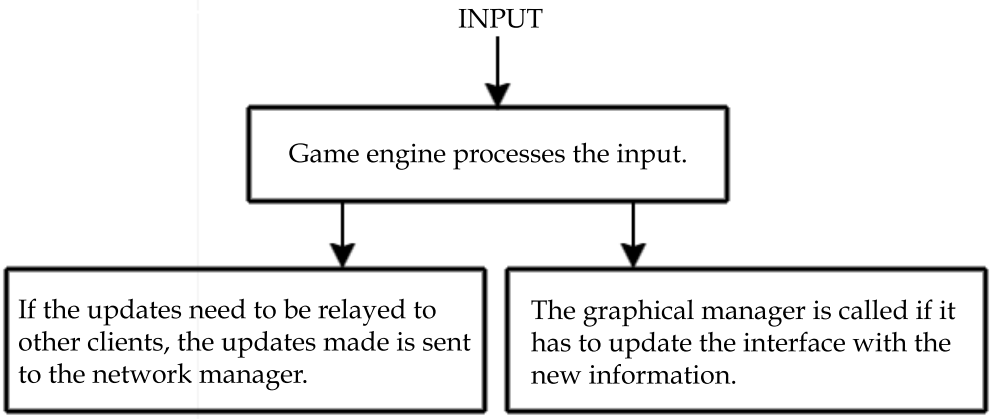
2.3.1 Dataflow for user input



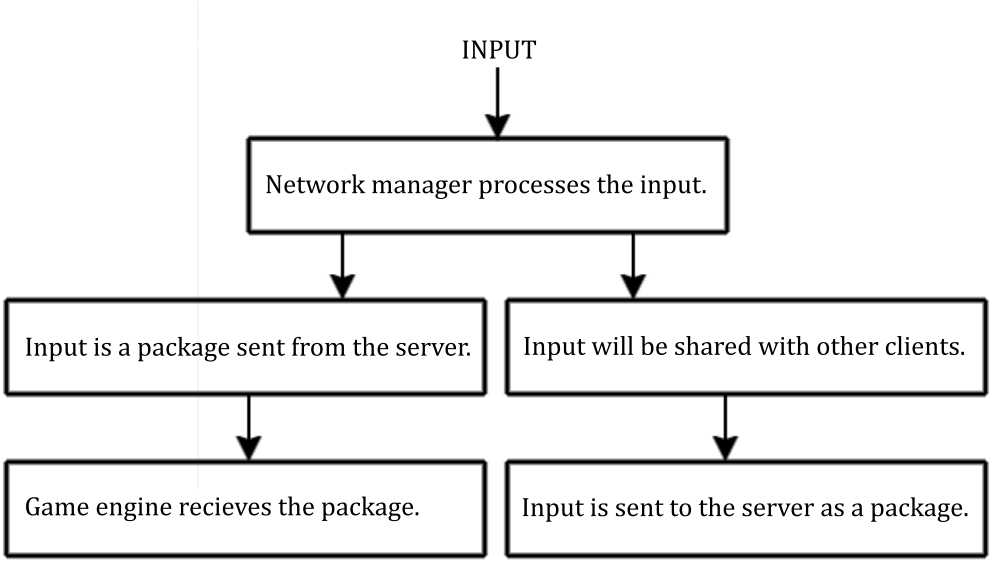
2.3.2 Input manager control flow



2.3.3 Game engine control flow



2.3.4 Client network manager control flow



3 Design considerations

3.1 Assumptions and considerations

When designing our system we have made several assumptions considering the specification of the hardware, software and operating system that the game is to run on. These assumptions have been made to facilitate the testing of our system but when designing the system we have made choices that opens up for a wider interoperability in the future.

Since the majority in our target user audience, users who often use their computers for entertainment such as games like ours, are using the Microsoft Windows XP operating systems, we have aimed to make our system work properly on Microsoft Windows XP. As such we assume that the end user will run Windows XP and do not guarantee any kind of performance on other operating systems.

In order to use the multiplayer functionality in our game the user might need some basic knowledge about the configuration of his firewall and anti-virus software in case it somehow interferes with the connection between hosts and clients.

3.2 General constraints

3.2.1 Hardware

Our system should be fully functional on a system with the following hardware setup or something with equal or greater performance.

- A processor operating at 1.4GHz.
- A total memory capacity of 512MB.
- A hard drive space of 50MB.
- A shared video memory of 16MB.

3.2.2 Software

Since the system is written in C++ with graphics rendered in OpenGL we can include all the files needed to avoid software constraints on the user.

3.2.3 Network communication

When using the multiplayer functionality the user should have a minimum network speed of 1Mbit/s for the game to run smoothly. We are going to use the UDP protocol when communicating between host and clients.

4 Graphical User Interface

4.1 Overview

Figure 1: Main menu

When the game is started this screen will be presented to the user. It contains buttons leading to the game itself and the settings menu.

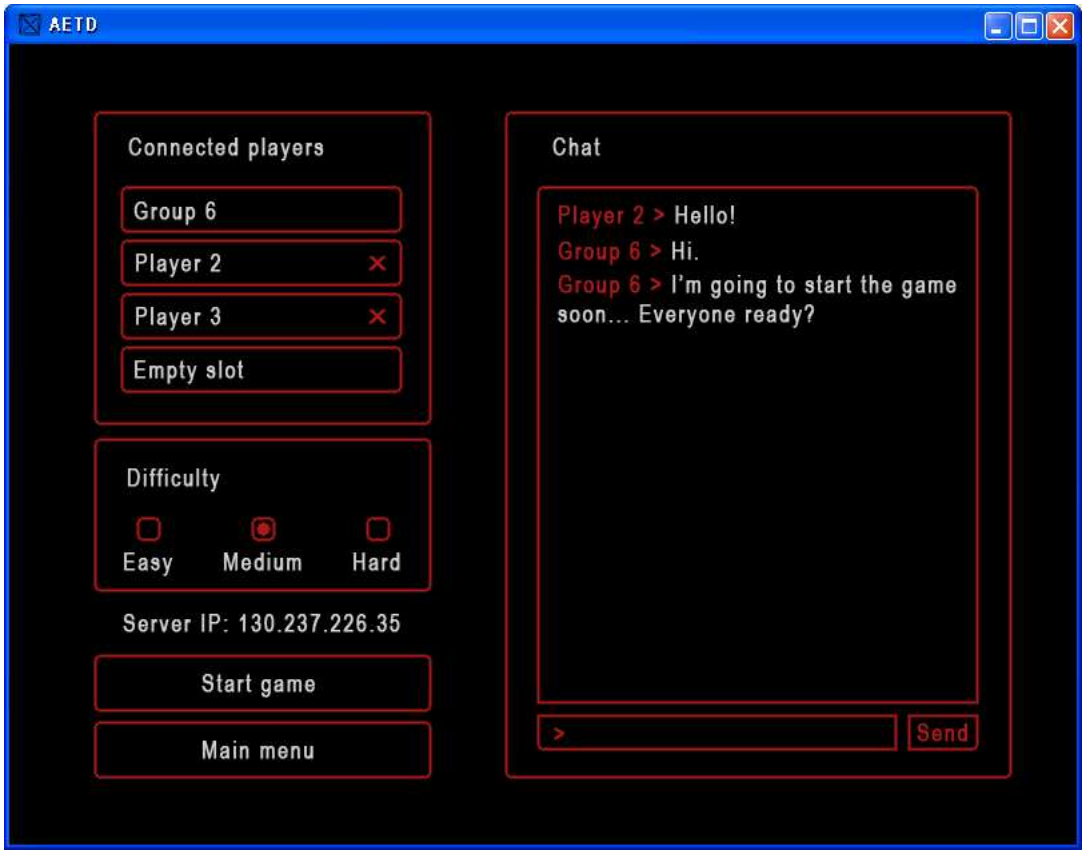


Figure 2: Wait screen of the Network game

When a user enters a network game (either hosts or joins one) he will be placed in the game-lobby. Here he can see who else is connected to the game and chat with the other connected players.

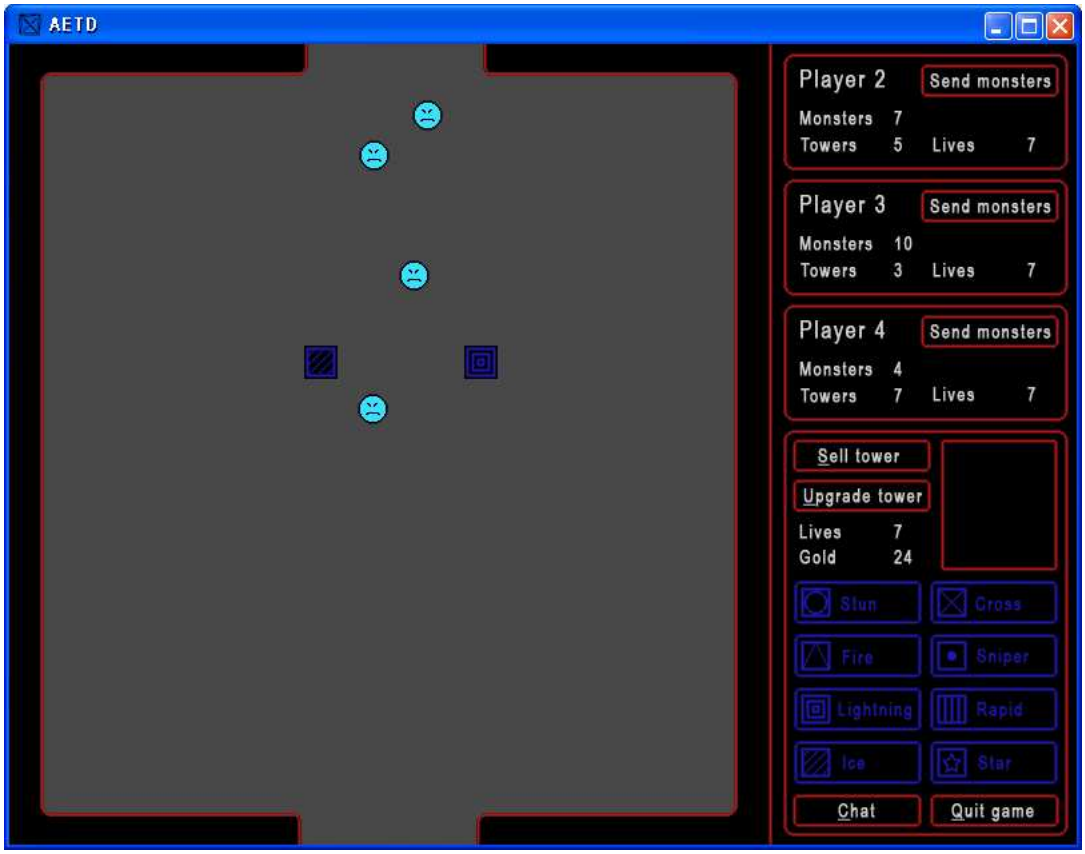


Figure 3: In-game screenshot

The main graphical user interface (GUI) when playing the game looks like the image above. It's divided into the playing field to the left and opponent information and tower related functions to the right.

It's on the playing field where all the action takes place. Monsters appear at the top and try to reach the bottom. The user builds towers on the field in order to hinder the monsters from reaching their goal.

The opponent information tells the user how his opponents are doing. With the information given the user can decide if he wants to send monsters.

In the tower related functions, one can build, upgrade and sell towers as well as see information about towers or monsters. This part of the GUI also gives functions for chatting and quiting the game.

4.1.1 Flowchart

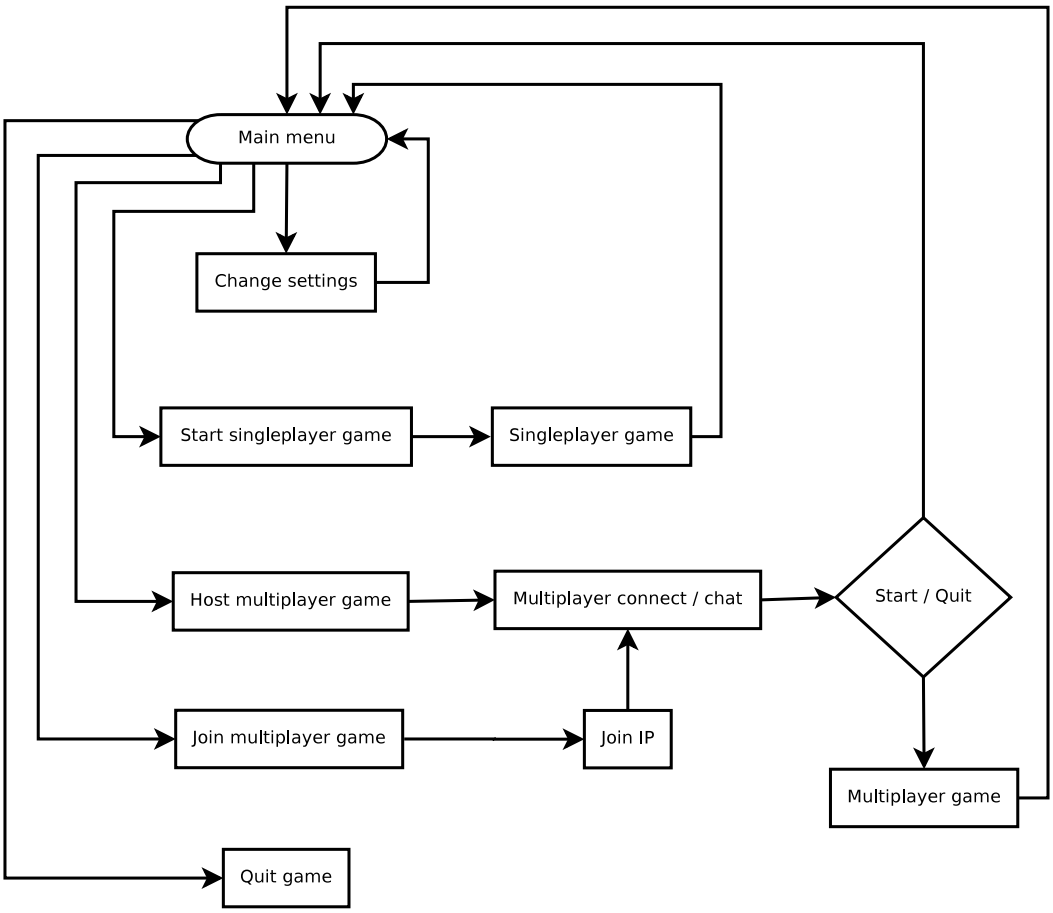


Figure 4: Flowchart

This flowchart describes the order in which the images should be interpreted. It also tells the order that menus and events will occur and how to get to a certain point.

4.2 Detailed descriptions

4.2.1 Main menu

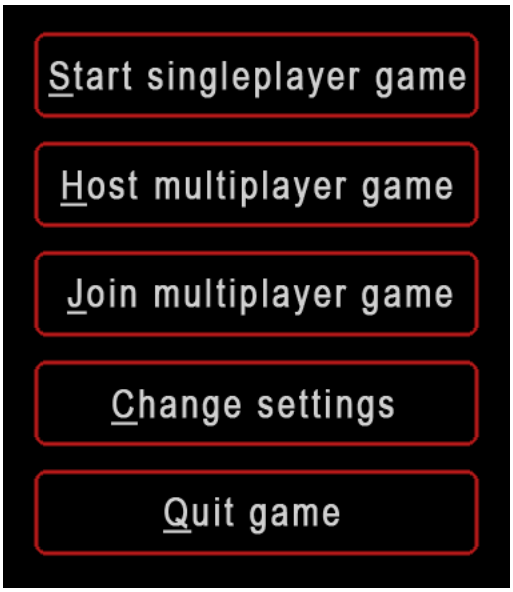


Figure 5: The menu-buttons of the Main menu

- The "Start singleplayer game" button starts a singleplayer game.
- The "Host multiplayer game" button starts up a server in the background as well as sends the user to the game-lobby as a host.
- The "Join multiplayer game" button sends the user to the Join game menu.
- The "Change settings" button sends the user to the settings menu.
- The "Quit game" button closes the game.

All buttons above can be initiated either by left-clicking them with the mouse or typing the underlined letter for the keyboard-shortcut.

This image implements the functional requirements:

- Play singleplayer game
- Host multiplayer game
- Join multiplayer game
- Configure game settings

4.2.2 Join game

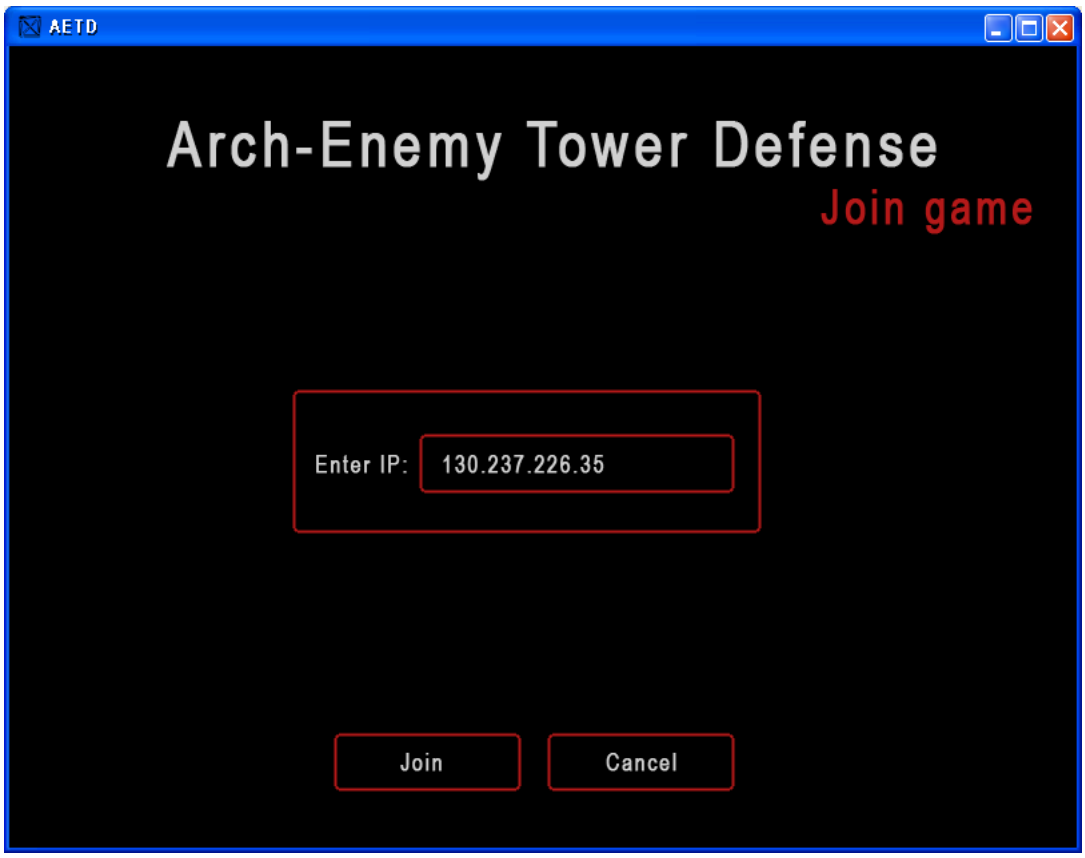


Figure 6: Enter the server IP to join a Network game

The join game menu has a prompt in the middle of the screen that prompts the user for an IP-address to a server. Once the IP of a server is entered and the user left-clicks the "Join" button he is sent to the game-lobby as a client. If the user wishes to go back to the main menu he can do so by left-clicking the cancel button.

This image implements the functional requirements:

- Join multiplayer game.

4.2.3 Game-lobby

Figure 7: The players connected before the Multiplayer start

At the top left section of the game-lobby there will be a frame containing the names of the connected players. For the host user there will be a button next to the names which he can left-click to kick a player from the game.

Figure 8: The Multiplayer difficulty

Below the connected players there will be a frame showing the difficulty of the game. The speed and life of the monsters vary with the game-difficulty. To change the difficulty setting the user left-clicks the box relating to his choice. The selected difficulty box will then be filled. Only the host can change the difficulty.

Figure 9: Server IP and Start multiplayer game / quit to Main menu

Below the difficulty the IP-address of the server is shown, this is useful for letting others know what server to connect to. To start the game, the

host user must left-click the “Start game” button. If a user wants to leave the multiplayer session he can do so by left-clicking the “Main menu” button.

Figure 10: Multiplayer wait-chat

To the right there is a frame for chatting with the other connected players. To send a message the user types in the chat-prompt at the bottom of the screen and messages received will be displayed above.

This image implements the functional requirements:

- Chat with other players.

4.2.4 In-game

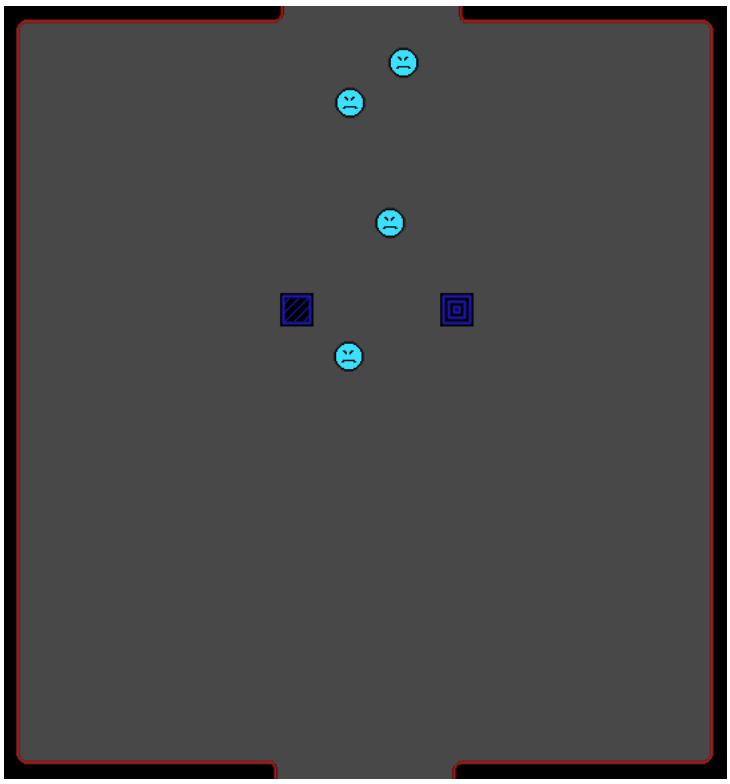


Figure 11: The playing field, with two towers and four monsters

The monsters enters the playing field through the gate at the top and will start moving towards their goal at the gate in the bottom. Monsters are represented in the picture with the smileys. Monsters will enter the playing field at a set interval maintained by the game, as well as when opponents send monsters to the player. The towers, represented by the squares with symbols, are built on the playing field by the player. When monsters enter the range of towers, the towers will automatically attack them. This will be represented by a small projectile leaving the tower towards the monster.

This image implements the functional requirements:

- Select a specific monster
- Select a specific tower
- Monster movement
- Kill monsters

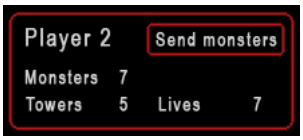


Figure 12: Multiplayer status bar

This section shows information about the opponents, that is the other players in the same game. For each other player there will be a separate field containing his name, number of monsters and towers currently on his playing field, number of lives the he has left and a button to send monsters to him. These values are updated via the network when the values change for the opponent. When the player left-clicks the "Send monsters" button, additional monsters will appear for the opponent and gold will be debited for the player.

This image implements the functional requirements:

- Send monsters to other players.

Figure 13: Player status bar and build tower-menu

This section contains the different towers that the player can build. To build a tower the user must left-click the button representing the tower from the menu. The user then left-clicks on the playing field where he wants the tower to be built. To use the "Sell tower" and "Upgrade tower" buttons, the user must first left-click a tower already built on the playing field. He can then left-click the button to preform the task he wanted. At the bottom of the screen there are buttons for initiating player chat and quitting the game.

This image implements the functional requirements:

- Sell towers
- Upgrade towers
- Build towers

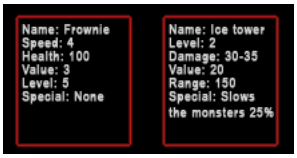


Figure 14: Monster and Tower statistics

When a tower is selected, either from the build menu or on the playing field, the information box will show information relating to that tower. The box will contain information about the name, level, value, range, damage and special abilities for the selected tower. If a monster is selected instead of a tower the box will show information relating to that specific monster, that is its name, speed, health, value and special abilities.

This image implements the functional requirements:

- Receive information about monsters health
- View individual tower statistics

Figure 15: The in-game multiplayer chat

When a user left-clicks the chat button or uses the keyboard-shortcut, a chat-prompt will appear at the bottom of the playing field allowing the user to send a text message to his opponents. When a message is entered the user can left-click the "Send" button or use the keyboard-shortcut to send the message. When the user receives a message it will be shown above the prompt.

This image implements the functional requirements:

- Chat with other players.

Figure 16: The Settings menu

4.2.5 Settings menu

This screen shows the settings that a user can change for the game. The different difficulties for a singleplayer game can be use to determine the speed and health of monsters, this does not affect multiplayer games. To change the difficulty setting the user left-clicks the box relating to his choice. The selected difficulty box will then be filled. Below this there will be a field where the player can edit the name that will be shown as his name when playing a multiplayer game. To save any changes made, the user left-clicks on the "Save" button. To return to the main menu without saving, the user can left-click the "Cancel" button.

This image implements the functional requirements:

- Configure game settings.

5 Design Details

5.1 Class Responsibility Collaboratior (CRC) Cards

Class: AETD	
Handle main menus. Configure player settings. Start server. Join specific server. Launch game.	Game Server Graphics

Class: Server	
Synchronize information with clients. Control IP address. Control port number.	Client Game Player Chat

Class: Tower	
Constructs projectiles. Block monster path. Upgrade capabilities. Control fire rate. Select type of projectile.	Entity Graphics Projectile

Class: Game	
Initialize game board. Initialize towers and monsters. Count gold. Countdown to next wave.	Client Server Entity Tower Monster Chat

Class: Monster	
Calculates path to exit. Trigger graphics engine to draw object. Speed. Hitpoints. Special ability.	Entity Graphics

Class: Projectile	
Calculates path to monster. Trigger graphics engine to draw object. Speed. Damage. Special ability.	Entity Graphics

Class: Entity	
Defines functions for towers, monsters and other visible objects. Draw entity. Remove entity.	Graphics

Class: Graphics	
Draw visible objects.	Entity Game AETD

Class: Chat	
Display messages. Send new messages to other players.	Graphics Client Server

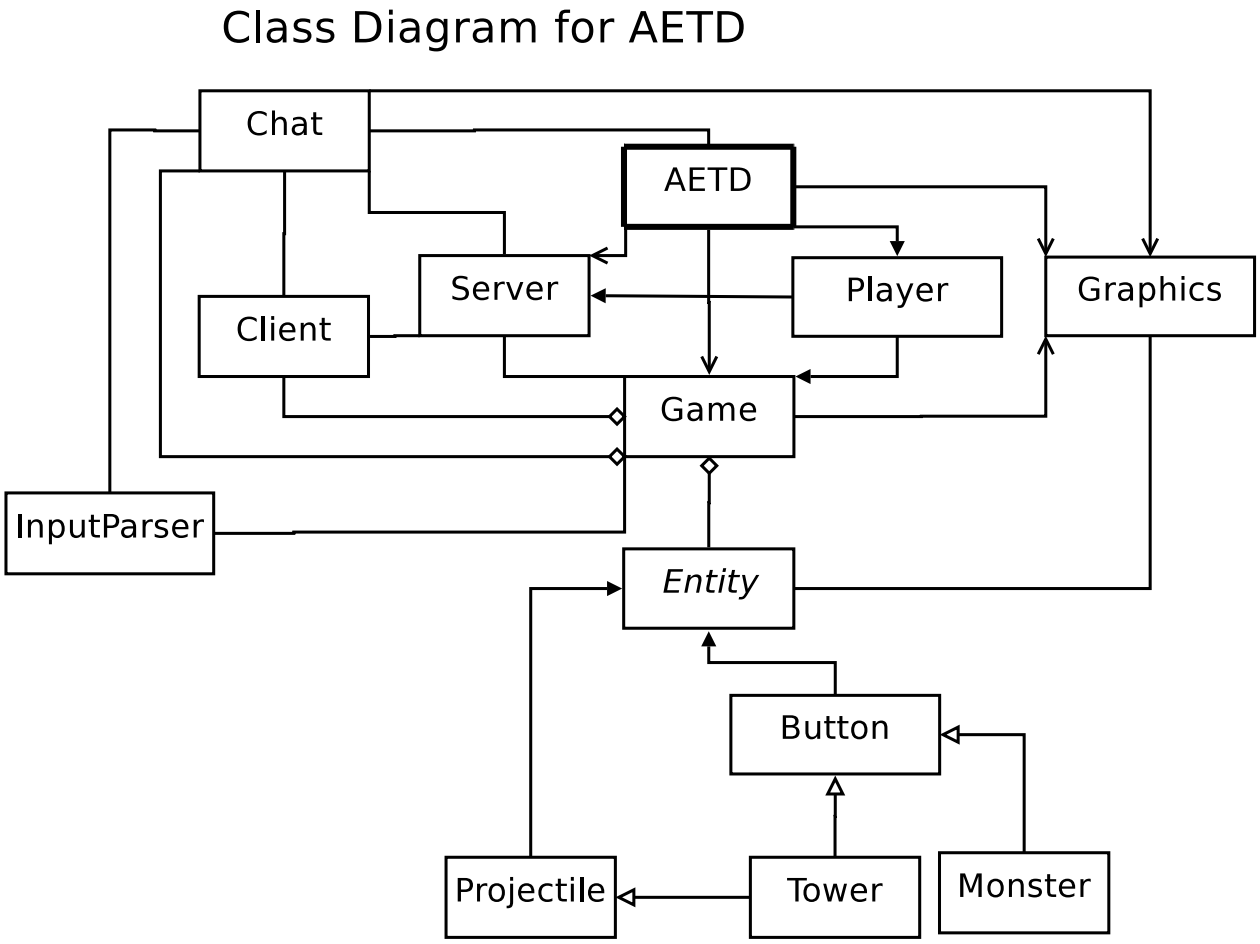
Class: Client	
Communicate information to host computer. Control IP address. Control port number.	Server Game AETD

Class: Player	
Number of monsters. Number of towers. Number of lives. Amount of gold. Control player name. Control player id.	AETD Game Server

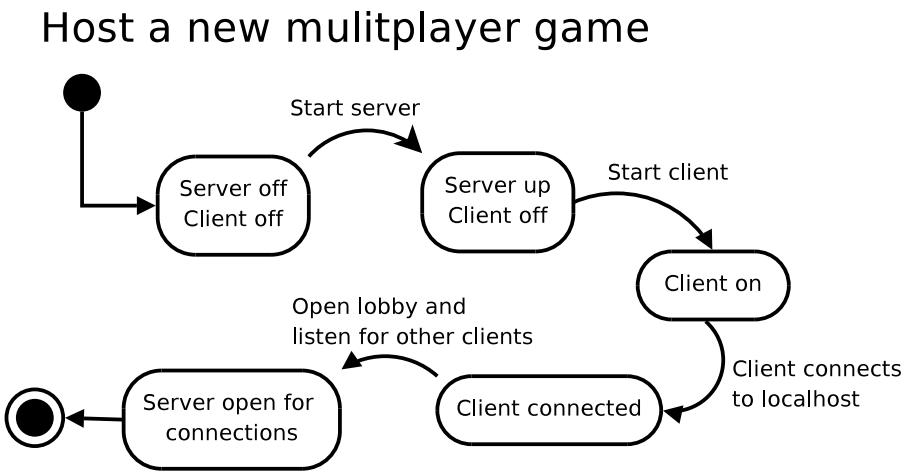
Class: Button	
Connnect visual button to a function in another class.	AETD Game Graphics Entity

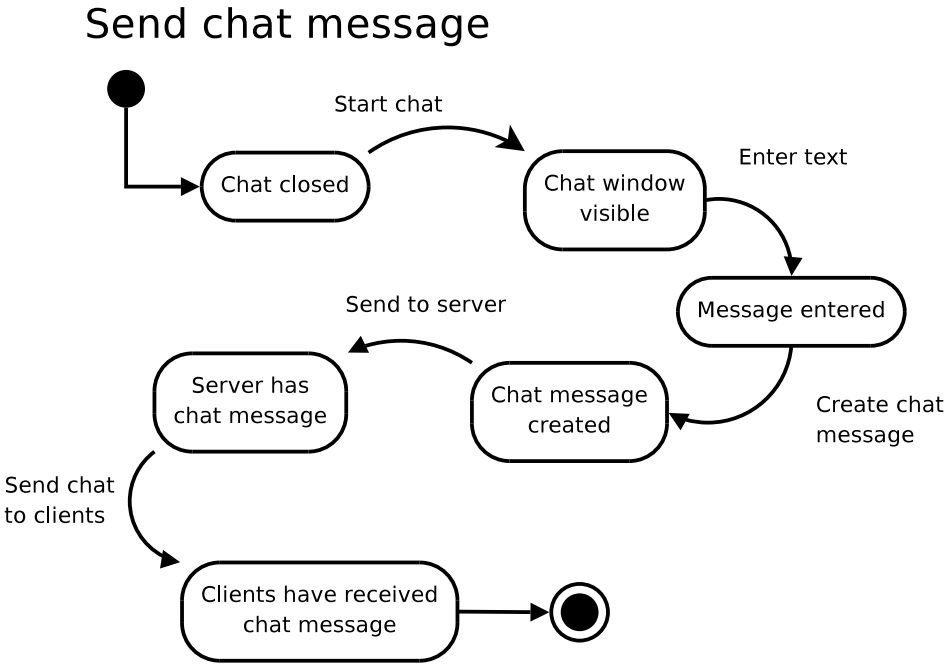
Class: InputParser	
Listens for user input through keyboard and mouse. Parse commands and send to designated class.	AETD Game Chat

5.2 Class Diagram



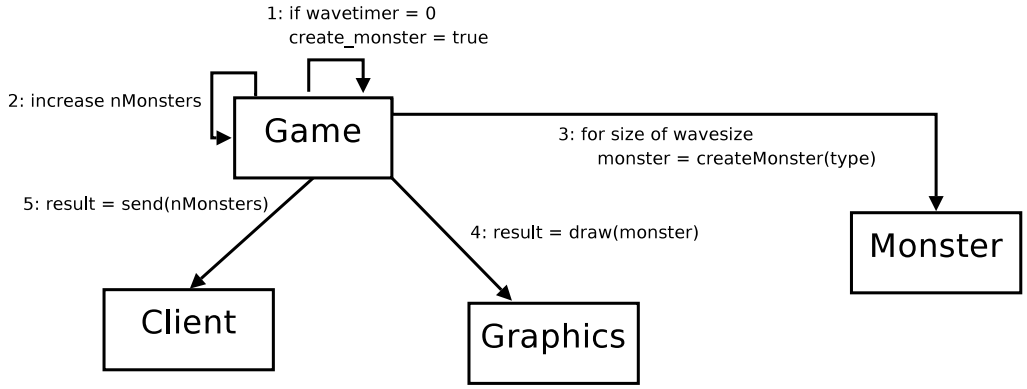
5.3 State Charts



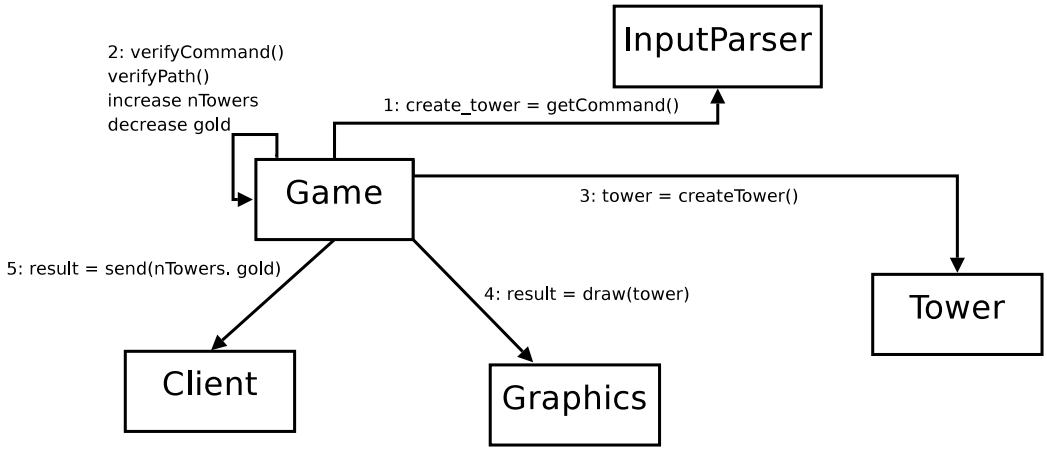


5.4 Interaction Diagrams (Collaboration diagrams)

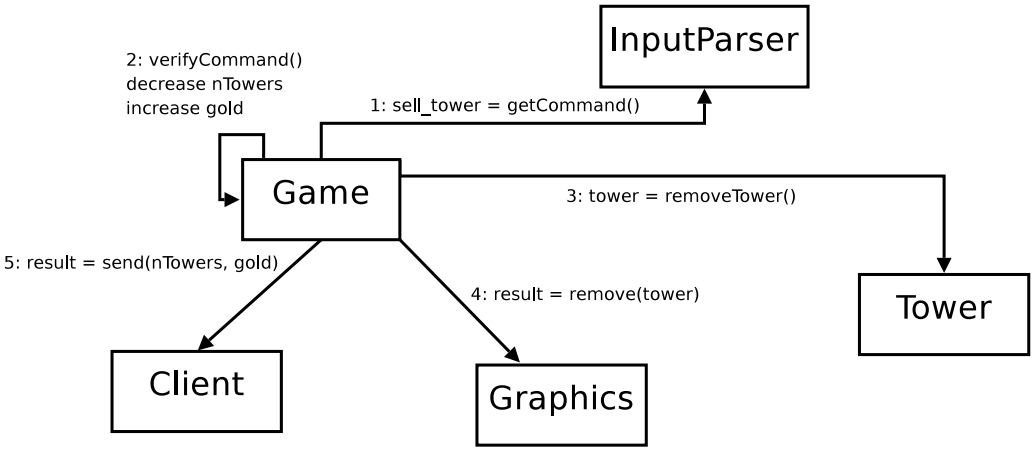
Create monster collaboration



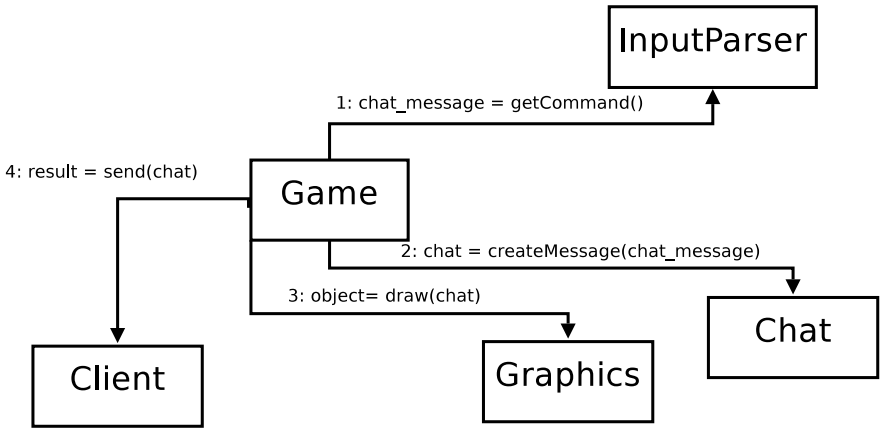
Create tower collaboration



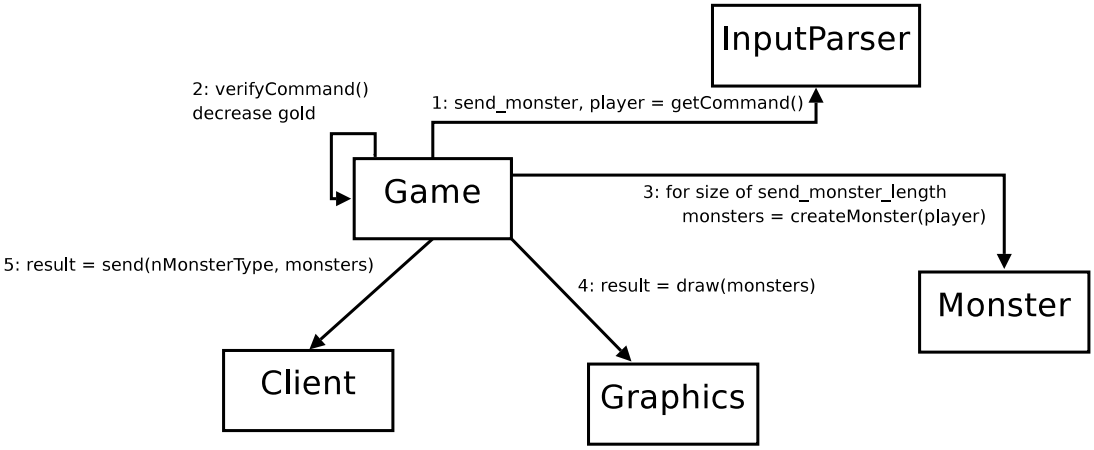
Sell tower collaboration



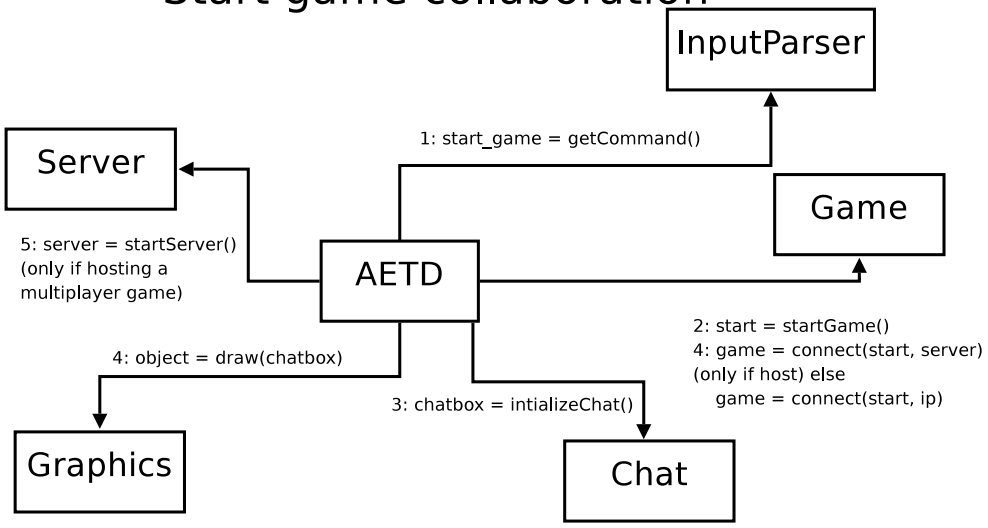
Send chat message collaboration



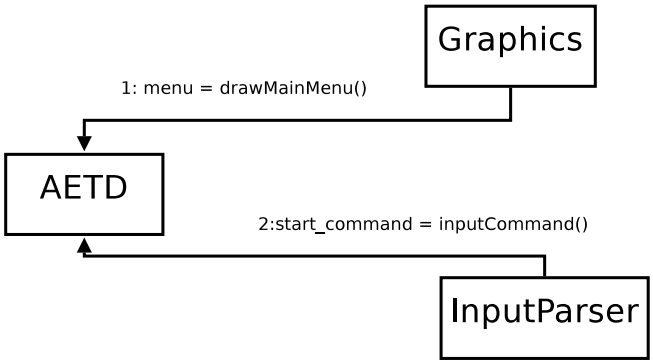
Send monsters collaboration



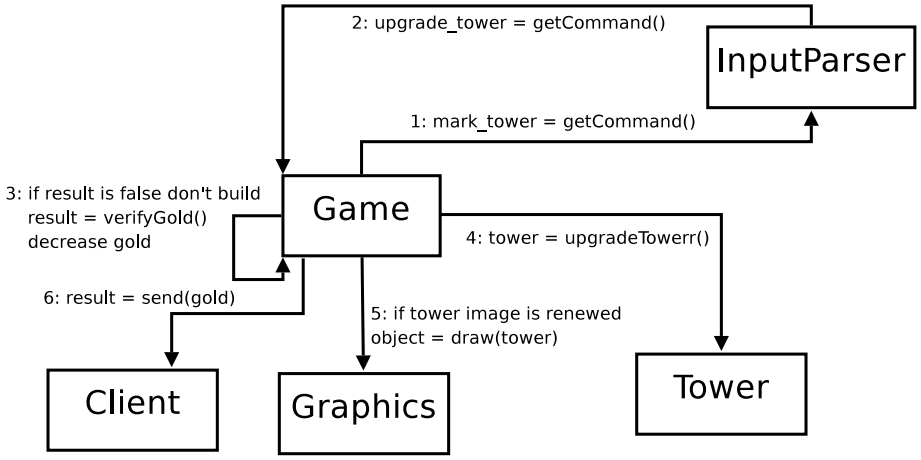
Start game collaboration



Startup collaboration



Upgrade tower collaboration



5.5 Detailed Design

All variables are written in a C++ syntax. That means type first and then the name of the variable. Get/Set functions for variables will be named getVariableName or setVariableName. Variables will be named variable_name.

5.5.1 Class: AETD

Variables

vector<vector <Button>> menu
int current_frame
Graphics graphic_manager
Game game
InputParser input_manager
int difficulty
String player_name
Client client
Server server

Functions

mainLoop
The main loop for AETD. Iterates over the menu currently showing to decide what button that has been pressed.

- Input: none.
- Called by: starting the program.
- Validity checks: Checks if there is a connection in Client. Does nothing in client if not. Checks if there is a connection in Server. Does nothing in server if not.
- Calls: sendTo and recvFrom in Client and in Server and drawMenu.
- Return value: none.

showSettingsMenu
Sets current_frame to settings menu.

- Input: none.
- Called by: Button.

- Return value: none.

showNetworkLobbyMenu
Sets current_frame to network lobby menu.

- Input: none.
- Called by: Button.
- Validity checks: Checks if a server should be setup. Ignores server functions if not.
- Accesses: Client client and Server server.
- Calls: socket and bind in Client and Server.
- Return value: none.

showNetworkJoinMenu
Sets current_frame to network join menu.

- Input: none.
- Called by: Button.
- Return value: none.

drawMenu
Draws the vector<Button> at current_frame in the menu vector.

- Input: none.
- Called by: mainLoop.
- Validity check: If chat is to be displayed, printHistory is called.
- Accesses: vector<vector <Button>> menu.
- Calls: All functions in graphics manager and printHistory in chat.
- Return value: none.

5.5.2 Class: Game

Variables

vector<Tower> towers
vector<Monster> monsters
vector<Projectile> projectiles
vector<Button> buttons
vector<Player> players
Button current_target

Functions

mainLoop
Calls the different update and draw functions in game.

- Input: none.
- Called by: AETD.
- Calls: printHistory in chat and moveMonsters, updateTowers, updatePlayers, draw and drawDetails.
- Return value: none.

moveMonsters
Iterates over all monsters and calls monster functions.

- Input: none.
- Called by: mainLoop.
- Accesses: vector<Monster> monsters and Player.
- Calls: moveMonster and getDieFlag in Monster and setGold and getGold in Player.
- Post-condition: Removes the monster if getDieFlag is not 0. Gives the player gold equal to die_flag.
- Return value: none.

updateTowers
Iterates over all towers and calls the different tower related functions.

- Input: none.
- Called by: mainLoop.
- Accesses: vector<Tower> towers and Player.
- Calls: attackMonster, getSellFlag in Tower and setGold and getGold in Player.
- Post-condition: Removes the tower if getSellFlag is not 0. Gives the player gold equal to sell_flag.
- Return value: none.

updatePlayers
Checks client for new updates relating to players. If changes to the local player has happened set functions in client is called.

- Input: none.
- Called by: mainLoop.
- Validity checks: Checks that the data from the method getInData is information relating to number of monsters, lives and towers. Ignores the data otherwise.
- Accesses: vector<Player> players, Client client in AETD
- Calls: setOutData,getInData in Client and setLives, getLives, setTowers, getTowers, setMonsters and getMonsters in Player and getClient in AETD.
- Return value: none.

draw
Calls graphic with the interface and all entities it has to draw.

- Input: none.
- Called by: mainLoop.

- Accesses: Graphics graphics, vector<Tower> towers, vector<Monster> monsters, vector<Projectile> projectiles, vector<Button> buttons and vector<Player> players.
- Calls: drawEntity in Graphics with the entities from the vectors and getLives, getTowers, getMonsters for each player.
- Return value: none.

drawDetail

Calls graphic with data from the current selected target to draw the detailed information box.

- Input: none.
- Called by: mainLoop.
- Validity checks: Checks if Button current_target is a tower. If not it will ignore the tower functions. Checks if Button current_target is a monster. If not it will ignore the monster functions.
- Accesses: Graphics graphics, vector<Tower> towers and vector<Monster> monsters.
- Calls: printString in Graphics and getName, getRate, getLevel, getDamage from Tower and getName, getHp, getSpeed, getLevel from Monster.
- Return value: none.

5.5.3 Class: Player

Variables

int monsters
int towers
int lives
int gold
String name
int playerID

Functions

Get functions for all variables
Returns the different variables.

- Input: none.
- Called by: Game.
- Return value: the variable value.

Set functions for all variables
Set the different variables.

- Input: the variable value.
- Called by: Game.
- Return value: none.

5.5.4 Class: Entity

Variables

String image_name
Coordinate pos

Functions

Get functions for all variables
Returns the different variables.

- Input: none.
- Called by: Game.
- Return value: the variable value.

5.5.5 Class: Projectile

Variables

Monster target
float speed

Functions

moveProjectile
Moves the projectile towards the target at the speed set by float speed.

- Input: none.
- Called by: moveProjectiles in Tower.
- Accesses: Monster.
- Calls getHp, getPos and setDisplayHp in Monster.
- Post-condition: When the position of the projectile is equal to the position of the target it will set the display_hp of the target to its hp. If hp is less than or equal to zero it will set the die_flag to value.
- Return value: none.

5.5.6 Class: Button

Buttons will contain a pointer to the function that will be executed when the button is clicked. The function will be specified in the constructor. Buttons inherit from Entity.

Variables

functionPointer function_pointer

Functions

buttonFunction()
Calls the function associated with the button.

- Input: none.
- Called by: AETD and Game.
- Calls: the function specified by function_pointer.
- Return value: none.

5.5.7 Class: Tower

Tower inherits from Button and every tower-type will have its own subclass inheriting from the tower class that contains specialized methods for the specific tower-type.

Variables

float range
float damage
int rate
int level
int time_to_fire
int value
int sell_flag
Monster target
vector<Projectile> projectiles

Functions

findMonster
Iterates over the monsters currently on the playing-field to find any monster in range.

- Input: none.
- Called by: attackMonster.
- Accesses: vector<Monster> monsters in Game and Monster target.
- Calls: getPos in Monster.
- Post-condition: The first monster to be found in range will be set to target.
- Return value: True if a monster is found in range.

attackMonster
Attacks the target is attacked with the damage given by the variables.

- Input: none.
- Called by: Game.

- Validity checks: Check that Monster target is not null or out of range. Calls findMonster if check fails. Also checks that timeToFire is true. Otherwise it waits.
- Accesses: Monster target.
- Calls: timeToFire and findMonster.
- Return value: none.

upgrade

Upgrades the tower to the next level, increasing the different variables as appropriate.

- Input: none.
- Called by: Button.
- Validity checks: Checks that the tower is not max level and that the player has enough gold for an upgrade.
- Accesses: Player.
- Calls: getGold in Player.
- Post-condition: Tower is at the appropriate level.
- Return value: none.

sell

Sets sell_flag to int value.

- Input: none.
- Called by: Button.
- Accesses: vector<Monster> monsters.
- Calls: calculatePath in Monster.
- Return value: none.

timeToFire
Decreases time_to_fire with 1.

- Input: none.
- Called by: attackMonster.
- Post-condition: Sets time_to_fire to int rate if it is 0.
- Return value: True if time_to_fire is 0, false otherwise.

moveProjectiles
Iterates over all projectiles and calls moveProjectile.

- Input: none.
- Called by: attackMonster.
- Calls: moveProjectile in Projectile
- Return value: none.

Get functions for all variables
Returns the different variables.

- Input: none.
- Called by: Game.
- Return value: the variable value.

5.5.8 Class: Monster

Monster inherits from Button and every monster-type will have its own subclass inheriting from the monster class that contains specialized methods for the specific monster-type.

Variables

int hp
int display_hp
int die_flag
int level
float speed
list<Coordinate> path

Functions

calculatePath
Calculates the shortest path between the current position and the goal and saves it in list<Coordinate> path.

- Input: none.
- Called by: constructor and sell in Tower.
- Post-condition: The path will not be saved if the method returns false.
- Return value: False if there is no path, true otherwise.

moveMonster
Moves the monster towards the next coordinate in the path at the set speed.

- Input: none.
- Called by: Game.
- Return value: none.

setDisplayHp
Sets display_hp.

- Input: New int value for display_hp.
- Called by: Projectile.
- Return value: none.

Get functions for all variables
Returns the different variables.

- Input: none.
- Called by: Game and Projectile.
- Return value: the variable value.

5.5.9 Class: Graphics

Variables

Functions

printString
Prints a string at the position given by the coordinate.

- Input: The string and coordinate to print.
- Called by: AETD and Game.
- Return value: none.

drawEntity
Draws an entity with the image and at the coordinate specified by the entity variables.

- Input: an Entity to print.
- Called by: AETD and Game.
- Accesses: Entity.
- Calls: getImageName and getPos in Entity.
- Return value: none.

drawSquare
Draws a colored square between the coordinates specified.

- Input: The color and the coordinates of the upper left and lower right corners of the square.
- Called by: AETD and Game.
- Return value: none.

5.5.10 Class: Chat

Variables

vector<String> history
Coordinate pos

Functions

addInputMessage
Calls Client with the message received from the InputManager and adds the it to history.

- Input: The string to send.
- Called by: InputManager.
- Accesses: Client and vector<String> history.
- Calls: setOutData in Client.
- Return value: none.

addNetworkMessage
Adds the message from the Client to history.

- Input: The string received.
- Called by: printHistory.
- Validity checks: Checks that the data from the method getInData is information relating to the chat. Ignores the data otherwise.
- Accesses: vector<String> history.
- Calls: getInData in Client.
- Return value: none.

printHistory
Calls graphic with the history variable and the positions given by the coordinate.

- Input: none.
- Called by: AETD and Game.
- Calls: printString in Graphics and addNetworkMessage.
- Return value: none.

5.5.11 Class: Server

Variables

queue<String> out_data
queue<String> in_data
vector<int> port
vector<int> ip

Functions

sendTo
Sends out_data to the clients.

- Input: none.
- Called by: AETD and Game.
- Accesses: queue<String> out_data.
- Return value: none.

recvFrom
Receives data from a client and stores it to in_data.

- Input: none.
- Called by: AETD and Game.
- Accesses: queue<String> in_data
- Return value: none.

socket
Creates an endpoint at the server.

- Input: none.
- Called by: AETD and Game.
- Return value: none.

bind
Function for setting up the connection with the clients socket.

- Input: none.
- Called by: AETD and Game.
- Accesses: vector<int> port and vector<int> ip.
- Return value: none.

5.5.12 Class: Client

Variables

queue<String> out_data
queue<String> in_data
int port
int ip

Functions

sendTo
Sends out_data to the server.

- Input: none.
- Called by: AETD and Game.
- Accesses: queue<String> out_data.
- Return value: none.

recvFrom
Receives data from the server and stores it to in_data.

- Input: none.
- Called by: AETD and Game.
- Accesses: queue<String> in_data
- Return value: none.

socket
Creates an endpoint at the client.

- Input: none.
- Called by: AETD and Game.
- Return value: none.

bind
Function for setting up the connection with the servers socket.

- Input: none.
- Called by: AETD and Game.
- Return value: none.

5.5.13 Class: InputParser

Variables

bool left_state
Coordinate pos
char key

Functions

chatMsg
Sends a written chat message to Chat.

- Input: none.
- Validity checks: Checks that the string written is a chat message.
- Calls: addInputMessage in Chat
- Return value: none.

leftClick
Checks if the location of the click was the location of a button.

- Input: none.
- Calls: buttonFunction
- Return value: none.

Get functions for all variables
Returns the different variables.

- Input: none.
- Called by: Game or AETD.
- Return value: the variable value.

5.6 Package Diagram

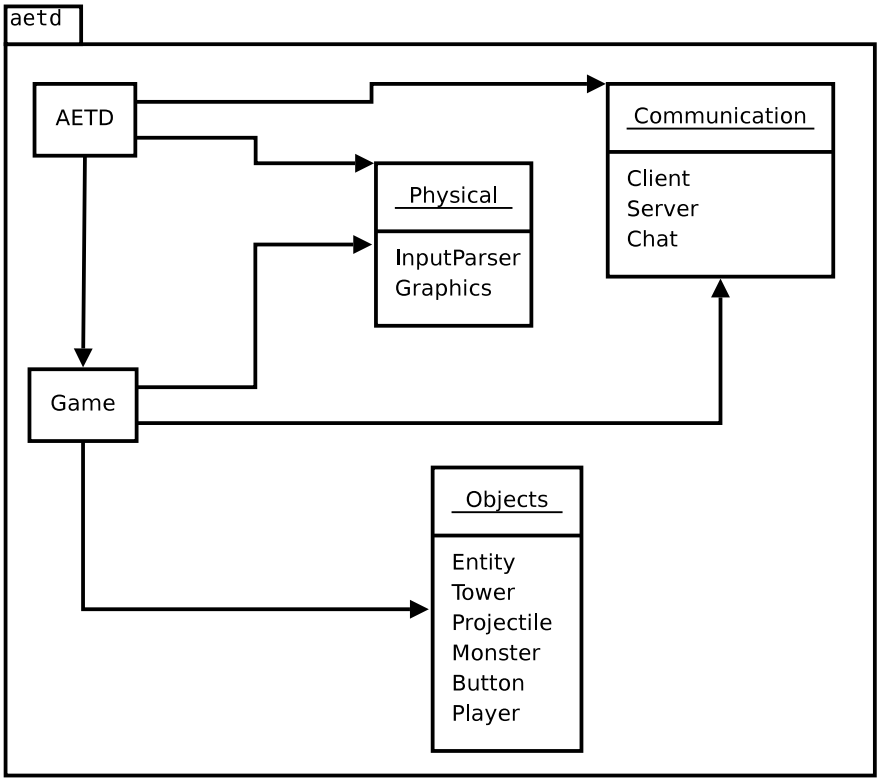


Figure 17: Package Diagram

6 Functional Test Cases

6.0.1 Download the program

- **Description**

  The program should be available to download from our homepage.

- **Test-precondition**

  User must have Internet access.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.1 page 22.

- **Input**

  Address to webserver.

- **Output**

  File downloaded to the users computer.

- **Instruction**

  1. Enter the address to our homepage in a web browser.
  2. Click the download link.
  3. Wait until file is downloaded.

6.0.2 Install the program

- **Description**

  Install the game on the users computer.

- **Test-precondition**

  Installation file downloaded to the users computer.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.1 page 22.

- **Input**

  Where to install the program.

- **Output**

  Program installed on the users computer.

- **Instruction**

  1. Run the install file.
  2. Choose install path.
  3. Wait until the installation finish.

6.0.3 Configure game settings

- **Description**

  Alter settings such as player name.

- **Test-precondition**

  User is in the main menu.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.8 page 27.

- **Input**

  Users choice of alterations to game settings.

- **Output**

  Selected game settings have been altered.

- **Instruction**

  1. Click on the “Settings” button.
  2. Alter the settings as preferred.
  3. Click on the “Save” button to save and exit to main menu.

6.0.4 Play singleplayer game

- **Description**

  To start a singleplayer game without having and Internet connection.

- **Test-precondition**

  User is in the main menu.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.2 page 23.

- **Input**

  None.

- **Output**

  The single player game starts.

- **Instruction**

  1. Click on the "Start singleplayer game" button.

6.0.5 Host multiplayer game

- **Description**

  The user makes his computer the host computer of a multiplayer game.

- **Test-precondition**

  User is in the main menu.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.3 page 24.

- **Input**

  None.

- **Output**

  The multiplayer lobby is displayed.

- **Instruction**

  1. Click on the "Host multiplayer game" button.

6.0.6 Join multiplayer game

- **Description**

  The user connects to a host computer of a multiplayer game.

- **Test-precondition**

  User is in the main menu.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 18.
  Section 6.3.4 page 24.

- **Input**

  IP-address to the host.

- **Output**

  The multiplayer lobby is displayed.

- **Instruction**

  1. Click on the "Join multiplayer game" button.
  2. Enter the IP-address of desired host.
  3. Click on the "Join" button.

6.0.7 Build Towers

- **Description**

  To purchase and place a tower on the playing field.

- **Test-precondition**

  User is in a game and can afford the selected tower type.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 19.
  Section 6.3.5 page 25.

- **Input**

  A valid position where the tower should be built and the type of tower the user wishes to build.

- **Output**

  A tower of the correct type is built on the playing field where the user specified.

- **Instruction**

  1. Select a tower type from the menu by clicking on its symbol.
  2. Move the mouse to the position on the playing field where the tower is to be built.
  3. Click the mouse to build the tower.
  4. Wait until the tower is built.

6.0.8 Select a specific tower

- **Description**

  Select a tower already built on the playing field.

- **Test-precondition**

  Tower built on the playing field.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 19.
  Section 6.3.6 page 26.
  Section 6.3.9 page 28.

- **Input**

  A position of a tower built on the playing field.

- **Output**

  Tower selected gets marked.

- **Instruction**

  1. Click on a tower on the playing field.

6.0.9 Sell towers

- **Description**

  Sell a tower already built on the playing field.

- **Test-precondition**

  User is in a game and has at least one tower on his playing field.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 19.
  Section 6.3.9 page 28.

- **Input**

  A tower on the playing field.

- **Output**

  The selected tower is sold and removed from the playing field.

- **Instruction**

  1. Select the tower to be sold on the playing field.
  2. Click the "Sell tower" button to sell the tower.
  3. Wait until the tower is removed from the playing field.

6.0.10 Upgrade towers

- **Description**

  Upgrade a tower already built on the playing field.

- **Test-precondition**

  User is in a game and has at least one tower on his playing field and can afford to upgrade the selected tower. The selected tower is not already max level.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 19.
  Section 6.3.6 page 26.

- **Input**

  A tower on the playing field.

- **Output**

  The selected tower is upgraded.

- **Instruction**

  1. Select the tower to be upgraded on the playing field.
  2. Click the "Upgrade tower" button to upgrade the tower.
  3. Wait until the tower is upgraded to a higher level.

6.0.11 View individual tower statistic

- **Description**

  See information about the selected tower.

- **Test-precondition**

  Tower built on the playing field.

- **Reference to Requirements Document**

  Section 4.1 page 13.
  Section 6.1 page 20.

- **Input**

  A tower on the playing field.

- **Output**

  Tower statistic is displayed in the tower information box.

- **Instruction**

  1. Click on a tower on the playing field.
  2. Observe tower statistic in the tower information box.

6.0.12 Monster levels

- **Description**

  Monsters will appear at set intervals and at increasing levels.

- **Test-precondition**

  User must be in a game.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 20.

- **Input**

  None.

- **Output**

  The level of the monster in the monster information box has changed.

- **Instruction**

  1. Observe the level of the monsters in the current wave.
  2. Observe the level of the monsters in the next wave.

6.0.13 Monster movement

- **Description**

  The monsters will automatically move from one side of the playing field to their goal, always choosing the shortest path available.

- **Test-precondition**

  User must be in a game.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 20.

- **Input**

  None.

- **Output**

  Monster is moving on screen.

- **Instruction**

  1. Observe the monster moving on the screen.

6.0.14 Kill monsters

- **Description**

  Monsters can be killed by towers.

- **Test-precondition**

  User must be in a game and have built a tower.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 20.

- **Input**

  None.

- **Output**

  Information about how many monster one player has is updated for the other players when in multiplayer mode.

- **Instruction**

  1. Monster passes a tower.

6.0.15 Receive information about monster's health

- **Description**

  See information about the selected monster.

- **Test-precondition**

  User must be in a game and a monster must be visible on the screen.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 21.

- **Input**

  None.

- **Output**

  Information about how many health point the specific monster have displayed in the monster information box.

- **Instruction**

  1. The user left-clicks with the mouse on a specific monster.

6.0.16 Send monsters to other players

- **Description**

  Player can send monsters to other players in a multiplayer game.

- **Test-precondition**

  User must be in a multiplayer game and have enough gold to send monsters.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 21.
  Section 6.3.7 page 27.

- **Input**

  None.

- **Output**

  The monster counter is updated in another players information box. Gold is deducted.

- **Instruction**

  1. The user presses the "Send monster" button on the information box of another player.

6.0.17 Chat with other players

- **Description**

  The users can send text messages to each other.

- **Test-precondition**

  The user must be in a multiplayer game.

- **Reference to Requirements Document**

  Section 4.1 page 14.
  Section 6.1 page 21.
  Section 6.3.10 page 28.

- **Input**

  The message that is to be sent to the other players.

- **Output**

  The text written is displayed on the screen.

- **Instruction**

  1. The user presses the "Chat" button.
  2. The user types the message that are to be sent to the other users and presses the "Send" button.