ECSE 321 Introduction to Software Engineering

McGill University Winter 2015

Tower defense project:  
Software Design Specifications

Andrei Chubarau 260581375

Yordan Neshev 260587938

Dang Khoa Do 260584925

Steven Voyer 260531264

Table of Contents

[1 Introduction 2](#_Toc414197302)

[1.1 Purpose of the document 2](#_Toc414197303)

[1.2 audience 2](#_Toc414197304)

[1.3 Scope 2](#_Toc414197305)

[1.4 Related documents 2](#_Toc414197306)

[2 System Overview 3](#_Toc414197307)

[3 Design Considerations 4](#_Toc414197308)

[3.1 Assumptions and Dependencies 4](#_Toc414197309)

[3.2 Constraints 4](#_Toc414197310)

[3.3 Goals and Guidelines 4](#_Toc414197311)

[3.4 Development Methods 4](#_Toc414197312)

[4 System Architecture 5](#_Toc414197313)

[4.1 Architectural Strategies 5](#_Toc414197314)

[4.2 Architectural Diagram 5](#_Toc414197315)

[5 Detailed System Design 6](#_Toc414197316)

[5.1 Component level design 6](#_Toc414197317)

[5.1.1 Layer 1 6](#_Toc414197318)

[5.1.2 Layer 2 6](#_Toc414197319)

[5.1.3 Layer 3 6](#_Toc414197320)

[5.1.4 Layer … 6](#_Toc414197321)

[5.2 Object Oriented Principles and Patterns 7](#_Toc414197322)

[5.2.1 Principles Used 7](#_Toc414197323)

[5.2.2 Pattern Used 7](#_Toc414197324)

[6 Dynamic Behaviour: Sequence Diagrams 8](#_Toc414197325)

[7 (System Logic…) Design 9](#_Toc414197326)

[8 User Interface Design 10](#_Toc414197327)

[8.1 Description of the User Interface 10](#_Toc414197328)

[8.2 Visual Presentation of Graphical User Interface 10](#_Toc414197329)

[9 References 11](#_Toc414197330)

# Introduction

## Purpose of the document

The purpose of this document is to precisely outline the system architecture and detailed design plans of the Tower Defence game. This document will describe exactly how the game will be built and exactly how it will run whilst taking the audience and constraints into consideration. This will be achieved with architectural diagrams, descriptions of the different layers of the design and sequence diagrams to analyze the dynamic behaviour of the game.

## audience

The primary intended audience for this game is the instructor (and Teaching Assistant) of the ECSE-321 course at McGill University since this is a class project. However, since this will be a fully functional and playable game, the secondary intended audience is also any player interested in playing the game.

With regards to this document, the intended audience is first and foremost the very people writing this document since they are the same people developing the game. This document will greatly aid in organizing thoughts and lead to a more structured development process. In addition, this document is a great reference point in testing the system and ensuring that all components work as expected. Secondly, this document is also intended for the graders of this project since it will allow them to gain more insight into the internal workings of the game.

## Scope

The scope of this document includes all relevant architectural models and design specifications to the Tower Defence game. The boundaries of this scope exclude everything that is not required by the graders of this project.

## Related documents

Please see the Software Requirements Document for more information of the requirements of the development of this game.

# System Overview

Tower Defence is a type of real-time strategy video game that has been around since the early 1990’s. In the case of this project, the object of the game is to prevent computer controlled enemies from reaching the end of a certain path. To achieve this, the player will strategically place defensive structures or towers which will attempt to eliminate the enemies as they travel along the path. Killing (or damaging) enemies and completing levels and tasks will grant score points to the player which can be used to buy more towers or even upgrade current ones in anticipation of stronger enemies in upcoming levels. Total Score will be recorded and displayed in the Leaderboards. In addition, the player also has the option to create or edit custom maps. This can be done by using the in-game map manager.

In regards to how this will be implemented, the system is made up of three main parts: the Map class, the Enemies class and the Structure class. The Map class creates, maintains and manages everything map related. The Enemies class creates, maintains and manages everything about all critters that will be encountered within the game. Finally, the Structure class takes care of all structures, including defensive towers and scenery, which will appear on the map.

However important these three components may be to the game, there are other classes that must be considered in order to tie these components together and create a playable game. Such classes include movable which takes care of all movable map components. UGameView which handles the layer of the game directly presented to the user. GameTime which holds all current game parameters and GameController which is a listener and is essentially the link between the user and the game. These, along with the three main componential classes (and some other smaller classes), make up the system that will make the game function as intended.

# Design Considerations

## Assumptions and Dependencies

It is important to first state all assumption made during the development of this project along with any dependencies. Firstly, the project instructions does not state which type of user interface should be implemented. Therefore, a graphical interface (as opposed to text-based) will be made which will enable the user to visually see the layout of the map, enemy positions, tower positions, scenery positions and tower-enemy interactions. This will be done by manipulating graphical sprites. In addition, the login, main and game time menus will all be graphically implemented as well.

Secondly, while the main object of the game will remain intact, the developers are free to produce their own versions of towers and enemies (as well as overall theme of the game) since these parameters have not been constrained in the requirements of the client.

## Constraints

In terms of constraints, the first and most important thing to note is that this game must be completed by April 6th, 2015. It is essential that this time constraint be respected.

Technologically, this game must be programmed in JAVA and all in-game sprites be designed using the Swing library and public-use images. While little intricate features of the game are open to programmer imagination, the integrity of the original Tower Defence game must be maintained. The following features must be present:

* Main menu and Leaderboards complete with a user login feature.
* A functional map editor which can differentiate between valid and non-valid maps. The path and the scenery must be separate (path must have one entry and one exit point)
* Currency system to buy different types of towers with different attributes (Status ailment, range, strength, projectile type etc…)
* Enemies (Wave-based) that will attack the player. There will be several types of enemies each with their own unique attributes (hit points, speed, kill worth etc…)
* Towers can only be put on the scenery and enemies can only travel on the path.

All of these features will be implemented in the final game along with others that were not specified in the requirements but will slightly alter the game for originality purposes.

## Goals and Guidelines

In terms of goals and guidelines, the most important goal is that the game be playable and amusing to the player. Secondly, the game must be complete by April 6th such that full marks will be received and the developers will be ready for the presentation.

In terms of the code of the game itself, an important goal is to make said code as adaptable to changes as possible. This means that if there are any rule changes or any slight modifications to be done, they can be done efficiently in just a single place of the code without affecting the other parts.

## Development Methods

The development methods that will be used in this project include object oriented programming, test driven iterative methods and the GRASP programming principles.

## 

# System Architecture

## Architectural Strategies

The design of the current system is based on the layered architectural style, where the main system is divided into two main subcomponents, namely Presentation and Business Logic layers. Such an approach limits interaction between the two layers and separates responsibilities to achieve higher system cohesion.

The previously mentioned duality of the system ensures separation of concerns. Business Logic layer handles all system parameters and user inputs, whereas Presentation layer has the responsibility of displaying the resulting system behaviour. This contributes to managing system complexity by separating system domain logic and system view functionality.

Moreover, Presentation layer depends on Business Logic layer as the latest provides services and information to the first. The Model-View separation principle is applied as Business Logic has no dependency on the Presentation layer. This design style contributes to high cohesion and low coupling of the system. Additionally, this allows the system to display the same domain logic component in different presentation styles, thus enhancing reuse and extendibility.

## Architectural Diagram

[INSERT DIAGRAM HERE]

# Detailed System Design

## Component level design

System subcomponents are explained in detail in the following sections.

### Presentation Layer

UGameView class is at the base of the Presentation layer. It is responsible for displaying the system parameters and communicating user input to the Business Logic layer. UGameView communicates with a single GameTime instance that maintains all game parameters of the system. A variation of subpanels is developed to provide user interface including game menus, game view components and game object representations.

[INSERT DIAGRAM HERE]

### Domain Logic Layer

Business Logic layer is mainly characterized by GameTime and Game Controller classes. An instance of GameTime contains all runtime parameters of the system such as Map, Structures, and others. Presentation layer, namely UGameFrame, accesses GameTime to acquire system state and display it, but GameTime does initialize communication with the presentation layer. GameController listens for and handles user input communicated to it through the presentation layer and passes on the information to the GameTime instance that transmits required changes to individual system components. GameController is implemented as a Façade Controller as it is singular and represents the overall system.

[INSERT DIAGRAM HERE]

## Object Oriented Principles and Patterns

This section explains and concretises on design decisions carried out during the development process of the current system. As these decisions were made based on common object-oriented principles and patterns (GRASP and GoF), the following text describes and demonstrates decision applicability and significance in terms of these notions. Additionally, relevant partial UML class diagrams or Domain Models are provided for visual inspection of considered system components.

### Principles Used

1. Principle: Information Expert

Problem: What system component should be assigned the responsibility of generating and maintaining the path for critters to move on?

Solution: In the current system, Map class manages and constructs all map related components of a game instance. This includes storing all tile information and links between these. The process of building a path depends on and is limited by map parameters which must be accessed during construction of path. Therefore, the Map class should be responsible for building and maintaining path and is the information expert in this scenario.

1. Principle: Creator

Problem: What system component should be assigned the responsibility of creating an instance of Map?

Solution: In the current system, GameTime class is responsible for holding all game time parameters of a played game, notably the map it is played on. GameTime also is responsible for delegating runtime system parameters to the Presentation Layer for display. Therefore, GameTime class should initialize an instance of Map upon runtime of the application.

1. Principle: Controller

Problem: What system component should be in charge of handling user input events?

Solution: In the current application, GameController class is implemented as a Façade Controller, which implies that it handles all user input to the system. It is coupled directly to GameTime class from Business Logic layer and to UGameView class from the Presentation layer to allow for relevant user input processing and runtime parameters manipulation.

1. Principle: Polymorphism

Problem: How should the application handle the behaviour of different runtime type of similar elements?

Solution: In this application, Polymorphism is applied in the definition of structures, critters and tiles. Each individual previously mentioned type is designed to have runtime variations, or subtypes. For example, subtypes implementing the Structure interface, must provide custom overridden definitions of upgrade(), damage() and inspect() methods. These methods are defined differently for each subtype based on given relevant criteria, although they result in similar final behaviour. Such approach allows for low coupling between subsystems and overall high system cohesion.

### Pattern Used

1. Pattern: Singleton

Problem: How to ensure that only one instance of UGameView is ever created?

Solution: Apply the Singleton pattern on UGameView. Using this pattern, UGameView is implemented in such a way that all attempts to create a new instance of it either return a new instance if none has been previously initialized or return the very same existing instance. In such a way, a maximum of only one element of UGameView is ever used.

1. Pattern: Decorator

Problem: How to add behavior to an individual instance of Structure during runtime of the application, without affecting the behavior of other objects of the same type?

Solution: Create a DecoratorStructure class which is a subtype of Structure and in which a ‘decoratee’, an instance of Structure to be decorated, is maintained. When initializing an instance of DecoratorStructure (later referred to as DS), its ‘decoratee’ is defined as an instance of any given Structure or a subtype of Structure. When used during runtime, a DS will have exactly the same behaviour as the decoratee instance, except that it will also perform some extra ‘decorated’ behaviour. This is achieved by following the logical steps described below:

[DS is called to perform a specific method ‘m’ - ds.m()]

1. DS calls its decoratee to perform ‘m’ - decoratee.m()
2. DS performs an extra action as defined by its implementation of the m() method

In such a way, a behaviour is added to a decoratee instance without affecting the decoratee.

# Dynamic Behaviour: Sequence Diagrams

# (System Logic…) Design

# User Interface Design

## Description of the User Interface

In order to implement User Interface of the current application, Java SWING library was adopted. This allows the system to produce powerful visuals with high hardware performance. SWING contains convenient and easy to use implementations UI components such as of buttons, panels, fields, frames, etc. In the case of our application, JFrame, JPanel, JButton and JComponent were largely used.

## Visual Presentation of Graphical User Interface

An example of Graphical User Interface (GUI) can be observed on the figure below.

# References

**Daniel Sinnig PhD** Lecture Slides, ECSE-321. McGill University Winter 2015

**Martin Fowler** UML Distilled: A Brief Guide to the Standard Object Modelling Language