System Specification Document (SSD)

For the Smurf Tower Defense game

1. Introduction

1.1 Purpose of this Document

This document outlines the software requirements for the Smurf Tower Defense game, developed as part of the university programming development course.

1.2 Scope of this Document

The document covers the requirements for a Java-based tower defense game using Java Swing, developed by a 6-person student team. The game will be a standalone desktop application running locally on Windows.

1.3 Overview

Smurf Tower Defense is a strategic tower defense game set in the Smurfs universe. Players defend their territory against waves of enemies using various Smurf characters as defensive towers.

2. General Description

2.1 Product Functions

- Strategic tower placement and upgrade system
- Wave-based enemy progression
- In-game economy for purchasing and upgrading towers
- Single-player gameplay on predefined maps
- Character progression through skill trees

2.2 User Characteristics

Primary users will be:

- Course instructors evaluating the project
- Fellow students testing the game
- Team members during development

2.3 Operating Environment

- Windows operating system
- Java Runtime Environment (JRE)
- Java Swing GUI framework
- Local file system for save data

3. Functional Requirements

3.1 Gameplay Mechanics

1. Tower Placement
    - Players must be able to place towers freely except for on enemy roads
    - Tower placement costs in-game currency
    - Maximum of one tower in one area, no stacking
2. Enemy Wave System
    - Progressive difficulty increase per wave
    - Predetermined enemy paths
    - Variable enemy types with different attributes
3. Economy System
    - Currency earned from defeating enemies
    - Bonus rewards for completing waves
    - Tower upgrade and selling mechanics

3.2 User Interface

1. Main Menu
    - New Game option
    - Level selection
    - Exit game option
2. In-Game Interface
    - Currency display
    - Wave information
    - Tower selection and upgrade menus
    - Health/Lives indicator

4. Interface Requirements

4.1 Software Interfaces

- Java Development Kit (JDK)
- Java Swing for GUI components
- Local file system for saving game state

4.2 User Interfaces

- Mouse-driven interface for tower placement and menu selection
- Keyboard shortcuts for common actions
- Java Swing windows and dialogs for game menus

5. Performance Requirements

5.1 Response Time

- Game should maintain playable frame rate on standard university computers
- UI actions should feel responsive and consequential

5.2 Capacity

- Support the maximum number of game objects required for largest wave
- Handle multiple towers and enemies simultaneously
- Manage sprite animations without significant lag

6. Design Constraints

6.1 Development Constraints

- Must use Java and Java Swing
- Must run on Windows
- Must be developed within course timeline
- Must follow course coding standards and OOP practices

6.2 Team Organization

- 6-person development team
- University course timeline and deadlines
- Version control usage (e.g., Git)
- Github organisation tools (Project board, tasks etc.)

7. Non-Functional Attributes

7.1 Code Quality

- Clear documentation
- Consistent coding style
- Modular design for testing

- Comments in code

7.2 Reliability

- Game should not crash during normal play
- Save game functionality
- Error handling for invalid user actions

8. Preliminary Schedule and Budget

8.1 Development Timeline

[To be completed based on course schedule]

8.2 Resources

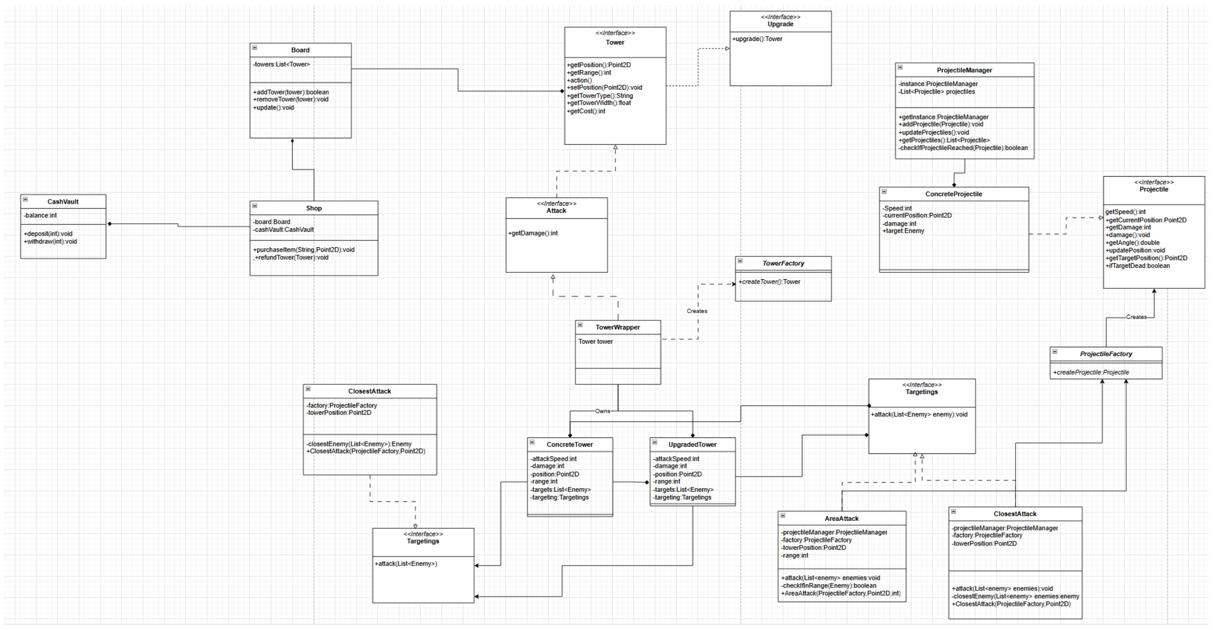
- 6-person student team
- University development environment
- Course-provided resources
- AI generative models

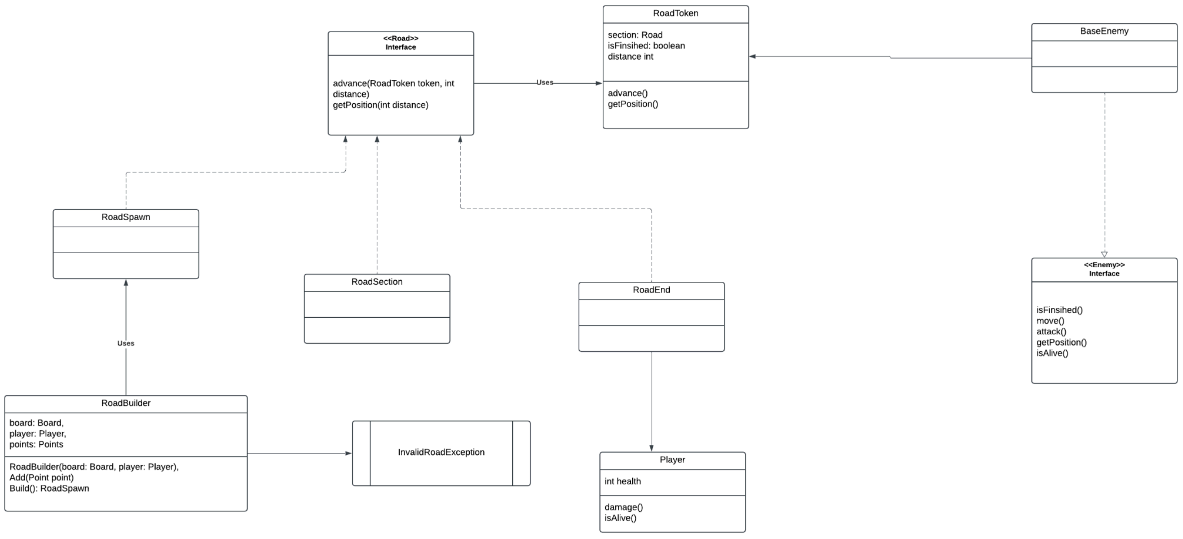
9. Appendices

9.1 Definitions

- TD: Tower Defense
- GUI: Graphical User Interface
- JRE: Java Runtime Environment

9.2 UML





Road

Road consists of a generalized linked list of road sections which is converted from an arbitrary length of 2d points by the builder. A RoadToken moves through the linked list oblivious of change of sections and position and only advances by wanted distance. It is notified when it arrives at the roadend.

Enemy

Handles the health, position dynamic through use of RoadToken, and reward of enemies. Interface is the main api towards other components.

Round

Road consists of a list of Round objects which in turn consists of RoundEvents which specify time and hold an Enemy Factory.

Cashvault

At the moment, the cash vault is given a list of dead enemies removed from the game loop. An improvement would be to make the cash vault an observer of enemy health instead.

Game

This is where the game loop subscribes to newly spawned enemies from the Round. It is an abstraction that handles the update of movement of alive enemies through delegation.

Board

Controls update of towers, and ensure valid position of them.

Shop

Mediates the interaction towerfactory, cashvault, and board. Propagates exception form cashvault and towerfactory for invalid placement or overdraft.

Tower Notifier

Mediates the subscription and unsubscription of towers to alive enemies to prevent dependency cycle between Board and Game. The reason for having two separate update handlers for enemy and towers is that we wanted to have towers that receive other input than enemies, e.g. boost nearby towers, and thus there need to be an explicit method call from the game loop on towers rather than just doing observer notifications.

Tower

Delegates attack modes through composition, and upgrades through a state pattern. Interface is the main api towards other components.