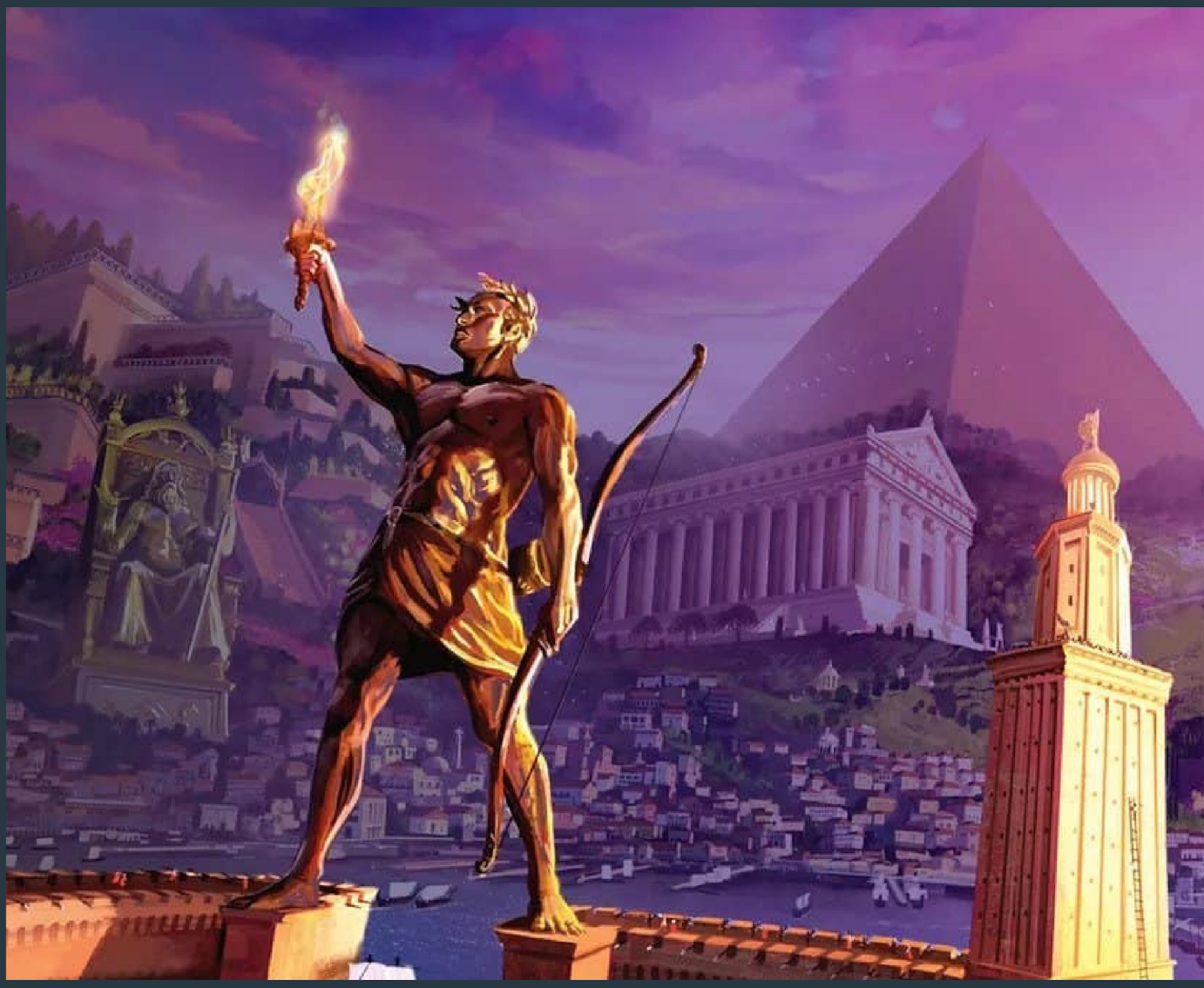
25/06/2023

7 Wonders

A Java project by BOARD BUSTERS



Doubabi Mustapha
Haddad Chirine
Martin--Dipp Daryl
Neveu Pierre
Pereira Guillaume
Zafizara-Benetrix Nathan

Introduction

Welcome to the Java documentation of our project, a digital adaptation of the board game ***Seven Wonders***. In this game, players lead one of the seven great cities of the Ancient World, striving to gather resources, develop trade routes, and build their city into a wonder of the world.

This project aims to provide an engaging digital rendition of ***Seven Wonders,*** implementing the game's rich strategy and quick pace in Java. We've carefully translated almost all of the game's rules and strategic options into this application while ensuring its compatibility from 3 to 7 players.

This documentation, and the class diagram (made on draw.io), serves as a guide for developers and users, elucidating the code's structure and the methods used to mirror the game's intricacies. Our design divides the code into key game components, such as resources, wonders, and trading between players. Whether you're a developer interested in the project's technical aspects or a user keen to explore the digital ***Seven Wonders***, this document will provide valuable insights.

Improvements since version 2

In this latest release of our Java project, we've introduced notable enhancements. New effects have been implemented which adds depth and interactivity to the game. We've revamped the user interface, providing a more user-friendly visual representation of game stages and their associated methods for an intuitive and appealing gaming experience. Furthermore, a major refactoring has been conducted to improve code efficiency and maintainability by restructuring the codebase, increasing readability and eliminating redundancy and unnecessary complexity.

New effects :

The Seven Wonders Java project has incorporated several new game mechanics with the addition of new effects, significantly enhancing the gameplay:

- **EffectGuildBuilders**: Players can earn victory points equivalent to the sum of the current stages of their own wonder and those of their neighbours.
- **EffectGuildDecorators**: If a player's wonder is at its final stage, they can secure additional victory points.
- **EffectGuildEarnVictoryPoints**: Players can earn victory points according to the number of a specific type of card that their neighbours have, bringing in a dynamic interaction with neighbouring players.
- **EffectGuildShipowners**: Players can earn victory points based on the number of their Raw Materials, Manufactured Goods, and Guilds cards, creating incentives for diversifying card types.
- **EffectEarnCoinAndVictoryPointFromCards**: Players can earn coins and, at the end of the game, victory points according to the number of a specific type of card they have built. This mechanic rewards both immediate and long-term planning.
- **EffectEarnCoinAndVictoryPointFromStages**: This effect grants players coins and, ultimately, victory points based on the number of stages they have built, thus rewarding the development of their wonders.
- **EffectEarnCoinsFromTypeOfCard**: Players can earn coins based on the number of a specific type of card that they and their neighbours have. This effect further incentivises strategic interactions with neighbouring players.

These additions infuse new strategic depth into the gameplay, fostering increased player engagement and more dynamic interactions.

Graphical User Interface

Our game's Graphical User Interface (GUI) was designed with an emphasis on simplicity, aesthetics, and clarity. We incorporated distinct clear action buttons and bright visuals for easy navigation. To enhance the gaming atmosphere, we generated our cards images with an AI (Midjourney) we also integrated fun music and sound effects for existing games like Fortnite, Clash Royale or Clash of Clans, providing an entertaining and user-friendly gaming experience.

Here is an overview of the key classes involved in managing the GUI for the BoardBusters game:

- **BattleController**: Manages the game's battle displays and transitions. Key methods include displayBattles(GameManager gameManager) for setting up the battle visuals for each player, and goToNextAge(ActionEvent actionEvent) for progressing the game to the next age.

- **ChooseCardActionController**: Governs player actions during card selection. The displayActions(GameManager gameManager) method presents possible actions for the chosen card, while endTurn() handles the end of a player's turn and transitions the game display accordingly.

- **ChooseCardsController**: Handles card display and selection. It uses displayCards(GameManager gameManager) to show a player's cards, while the chooseCard(GameManager gameManager, Card chosenCard) method enables card selection and transitions to the "Choose Card Action" scene.

- **ChooseNumberOfPlayersController**: Allows users to set the game's player count and initiates the game. The increaseNumberOfPlayer(ActionEvent event) and decreaseNumberOfPlayer(ActionEvent event) methods allow user control over the player count, while createGame(ActionEvent event) initializes the game with the selected player count and transitions to the "Choose Player Names" scene.

- **ChoosePlayersNameController**: Manages the GUI where players input their names. It creates text fields for each player, validates inputs, and starts the game upon successful validation. Also, it initiates game music and transitions to the card selection scene.

- **EndGameController:** In charge of endgame display. It presents final player rankings, displays the winner, and provides a close window function. Upon game conclusion, this class plays a victory sound and proceeds to present rankings.

- **ControllerUtils**: A utility class for updating the game's GUI display. It manages the presentation of cards, player properties, game headers, and sound effects. It offers methods for displaying card details (displayCard), updating game headers (displayHeaders), and showcasing player details (displayPlayerProperties).

- **MusicPlayer**: A singleton class managing the game's music and sound effects. It uses a MediaPlayer instance to play music or sound effects, with setMusic and playMusic methods for managing music, and playSoundEffect for sound effects.

- **GameManagerFx**: Extends Application to represent the game application. The overridden start method sets up the game stage, opens files, sets the UI layout, initiates and plays menu music, sets game icon and stage scene, makes the window non-resizable, and displays the stage.

- **MainFx**: The main class that launches the GameManagerFx application. The main method uses Application.launch with GameManagerFx.class as the argument.

- **FileOpener**: Facilitates loading various game resources. It offers methods for opening and reading a file (openFile()), and loading images from different sources: game source (openImageFromSourceGame()), card source (openImageFromSourceCard()), and resources source (openImageFromSourceResources()), which streamline resource management in the game.

Refactoring

The recent refactoring efforts for BoardBusters have introduced several significant enhancements to the code structure that greatly improve clarity, maintainability, and future development efficiency. These include:

- **Introduction of Enums**: The creation of Enums (CardType, EffectType, EndTurnEvent, PlayerAction) have added a more precise and efficient way to represent various game concepts, increasing code readability and reducing possible errors.

- **Addition of the BuyNeed Class**: The new BuyNeed class encapsulates the concept of a player's purchase requirement, providing a clearer and more organized representation of this game logic.

- **Renaming of Stage to WonderStage:** To avoid confusion with JavaFX's Stage class, the Stage class in BoardBusters was renamed to WonderStage, further eliminating ambiguity within the code.

- **Package Refactoring**: The restructuring of packages to have unique class names under the namespace "io.github.oliviercailloux.boardbusters" has increased code clarity and reduced the likelihood of name clashes, thereby streamlining development efforts.

- **Migration from Map to Multiset for Resources:** This change has simplified resource management within the game, enhancing performance and code readability.

- **Addition of Loggers**: The introduction of loggers has enhanced error tracking and debugging capabilities, aiding developers in maintaining and improving the game's codebase.

Potentials improvements

Potential avenues for improving the project have been identified based on the aspects that have not yet been fully implemented. These improvements could enhance the gameplay and user experience:

- **Multi-Neighbor Trading**: As of now, the game lacks the implementation of multi-neighbor trading, which allows a player to borrow resources from two different neighbors simultaneously. Incorporating this feature would add a strategic depth to the game and open up new tactical possibilities for players.

- **Variation in Cards**: Currently, the cards in the game do not exhibit a high degree of variation. Introducing more diverse cards, with different effects and capabilities, could enrich the gameplay and make each game session more unique and unpredictable.

- **Implementation of Wonder Phases B**: The 'B' phases of wonders are not yet implemented in the game. Adding these phases would not only increase the complexity and richness of the game, but it would also provide players with more options and strategies to explore.

- **Choice of Resources on Cards:** Another missing feature in the current implementation is the ability for some cards to offer a choice between two resources to produce (EffectGuildScientists). In the original board game, some cards provide players with the strategic decision to choose between two different resources that the card can produce.

- **Visibility of Other Player's Information**: One weakness of the current user interface is that players cannot see the information of other players during their turn. Enhancing the interface to display this information could improve strategic decision-making and add a layer of competition and interaction among players.

Each of these improvements would bring the game closer to a more comprehensive and engaging representation of the original board game, creating a richer and more enjoyable user experience.

Individual contributions

What did each member brought to the projet ?

The whole group participated in the structure of the code. Its writing is the result of our brainstorming. Here are everyone's contributions.

- **Daryl MARTIN-DIPP**

Unit testing & Code Reviewer

- **Mustapha DOUBABI**

Cards, Documentation writing, UML & Presentation

- **Guillaume PEREIRA**

Sounds effects, Player, GameManager, Javadoc & Documentation

- **Nathan ZAFIZARA-BENETRIX**

Implemented GUI, General Reviewer, Object enhancement and adapting to GUI, Battles, Wonder and WonderStage, Multiset resources Migration & Logger

- **Chirine HADDAD**

Objects design for GUI (cards and logo), UML & Class logics

- **Pierre NEVEU**

Effects, Objects initializing & Chained cards