High Level Design

Jumping Tetris

version 1.0

07.12.2024

Mihael Brsanovic, Konstantin Köfler, Corinna Pucher, Julian Schmitt

Table of contents

1. Introduction

1.1. Purpose

The purpose of this High-Level Design (HLD) document is to provide a comprehensive architectural and functional framework for the game "Jumping Tetris." This document aims to translate the design principles and game mechanics detailed in the Game Design Document (GDD) into a cohesive system illustrated in UML Diagrams. By blending the iconic mechanics of Tetris with dynamic platforming, "Jumping Tetris" aims to deliver a challenging, strategy-driven experience for players, demanding precision, adaptability, and creativity.

1.2. System Overview

The system for Jumping Tetris is designed as a single-platform application optimized for Windows PCs. The game integrates dynamic platformer mechanics with Tetris-inspired puzzle gameplay, offering both single-player and local cooperative experiences.

2. Component Overview

2.1. Role of Component in the Overall System

Jumping Tetris uses the monolithic architecture approach, which consists of the following components:

- **Game Engine**: The game utilizes the Unity Engine as its core framework, integrating all components into a cohesive system.
- **User Interface (UI)**: Provides players with interactive elements, such as menus and HUD, for accessing game features, viewing scores and configuring settings
- **Application Logic**: Implements the core gameplay functionality, including Player actions, Tetris Architect mechanics, and level progression. It manages user requests and system behavior.
- **Data Layer**: Handles interactions with the file system for saving and retrieving data like high scores, settings, and in-game progress. Data is stored in structured files locally on the player's PC.

Monolithic applications may also include components like External Dependencies and Middleware. Since these are not part of our game's architecture, they are not addressed in this document.

2.2. Functionality

2.2.1. User Interface (UI)

- Displays interactive menus, such as Main Menu, Shop Screen, and Pause Menu, offering options for starting the game, configuring settings, and viewing high scores.
- Shows real-time gameplay feedback through HUD elements like the current score displayed during play.
- Facilitates intuitive navigation and seamless interaction with key game features, including menus, gameplay, and customization options.

2.2.2. Application Logic

- Processes user input to control The Cat's movement, allow the Architect to control Block Movement, and let the user interact with UI Elements.
- Handles real-time gameplay mechanics, including initializing and updating the Gameboard, spawning Tetris blocks, enemies and items, as well as scaling levels and enemy behaviors.
- Implements logic for rewards, item effects, penalties, and transitions between levels or game-over states.

2.2.3. Data Layer

- Retrieves high scores, player preferences, unlocked items and in-game currency from the database for display and gameplay use.
- Saves progress, unlocked items, and updated scores after each playthrough.
- Saves Data inside of a JSON File.

3. Detailed Design

3.1. Architecture

As stated before, the architecture of Jumping Tetris follows a monolithic design, where all components are integrated into a single application. This section details the interactions between key components needed to achieve the desired functionality and performance.

3.1.1. Component Connections

Game Engine as Central Hub:

- The Unity Engine acts as the primary coordinator, linking the UI, Application Logic, and Data Layer.
- It renders visual elements and processes interactions between game objects, such as falling Tetris blocks and player movements.
- The Engine also captures player inputs (e.g., in-game actions, button clicks) and forwards them to the Application Logic.

Application Logic ↔ UI :

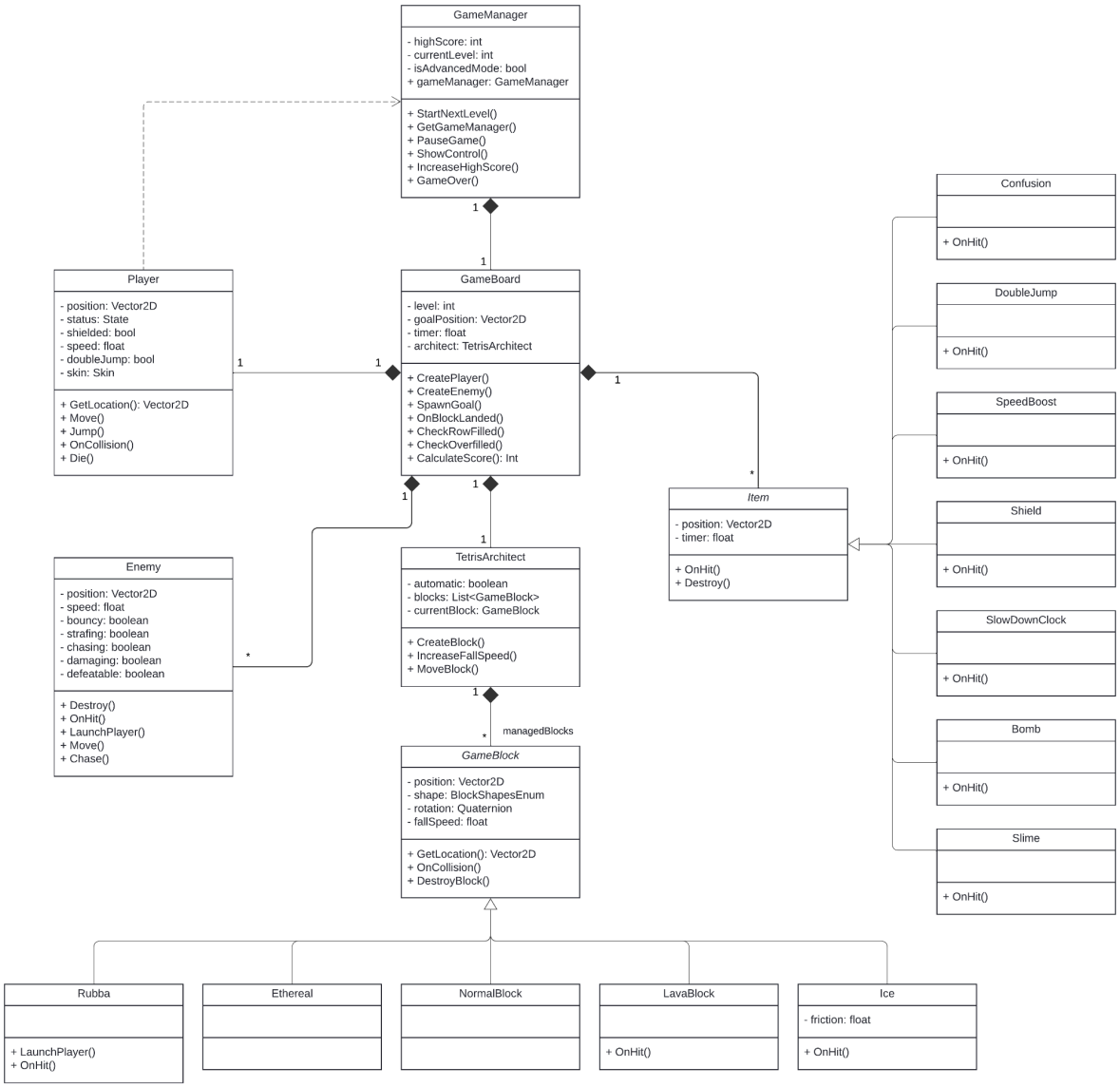
- The Application Logic processes gameplay data (e.g., player actions, block positions, score updates) and forwards the relevant outputs to the UI for display.
- Example: When the player earns points by clearing a level, the Application Logic calculates the updated score and passes it to the UI, which updates the HUD in real time.

Application Logic ↔ Data Layer:

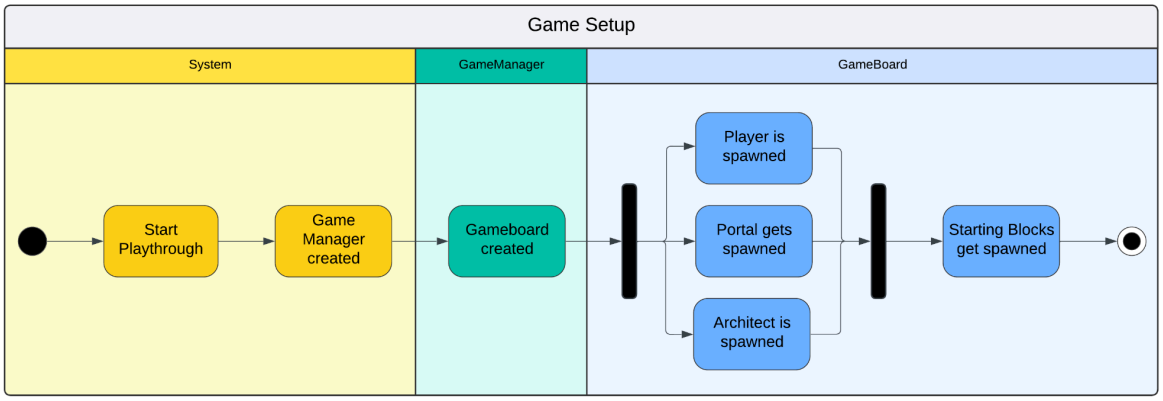
- The Application Logic accesses the Data Layer to retrieve saved player data (e.g., high scores, unlocked items) at the start of the game.
- During gameplay, updated data (e.g., new scores, in-game currency) is written to the Data Layer in JSON format.
- Example: After a game-over event, the Application Logic saves the player's updated score and earned currency.

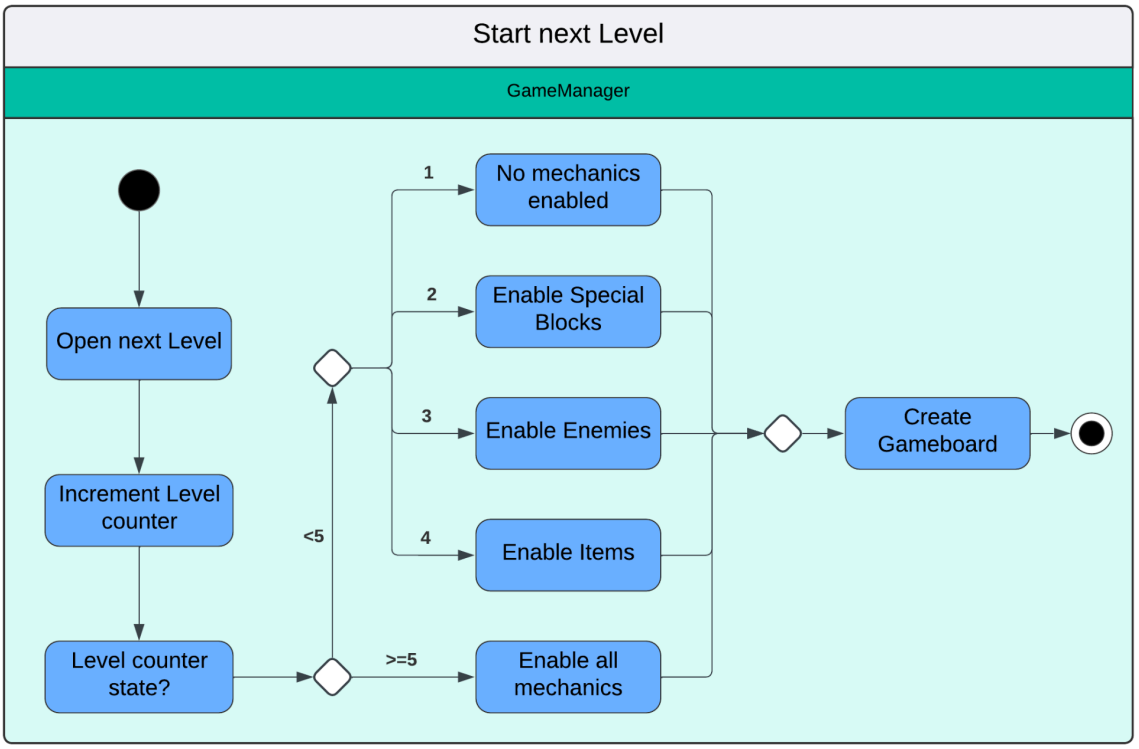
3.2. UML Diagrams

3.2.1. Class Diagram



3.2.2. Activity Diagram





3.2.3. Sequence Diagram