Tower Defence Game

Build - 3

By

Team 3

Yogesh
Gagandeep
Gaurav

Table of Contents

1. Introduction

A Tower Defense game is a simple desktop based application game developed in java. It is a single user game which can be played by one player.

The game consists of a map in the form of a grid where a path needs to be created by the user. This path will consist of a start point and an exit point. Critters will move from the start point to the end point. The user needs to place towers on the map outside the path which will fire at the critters as they move from start to end point. The user needs to destroy all the critters using the towers before they reach the exit point and earn coins. The user gets 250 coins in each game

If the user is not able to stop the critters from reaching the end point, the player's life in the game will start to decrease and ends if it crosses 20 points.

2. Functional Requirements

a- The User should be able to create and edit a map:

- User- Driven interactive creation of a map, user should be able to choose the size of the map.
- User Driven allocation of map elements such as scenery, path, entry and exit point.
- The user should be able to save a valid map.
  A map is considered valid if:
  - It consists of an entry point
  - It consists of an exit point
  - The map should be of size 20 * 12 (Maximum)(Width*Height) , 5*5 (minimum)
  - The map should be continuous and connected.
- The user should be able to load and edit an existing map.

b- The user should be able to play the game:

- Game starts by user selection of a previously user-saved map, then loads the map
- Wave-based play: First (pre-wave phase) the player can place new towers, upgrade towers, sell towers, and signify that critters are allowed in on the map, when all critters in a wave have been killed or reached the end point, a new wave starts.
- End of game, e.g. when a certain number of critters reach the exit point of the map, or the critters steal all the player's coins, or the player succeeds in killing a certain number of waves.
- Implementation of currency, cost to buy/sell a tower, and reward for killing critters.
- Critter waves are created with a level of difficulty increasing at every wave. Difficulty must involve increasing critter speed and toughness
- Implementation of at least three different kinds of towers that are characterized by special damage effects. The mandatory special damage effects are: splash (inflicts damage to critters around the targeted critter), burning (inflicts damage over time after a critter has been hit), freezing (slows down the critter for some time).
- The towers can target the critters using the following mandatory strategies: nearest to the tower, nearest to the end point, weakest critter, strongest critter. It must be possible to set a different targeting strategies for individual towers.
- Tower inspection window that dynamically shows its current characteristics, allows to sell the tower, increase the level of the tower, select the tower's targeting strategy and view the individual tower's log (see below).
- Critter observer that allows to dynamically observe the current hit points of any critter on the map.

c- The user should be able to manage the game by checking the logs:

- Game log that records all events happening in the game, including placement/upgrade/selling of towers, critter wave creation, etc. The log must allow for the viewing of the whole log in sequential order

(i.e. ordered in time) or certain portions of the log related to a certain aspect of the game, also ordered according to time, e.g. –

- **Individual tower log**: time-ordered log of all events related to a specific tower
- **Collective tower log**: time-ordered log of all events related to all towers (i.e. time inter-meshing of the previous)
- **Wave log**: all activities that happened in any specific wave of the game (select a wave and list sorted by time the events happened in this wave, from pre-wave tower edition phase to end of the wave).
- **Global game log**: all events that happened throughout the entire game up to now, sorted by time.

- Map log that records in the map file the time of original creation of the map, when it was edited, when it was played and what was the result of the game every time it was played. When a map is being played, the list of five highest scores is presented before the game starts, as well as when the game ends.
- Save/Load a game in progress: As a game is being played, allow the user to save the game in progress to a file, and allow the user to load the game in exactly the same state as saved.

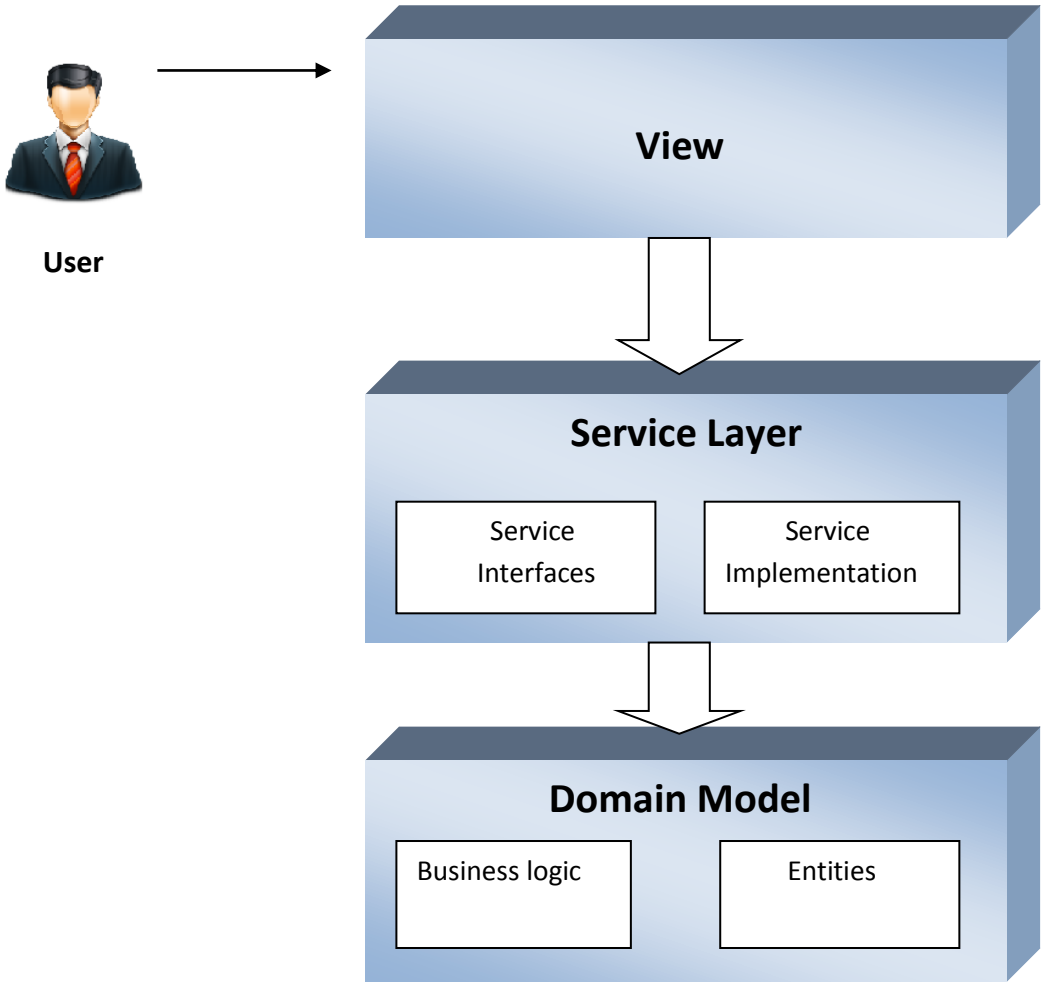
3. Requirement Analysis

The game needs to be developed which consists of towers and critters to defend the tower before the critters reaches the end point. One of the most important aspects of the game is the towers. A tower can be placed on the map and used to shoot the critters. The three towers in the project have different specifications and characteristics. A tower can be sold or upgraded with the coins/money available. Different maps can be created and edited by the user as per ones choice and only the valid maps can be saved. Once a certain number of critters reach the exit point, the player loses and the game ends.

4. Game Architecture

The project is being developed in 3 builds or releases. The 3 layered architecture is as shown below:

a) Architecture Diagram



b) Description

- **Domain Model**

  The model of the game controls the behavior of the application. The model directly manages the data, logic and rules of the application.

The model notifies its associated views and service classes (using observer design pattern) when a state change is initiated. This notification allows views to produce updated output to change the available set of commands.

Classes belong to Model component of the Tower Defense Game are:

- ChambyTower.java
- CitidelleTower.java
- Critter.java
- CritterMovingBehaviour.java
- CritterNormalMove.java
- Defender.java
- FirePower.java
- FireRange.java
- FireSpeed.java
- HeleneTower.java
- LifeManager.java
- Location.java
- MapContents.java
- MapGrid.java
- MapObjects.java
- NormalCritter.java
- Subject.java
- Tower.java
- TowerFeatureDecorator.java
- TowerLevelEnum.java
- TowerLineShooting.java
- TowerLineShootingBehaviour.java
- TowerMovingBehaviour.java
- TowerNoMove.java
- TowerShootingBehaviour.java
- TowerTypeEnum.java
- VisualMapObjects.java
- Wave.java

- **View**

  The view classes of the game render the model into a suitable form for visualization or interaction. Multiple views exist for a single model element of the game and rendered based on the game's state. When the model data of the game is modified, the view classes are notified about the state change and the view updates its presentation based on the model's state.

  Classes belong to view component of the Tower Defense game are:

  - CellView.java
  - CritterView.java
  - EmptyBarPanel.java
  - FireShoot.java
  - GameControllerPanel.java
  - GameInfoPanel.java
  - GridItemsLayerPanelView.java
  - GridLayerPanelView.java
  - IceShoot.java
  - LogConsole.java
  - MainFrame.java
  - MapEditor.java
  - MapFrame.java
  - MouseEventHandler.java
  - ObjectCanvas.java
  - ScoreCard.java
  - SelectedTower.java
  - SplashShoot.java
  - TowerInfoPanel.java
  - TowerInspectionPanel.java
  - TowerPanelLayout.java
  - TowerSelectionPanel.java
  - TwoLayerPanelView.java

- **Service Layer**

  The Service layer classes of the Game acts as the middle man of the application. Service layer defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation.

  Classes belong to Service layer of the Tower Defense Game are:

  - Box.java
  - CellService.java
  - CritterBurnService.java
  - CritterFireService.java
  - CritterFreezeService.java
  - CritterShootingService.java
  - CritterSplashService.java
  - DefenderInformer.java
  - GameStateController.java
  - GraphService.java
  - IPathCompressor.java
  - IPathEndPointsCheckerService.java
  - IPathValidityChecker.java
  - ITowerInformer.java
  - IUnion.java
  - LogController.java
  - MapController.java
  - MapUtility.java
  - MoneyController.java
  - Observer.java
  - PathCalculator.java
  - PathCompressor.java
  - PathEndPointsCheckerService.java
  - PathService.java
  - PathValidityChecker.java
  - PlayerLifeService.java

- PositionService.java
- TowerFactory.java
- TowerSellPriceCalculator.java
- WaveFactory.java

5. Programming Process

a) Coding Standards

The most general coding conventions have been followed while developing the codes which are as follows.

a. The name of the classes start with a upper case character e.g.: GameMainApp.java
b. Constants are named with upper case characters and include Underscore between two words (if applicable).
c. The name of the variables are descriptive and are written in lower Case including a capital letter to separate between words.
d. The name of the methods start with a lower case character and Use uppercase letters to separate words.
e. Enums are named with upper case characters

b) Design Pattern Used

The following design patterns have been used and implemented in the project:

- Factory pattern for creating different types of Towers
- Singleton pattern for maintaining Money and Life throughout the game
- Observer pattern for information about critter position
- Decorator pattern for decoration of tower features like firing speed, firing power and firing range

c) Architectural Design

The Game is developed and implements the service layer design architecture pattern.

d) Software Versioning Repository

The Concurrent Versioning System (CVS) in Eclipse has been used effectively in the development process. It has been used by the members to maintain different version as well has parallel development of the same class/module.

e) API Documentation

Java doc has been used in the project to make all the necessary comments for classes, methods and inline code and also to document it.

f) Unit testing Framework

Junit has been used in the project to write test cases for different tested class and tested methods. 52 different test cases have been written for the build 3 of the project.

6. Screenshots

Fig 1 – Launch Screen

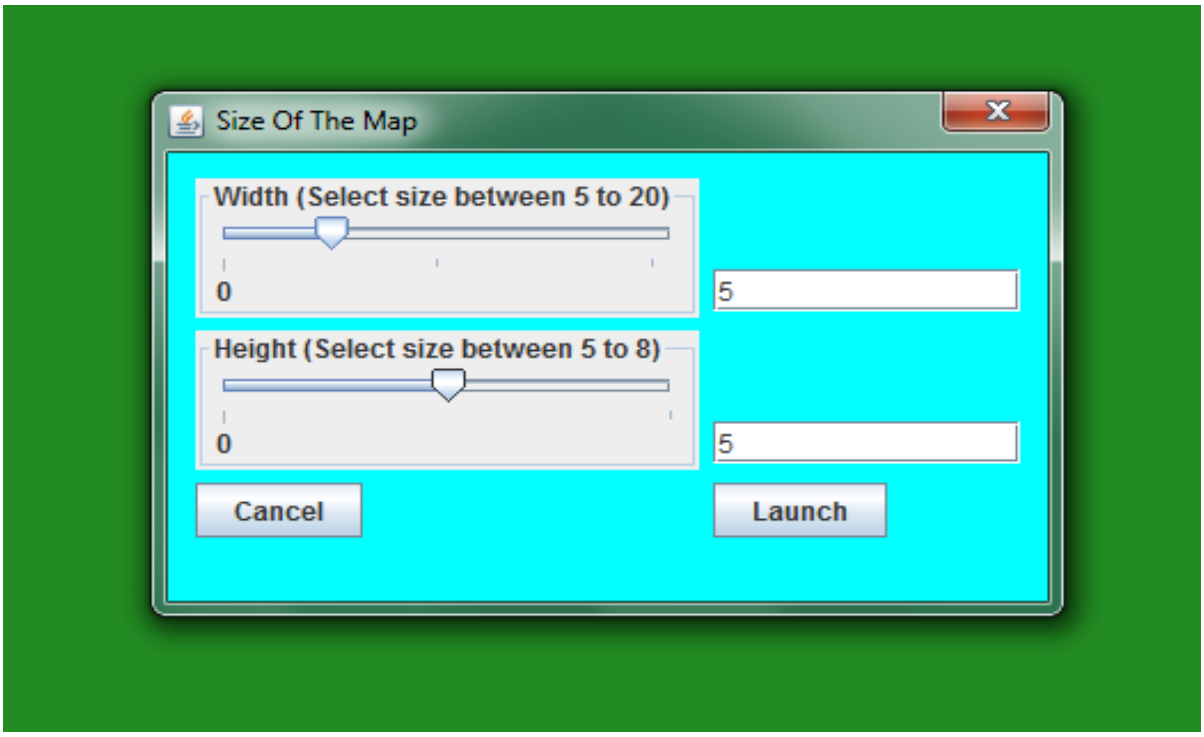


Fig 2A – Creating the Map

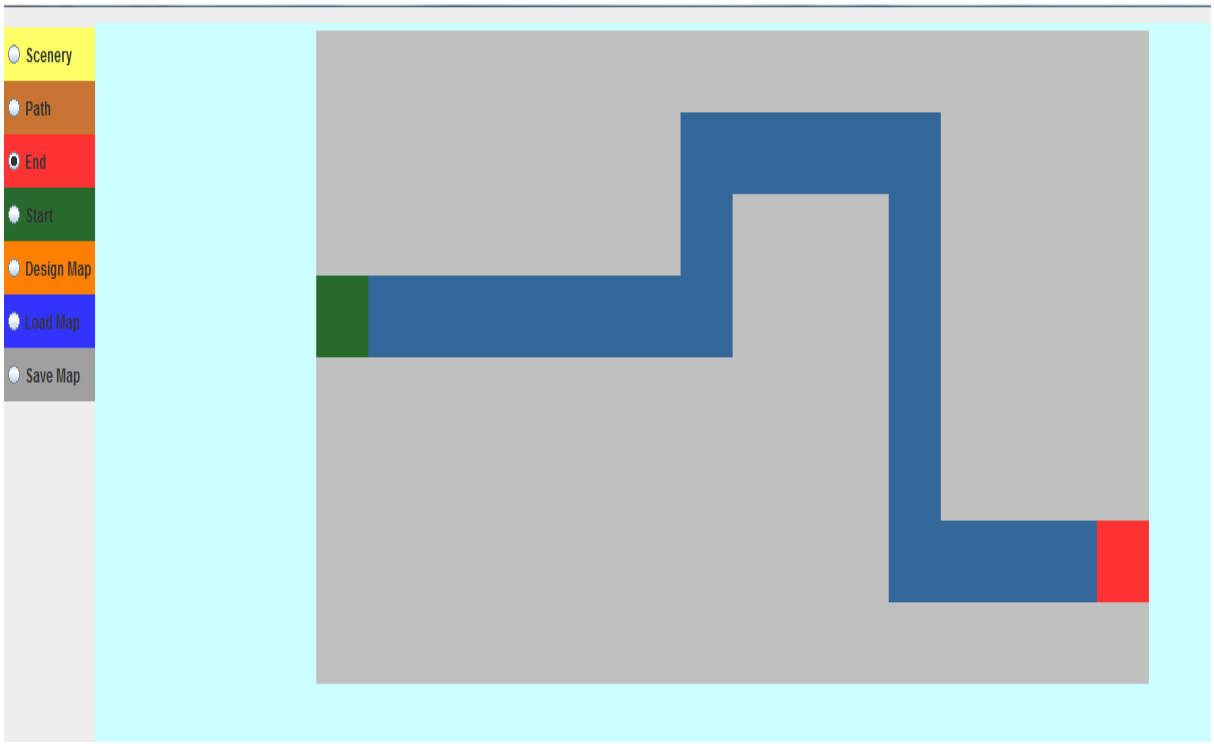


Fig 2B – Creating the Map

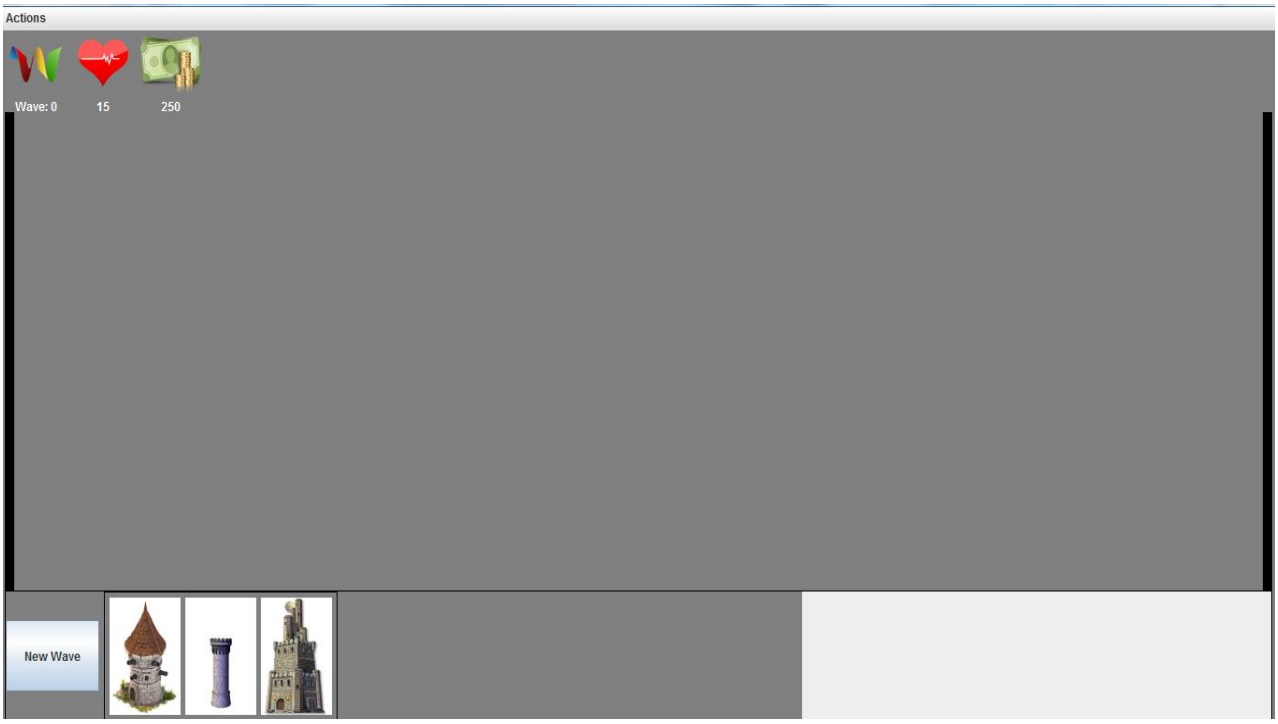


Fig 3 – Game Play screen

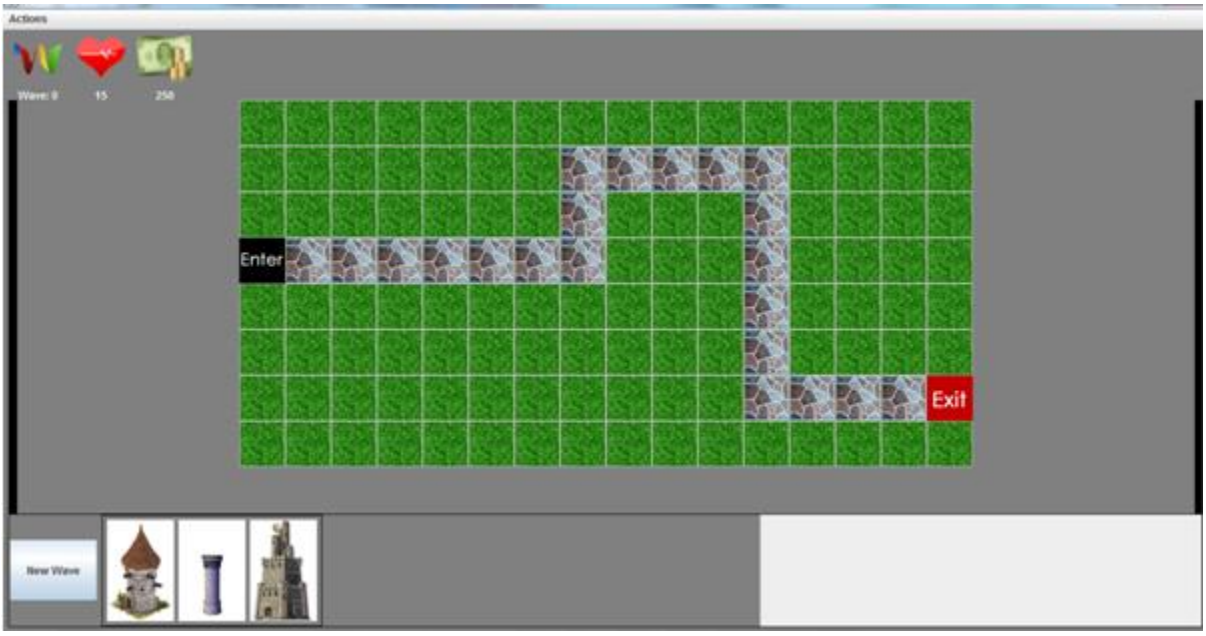


Fig 4 – Loading the Map

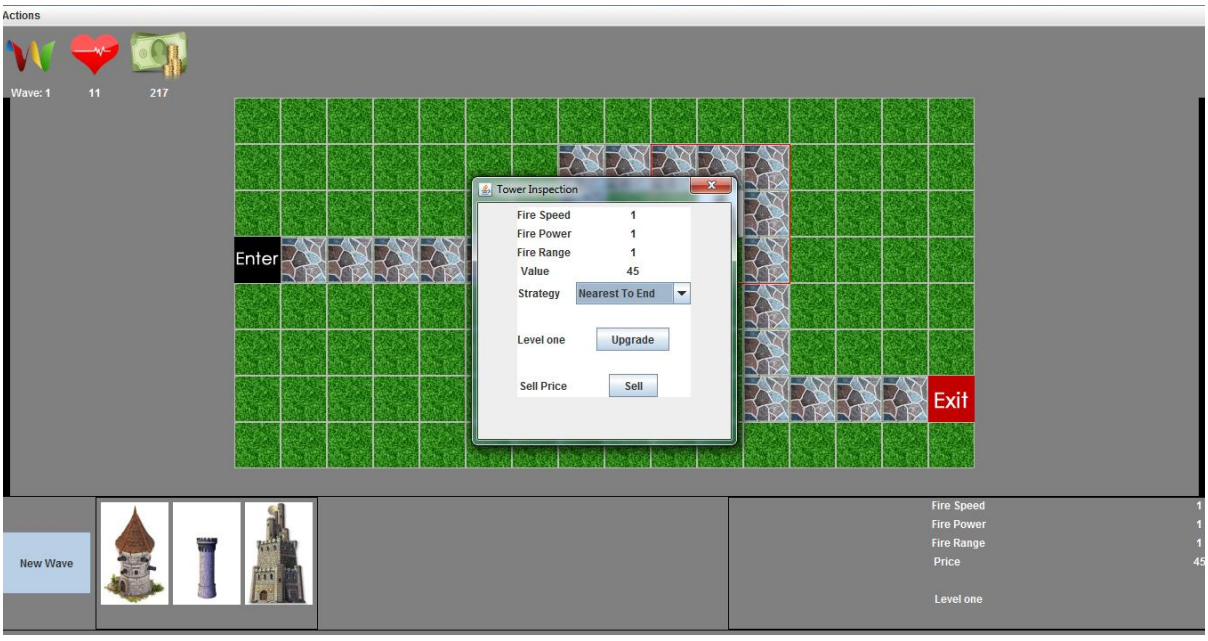


Fig 5 – Changing Tower Strategy

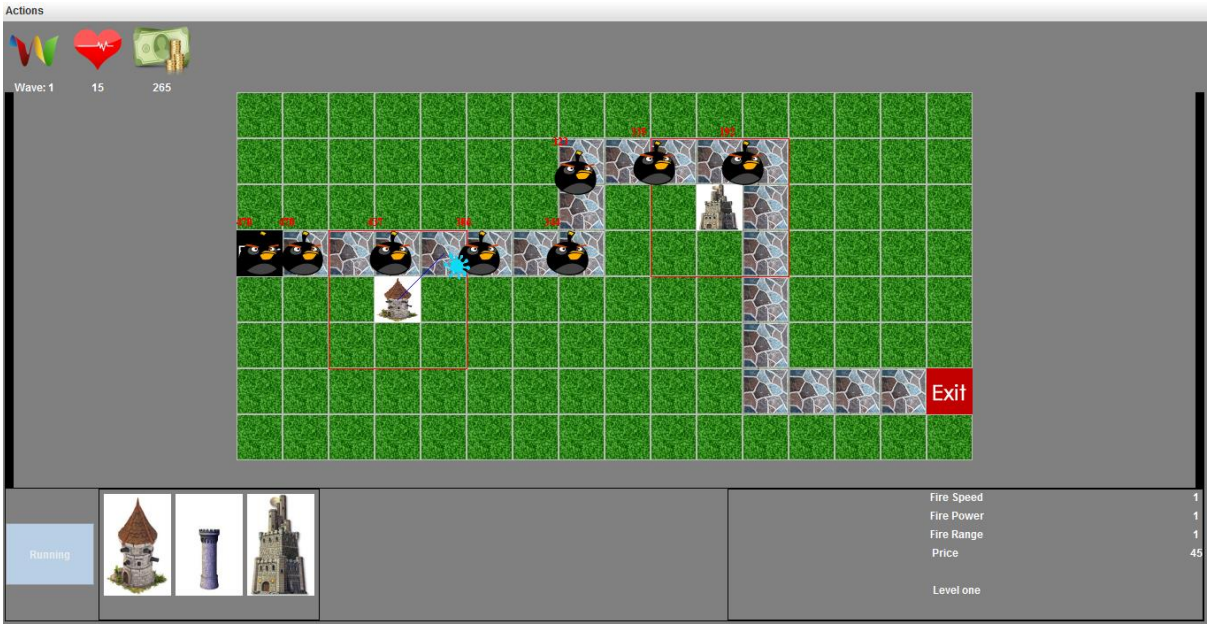


Fig 6 – Playing the game

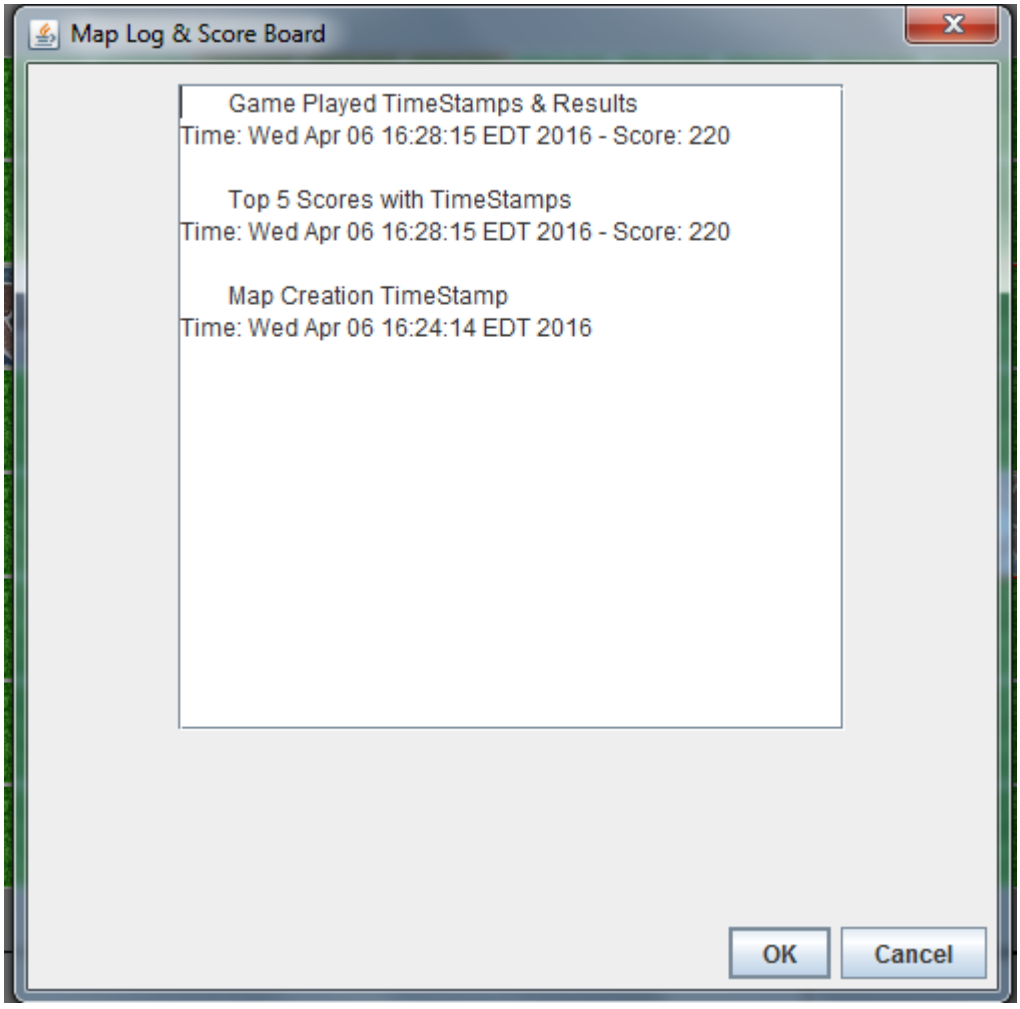


Fig 7 – Map log and Score board

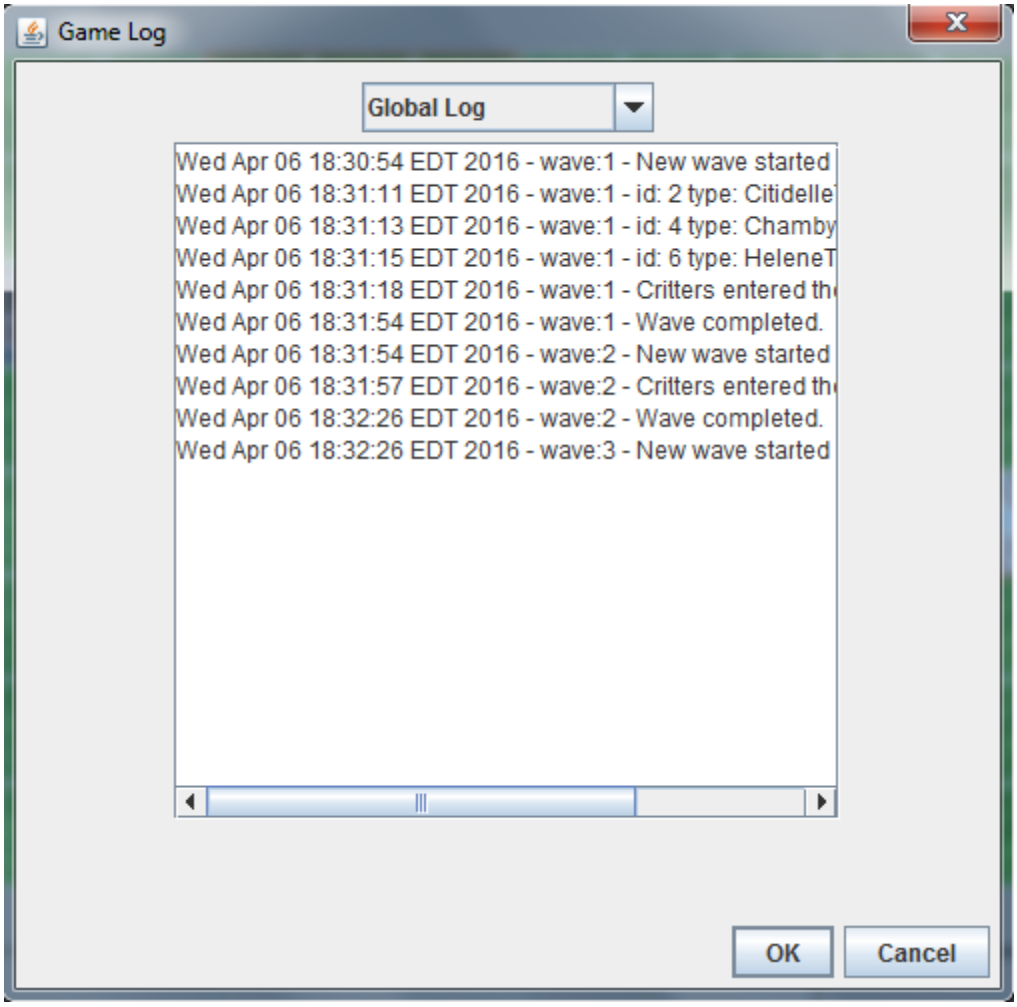


Fig 7 – Game Log

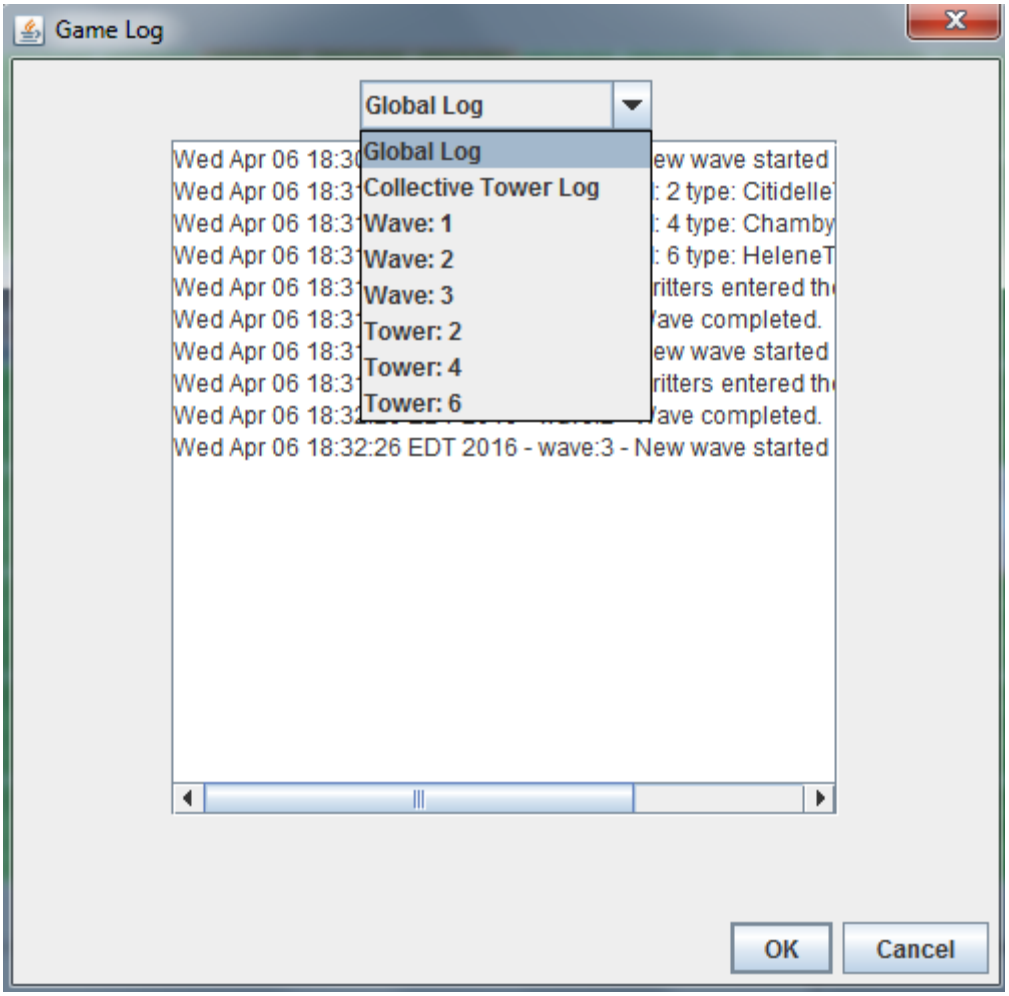


Fig 8 – Game Log