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](#_Toc415772817)

[1.1 Purpose of the document 2](#_Toc415772818)

[1.2 audience 2](#_Toc415772819)

[1.3 Scope 2](#_Toc415772820)

[1.4 Related documents 2](#_Toc415772821)

[2 System Overview 3](#_Toc415772822)

[3 Design Considerations 4](#_Toc415772823)

[3.1 Assumptions and Dependencies 4](#_Toc415772824)

[3.2 Constraints 4](#_Toc415772825)

[3.3 Goals and Guidelines 4](#_Toc415772826)

[3.4 Development Methods 4](#_Toc415772827)

[4 System Architecture 5](#_Toc415772828)

[4.1 Architectural Strategies 5](#_Toc415772829)

[4.2 Architectural Diagram 6](#_Toc415772830)

[5 Detailed System Design 7](#_Toc415772831)

[5.1 Component level design 7](#_Toc415772832)

[5.1.1 Presentation Layer 7](#_Toc415772833)

[5.1.2 Domain Logic Layer 8](#_Toc415772834)

[5.2 Object Oriented Principles and Patterns 9](#_Toc415772835)

[5.2.1 Principles Used 9](#_Toc415772836)

[5.2.2 Pattern Used 10](#_Toc415772837)

[6 Dynamic Behaviour: Sequence Diagrams 11](#_Toc415772838)

[7 (System Logic…) Design 12](#_Toc415772839)

[8 User Interface Design 13](#_Toc415772840)

[8.1 Description of the User Interface 13](#_Toc415772841)

[8.2 Visual Presentation of Graphical User Interface 13](#_Toc415772842)

[9 References 14](#_Toc415772843)

# Introduction

## Purpose of the document

## audience

## Scope

## Related documents

# System Overview

# Design Considerations

## Assumptions and Dependencies

## Constraints

## Goals and Guidelines

## Development Methods

# 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.

[INSERT DIAGRAM]

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’ which is common to DS and its decoratee]

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

In such a way, functionality is added to a decoratee instance without affecting its basic behaviour.

[INSERT DIAGRAM]

1. Pattern: Observer

Problem: How to handle runtime changes to game parameters with respect to updating Graphical User Interface (GUI)?

Solution: (This pattern was not implemented and what follows is an argument against its use)  
A ‘basic’ system-level event-based observer pattern (in which GUI would be updated only if some game parameter was changed) was judged to be ineffective for application in gaming environment of this scale and thus not implemented in current system. Instead, a slightly different approach was used to ensure that GUI displays recent information to the end user. In the current design, the system regularly updates GUI display with a predetermined frequency in order to provide the User with an updated visual representation of the game. This implies that all graphical elements are redrawn upon any given GUI update. This ensures that all game elements are updated and thus represent the most recently available game state. UGameView is the main handler and container of all components of the GUI and thus handles its own updates.

# 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.

[INSERT PICTURE]

# 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