Concordia University

Department of Computer Science and Software Engineering

Advanced Programming Practices

SOEN 6441 - Fall 2023

Group Name: W4
Submitted to: Prof. Amin Ranj Bar

Members:
Sneh Satishkumar Patel - 40264053
Romit Manishkumar Patel - 40273551
Yash Subhashbhai Chhelaiya - 40257682
Rushi Hasubhai Donga - 40269583
Abdul Sameer Jani Syed - 40272408
Mayank Nilkanth Parmar - 40269385

Risk Computer Game

A developed program which is compatible with the rules and map files and the command-line play of the “Warzone” version of Risk, which can be found at: https://www.warzone.com/.

About Risk Game

- A Warzone game setup consists of a connected graph map representing a world map, where each node is a country and each edge represents adjacency between countries.
- Two or more players can play by placing armies on countries they own, from which they can attack adjacent countries to conquer them.
- The objective of the game is to conquer all countries on the map.

Goals of Build 3

- Build 3 is a critical step in realizing the project's vision.
- **Strategic Refactoring:**
  - Emphasizes refactoring practices to enhance code maintainability and readability.
  - Introduces the Adapter and Strategy patterns for efficient file handling and dynamic player behaviors..
- **Map Editing and Management:**
  - Prioritizes the implementation of map save/load functionality.
  - Focuses on refining map editing capabilities, supporting multiple game map formats.
- **Dynamic Player Strategies:**
  - Highlights the strategic refactoring of the Player's issueOrder() method using the Strategy pattern.
  - Introduces diverse computer player behaviors to enhance user-driven player experiences.

Goals of Build 3

- **Advanced Game Features:**
  - Showcases the introduction of Tournament Mode for automated gameplay.
  - Implements game save/load functionality during ongoing sessions, along with startup, reinforcement, and order creation phases.
- **Importance of Incremental Code Build:**
  - Stresses the significance of incremental builds in achieving project goals.
  - Facilitates demonstrable progress, reduces development risks, and enhances collaboration and client engagement.
- In Summary, Build 3's primary goal centers on strategic development practices, emphasizing efficient code refactoring with the Adapter and Strategy patterns. This approach aims to enhance code quality, support dynamic player strategies, and implement advanced game features, ensuring a methodical and purposeful progression in achieving project objectives.

CODING STANDARDS

Code Layout

- **Indentation:**
  - Used 4 spaces for indentation.
  - The body of loop and conditional statements are indented in relation to their starting lines.
  - Body of a function is appropriately indented beneath its header.
- **Line Spacing:**
  - Blank lines serve the purpose of creating separation within code.
  - They are inserted between class declarations, methods, and significant segments within complex functions to enhance code readability.
- **Positioning of curly braces:**
  - The position of curly braces involves placing an opening curly brace immediately after the statement preceding it, thereby reducing the length of the code.
- **Spacing around operators and operands:**
  - Inserting spaces before and after operators to enhance the legibility of code.

MapHelper.java

```java
/**
 * Helper method to add/remove a new continent to the game map.
 *
 * @param p_mapFileName Name of map file.
 * @return Edited game map
 */

public GameMap editMap(String p_mapFileName) {
    String l_filePath = Constant.MAP_PATH + p_mapFileName;
    this.d_gameMap = new GameMap(p_mapFileName);
    File l_file = new File(l_filePath);

    if (l_file.exists()) {
        System.out.println(p_mapFileName + " map file exists. You can edit it.");
        this.readMap(l_filePath);
    } else {
        System.out.println(p_mapFileName + " does not exist.");
        System.out.println("Creating a new Map named: " + p_mapFileName);

        this.d_gameMap = new GameMap(p_mapFileName);
    }
    return this.d_gameMap;
}
```

Guidelines for Naming

- **Classes:**
  - Class names follow the convention of being written in UpperCamelCase, also known as Pascal Case.
  - Example : class MyClassName { }
- **Method Parameters, Member functions and Data Members:**
  - These are written in lowerCamelCase.
  - For method parameters, we use **‘p_’** as prefix and for data members, we use ‘**d_’** as prefix.
  - Example : int d_mapIndex = 1, public GameMap loadMap(String p_mapFileName)
- **Local Variables:**
  - Adheres to lower camelCase with the prefix **‘l_’** to indicate that it is a local variable.
  - Example : String l_line;
- **Constants:**
  - Uppercase letters with underscores.
  - Example : public static final String MAP_PATH = "src/main/resource/map/";

Guidelines for Writing Comments

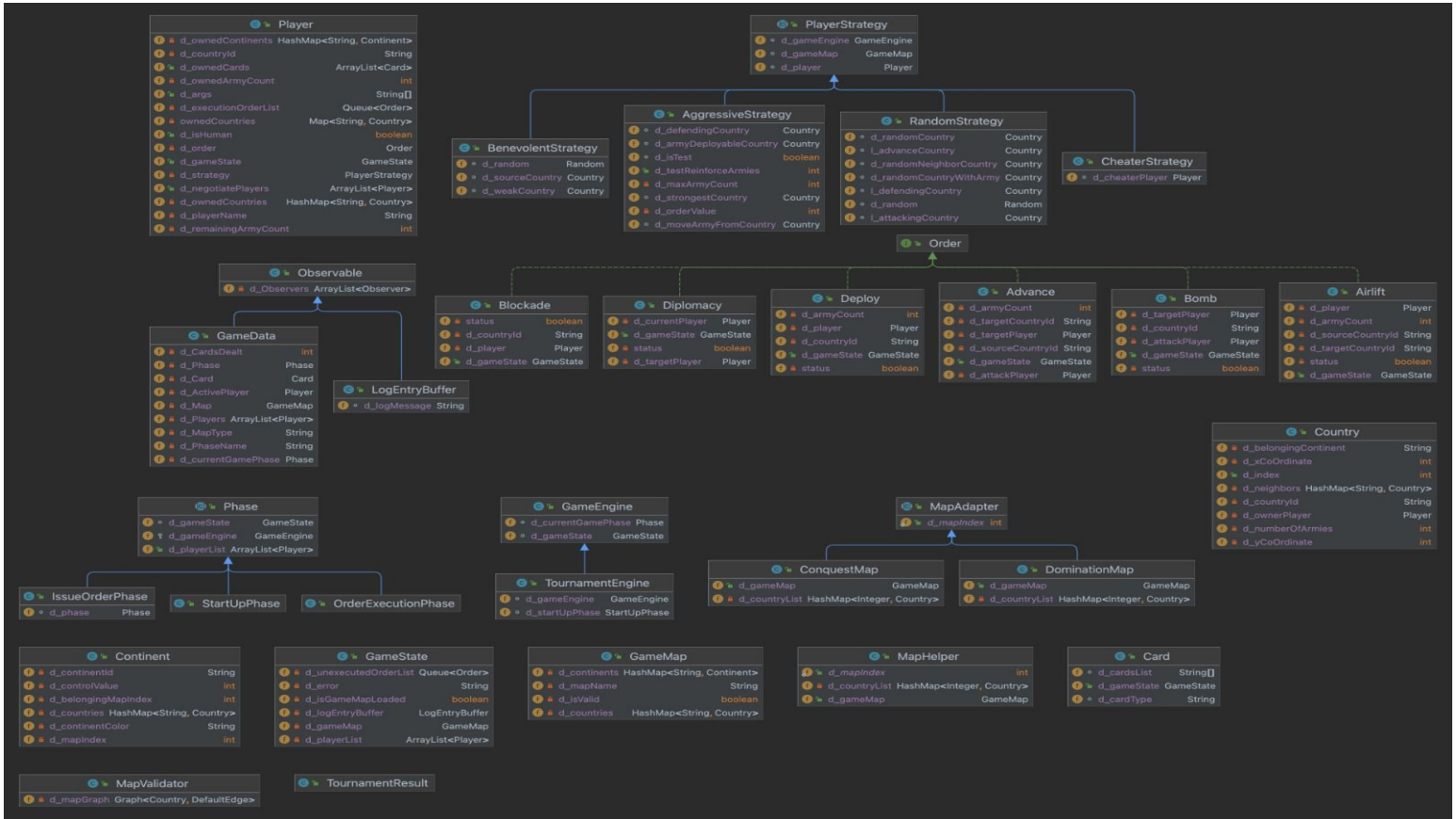
- **Code without any commented-out sections:**
  - Excess commented code is removed to enhance clarity and readability.
- **Documentation comments (JavaDoc):**
  - For each class, data member, and member function, there will be JavaDoc documentation provided above them.
- **In Complex Methods :**
  - In lengthy and complex methods, explanatory content is inserted to improve code comprehension.

StartUpPhase.java

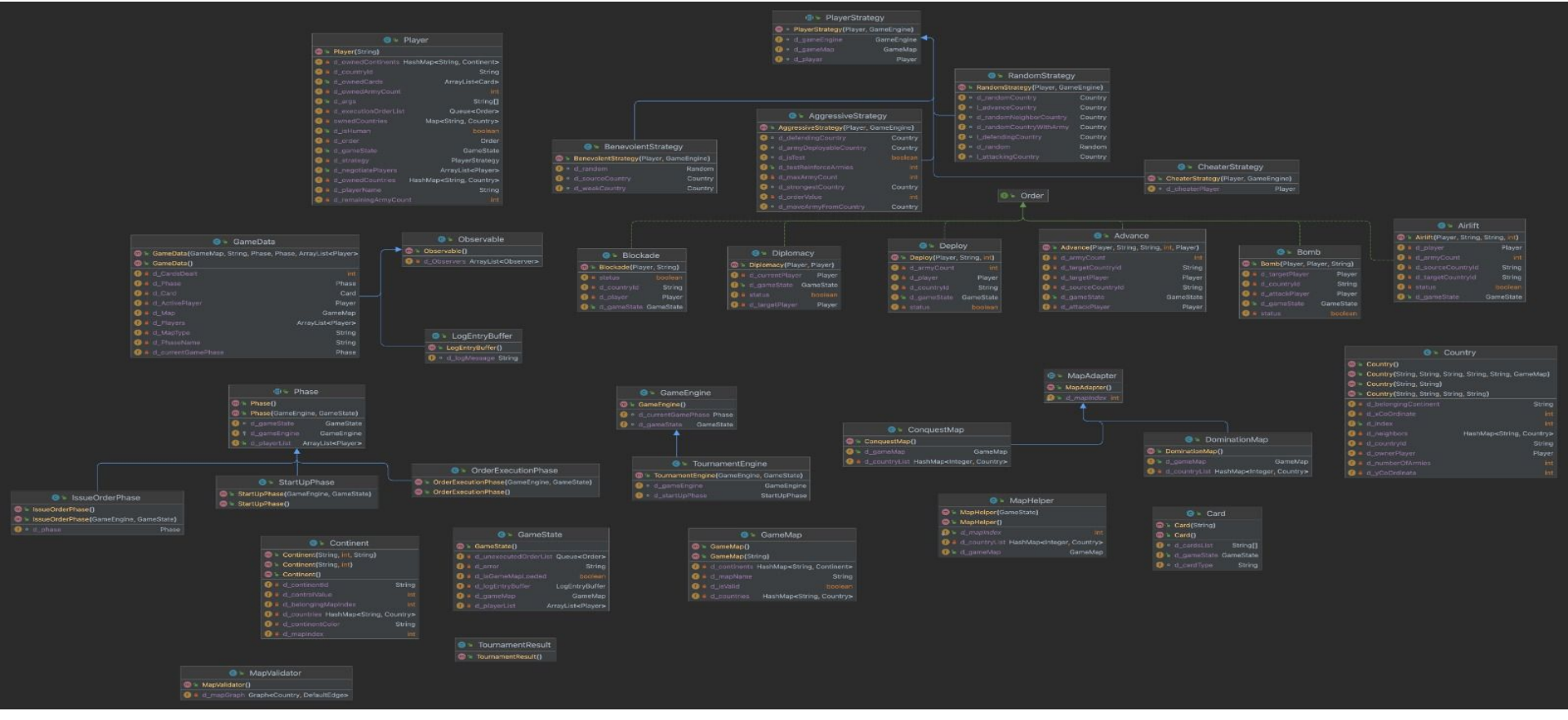
```java
/**
 * Getter method for player list.
 *
 * @return Returns list of game player.
 */
public ArrayList<Player> getPlayerList() {
    return this.d_playerList;
}

/**
 * Setter method for game phase.
 *
 * @param p_gamePhase GamePhase to set
 */
public void setGamePhase(Phase p_gamePhase) {
    this.d_gamePhase = p_gamePhase;
}
```

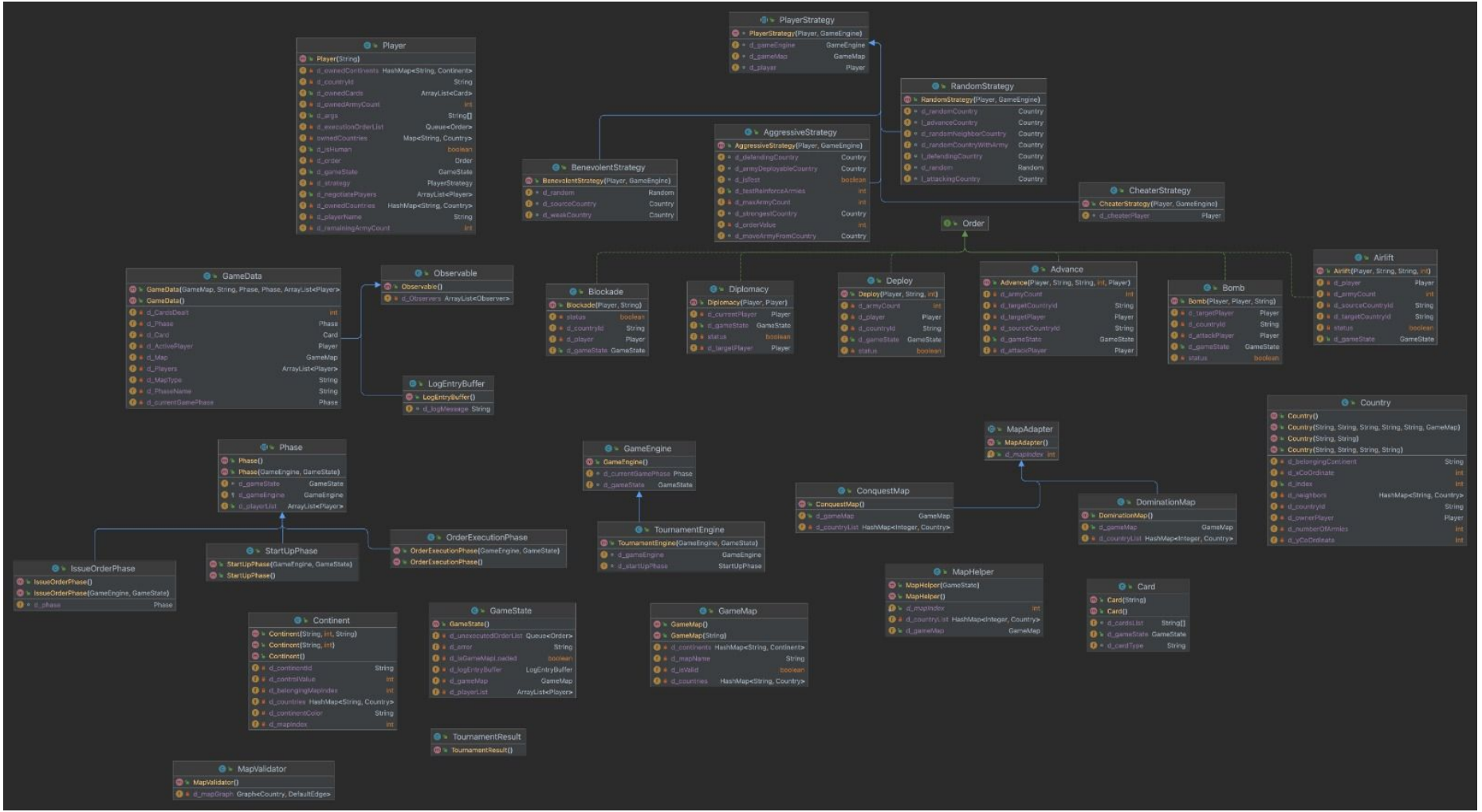
Architecture Diagram

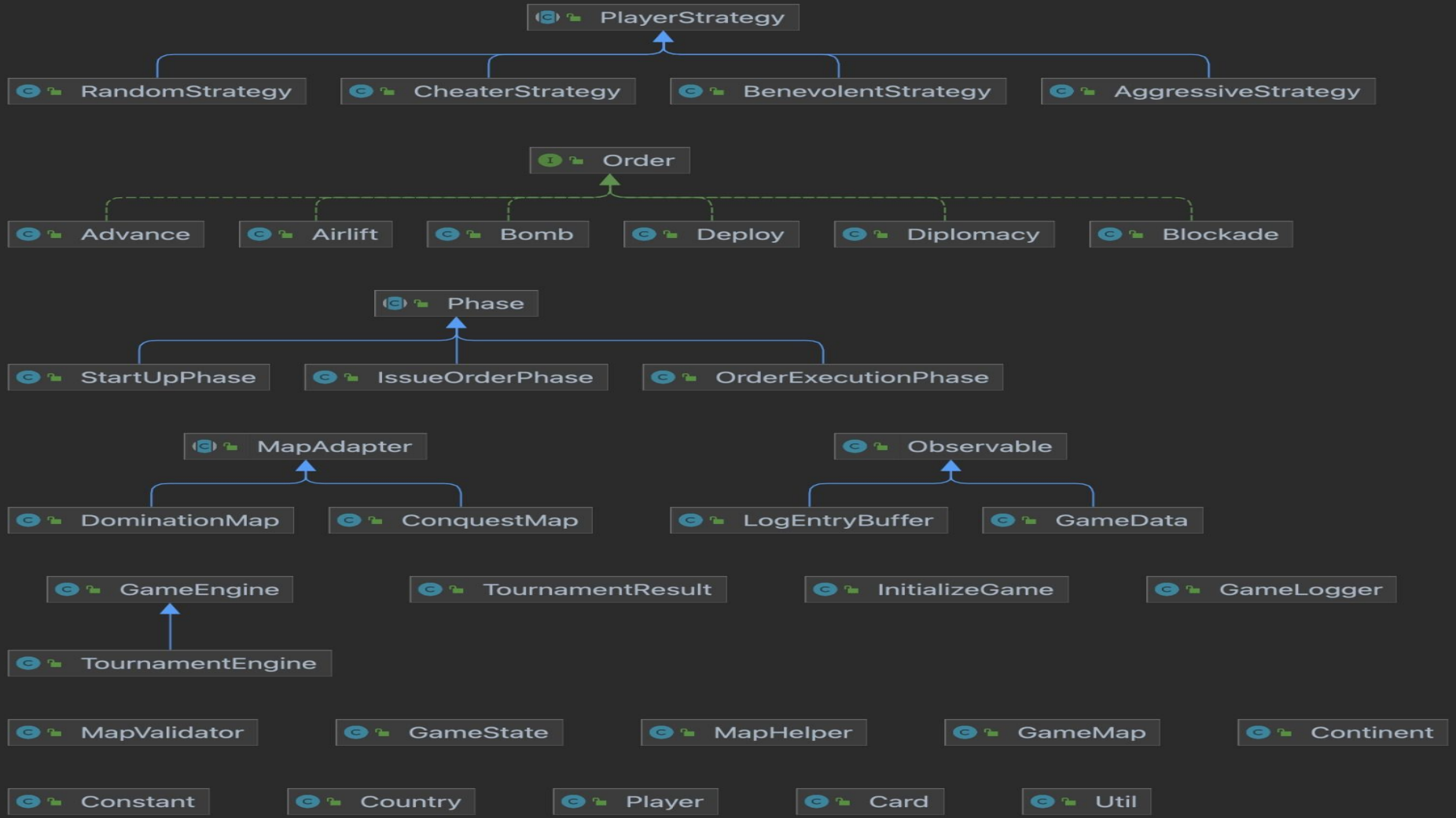
Functional Diagram



Class Diagram




Player
Player(String)
d_ownedContinents HashMap<String, Continent>
d_countryId String
d_ownedCards ArrayList<Card>
d_ownedArmyCount int
d_args String[]
d_executionOrderList Queue<Order>
ownedCountries Map<String, Country>
d_isHuman boolean
d_order Order
d_gameState GameState
d_strategy PlayerStrategy
d_negotiatePlayers ArrayList<Player>
d_ownedCountries HashMap<String, Country>
d_playerName String
d_remainingArmyCount int
PlayerStrategy
PlayerStrategy(Player, GameEngine)
d_gameEngine GameEngine
d_gameMap GameMap
d_player Player
RandomStrategy
RandomStrategy(Player, GameEngine)
d_randomCountry Country
i_advanceCountry Country
d_randomNeighborCountry Country
d_randomCountryWithArmy Country
i_defendingCountry Country
d_random Random
i_attackingCountry Country
AggressiveStrategy
AggressiveStrategy(Player, GameEngine)
d_defendingCountry Country
d_armyDeployableCountry Country
d_isTest boolean
d_testReinforceArmies int
d_maxArmyCount int
d_strongestCountry Country
d_orderValue int
d_moveArmyFromCountry Country
BenevolentStrategy
BenevolentStrategy(Player, GameEngine)
d_random Random
d_sourceCountry Country
d_weakCountry Country
CheaterStrategy
CheaterStrategy(Player, GameEngine)
d_cheaterPlayer Player
Order
GameData
GameData(GameMap, String, Phase, Phase, ArrayList<Player>
GameData()
d_CardsDeal int
d_Phase Phase
d_Card Card
d_ActivePlayer Player
d_Map GameMap
d_Players ArrayList<Player>
d_MapType String
d_PhaseName String
d_currentGamePhase Phase
Observable
Observable()
d_Observers ArrayList<Observer>
LogEntryBuffer
LogEntryBuffer()
d_logMessage String
Blockade
Blockade(Player, String)
status boolean
d_countryId String
d_player Player
d_gameState GameState
Diplomacy
Diplomacy(Player, Player)
d_currentPlayer Player
d_gameState GameState
status boolean
d_targetPlayer Player
Deploy
Deploy(Player, String, int)
d_armyCount int
d_player Player
d_countryId String
d_gameState GameState
status boolean
Advance
Advance(Player, String, String, int, Player)
d_armyCount int
d_targetCountryId String
d_targetPlayer Player
d_sourceCountryId String
d_gameState GameState
d_attackPlayer Player
Bomb
Bomb(Player, Player, String)
d_targetPlayer Player
d_countryId String
d_attackPlayer Player
d_gameState GameState
status boolean
Airlift
Airlift(Player, String, String, int)
d_player Player
d_armyCount int
d_sourceCountryId String
d_targetCountryId String
status boolean
d_gameState GameState
Phase
Phase()
Phase(GameEngine, GameState)
d_gameState GameState
d_gameEngine GameEngine
d_playerList ArrayList<Player>
GameEngine
GameEngine()
d_currentGamePhase Phase
d_gameState GameState
MapAdapter
MapAdapter()
d_mapIndex int
Country
Country()
Country(String, String, String, String, String, GameMap)
Country(String, String)
Country(String, String, String, String)
d_belongingContinent String
d_xCoOrdinate int
d_index int
d_neighbors HashMap<String, Country>
d_countryId String
d_ownerPlayer Player
d_numberOfArmies int
d_yCoOrdinate int
ConquestMap
ConquestMap()
d_gameMap GameMap
d_countryList HashMap<Integer, Country>
DominationMap
DominationMap()
d_gameMap GameMap
d_countryList HashMap<Integer, Country>
IssueOrderPhase
IssueOrderPhase()
IssueOrderPhase(GameEngine, GameState)
d_phase Phase
StartUpPhase
StartUpPhase(GameEngine, GameState)
StartUpPhase()
OrderExecutionPhase
OrderExecutionPhase(GameEngine, GameState)
OrderExecutionPhase()
TournamentEngine
TournamentEngine(GameEngine, GameState)
d_gameEngine GameEngine
d_startUpPhase StartUpPhase
MapHelper
MapHelper(GameState)
MapHelper()
d_mapIndex int
d_countryList HashMap<Integer, Country>
d_gameMap GameMap
Card
Card(String)
Card()
d_cardsList String[]
d_gameState GameState
d_cardType String
Continent
Continent(String, int, String)
Continent(String, int)
Continent()
d_continentId String
d_controlValue int
d_belongingMapIndex int
d_countries HashMap<String, Country>
d_continentColor String
d_mapIndex int
GameState
GameState()
d_unexecutedOrderList Queue<Order>
d_error String
d_isGameMapLoaded boolean
d_logEntryBuffer LogEntryBuffer
d_gameMap GameMap
d_playerList ArrayList<Player>
GameMap
GameMap()
GameMap(String)
d_continents HashMap<String, Continent>
d_mapName String
d_isValid boolean
d_countries HashMap<String, Country>
TournamentResult
TournamentResult()
MapValidator
MapValidator()
d_mapGraph Graph<Country, DefaultEdge>


PlayerStrategy
RandomStrategy
CheaterStrategy
BenevolentStrategy
AggressiveStrategy
Order
Advance
Airlift
Bomb
Deploy
Diplomacy
Blockade
Phase
StartUpPhase
IssueOrderPhase
OrderExecutionPhase
MapAdapter
Observable
DominationMap
ConquestMap
LogEntryBuffer
GameData
GameEngine
TournamentResult
InitializeGame
GameLogger
TournamentEngine
MapValidator
GameState
MapHelper
GameMap
Continent
Constant
Country
Player
Card
Util

Implementation Highlights

- **Refactoring Operations Post Build #2:**
  - Applied targeted refactoring operations based on identified potential targets.
  - Demonstrates a proactive approach to code improvement, ensuring the project evolves with a focus on long-term maintainability and scalability.
- **Code Refactoring with Adapter Pattern:**
  - Refactored codebase to utilize the Adapter pattern.
  - Enables the application to seamlessly read/write from/to a file using the "conquest" game map format. Allows dynamic switching between "domination" and "conquest" file readers.
- **Dynamic Player Strategies with Strategy Pattern:**
  - Strategically refactored the Player's issueOrder() method using the Strategy pattern.
  - Facilitates the implementation of diverse computer player behaviors, enhancing the player's experience through strategic decision-making.

Implementation Highlights

- **Advanced Game Features Integration:**
  - Introduced Tournament Mode for automated gameplay.
  - Allows users to engage in tournament-style gameplay with minimal intervention, showcasing the project's sophistication.
- **Game Save/Load Functionality:**
  - Enabled users to save and load games during ongoing sessions.
  - Enhances the user experience by providing a crucial feature for preserving game progress and facilitating a seamless gaming experience.
- **Architectural Design Refinement:**
  - Ensured the architectural design aligns with project goals.
  - Improves the clarity and cohesion of modules, incorporating design patterns such as State, Command, Observer, Adapter, and Strategy.

Implementation Highlights

- **Continuous Integration and Versioning:**
  - Established a well-populated versioning repository with at least 75 commits.
  - Ensures a stable and well-documented development history, promoting collaboration and allowing for efficient tracking of code changes.
- **Unit Testing Framework:**
  - Developed relevant test cases, including map validation, startup phase checks, order execution scenarios, and tournament mode.
  - Validates the correctness of the implementation and ensures robustness in handling various scenarios.
- **Coding Standards Adherence:**
  - Maintained consistent coding conventions, including naming conventions, code layout, and commenting practices.
  - Enhances code readability, maintainability, and promotes a uniform coding style across the project.

Functional Requirements

- **Purposeful Development:**
  - Strategic refactoring to enhance code maintainability.
  - Implementation of critical map editing and management features.
  - Dynamic player strategies for an engaging gaming experience.
  - Integration of advanced game features such as Tournament Mode.
- **Code Refactoring with Adapter Pattern:**
  - Refactor codebase to incorporate the Adapter pattern.
  - Seamless integration of "conquest" game map format.
  - Dynamic switching between "domination" and "conquest" file readers.
  - Improved file handling for map save/load functionality.
- **Map Save/Load Functionality:**
  - Showcasing the capability to save maps in different formats.
  - We will demonstrate loading of existing maps for editing or creating new ones.
  - User-friendly options for choosing file formats during save operations.

Functional Requirements

- **Strategic Implementation:**
  - We will Clearly demonstrate how the Adapter pattern improves code efficiency.
  - Visual representation of the codebase showing seamless file format switching.
  - Emphasize the purpose of the refactoring process in meeting functional requirements.
- In Summary, The Functional Requirements Demonstration showcases strategic code refactoring through the Adapter pattern, enhancing file handling for map save/load functionality. This implementation ensures seamless integration of different game map formats, offering users a user-friendly and dynamic experience in managing and interacting with maps.

Refactoring

- In Refactoring, we restructured and improved the existing code without changing its external behavior, And it became a crucial step after the completion of Project Build 2.
- **Identifying Potential Refactoring Targets:** Identifying potential refactoring targets within the existing codebase was a methodical process that aimed to enhance code quality and maintainability. The following steps were undertaken: Comprehensive Code Review, Collaborative Approach, Code Complexity,Error-Prone Areas.
- **Continuous Integration Workflow:**
  - Acknowledged the importance of a robust versioning and integration workflow.
  - Ensured a well-populated versioning repository with consistent commits and tagging for each build.
  - Improved collaboration, version tracking, and code stability.

Refactoring

- **Adapter Pattern Integration:**
  - Recognized the need for enhanced file handling to support both "domination" and "conquest" map file formats.
  - Applied the Adapter pattern to create a unified interface for reading different file formats.
  - Improved flexibility and maintainability in file handling operations.
- **Strategy Pattern for Player Behaviors:**
  - Identified the need for diverse player behaviors and strategies.
  - Utilized the Strategy pattern to refactor the Player's issueOrder() method.
  - Enhanced modularity and extensibility by encapsulating varying player behaviors.
- **Map Save/Load Enhancement:**
  - Acknowledged the importance of efficient map management and editing.
  - Enhanced map save/load functionalities to support different file formats.
  - Improved user experience with seamless map handling and editing capabilities.

Refactoring

- **Tournament Mode Implementation:**
  - Recognized the need for an automated tournament mode.
  - Integrated a new game mode with minimal user interaction for tournament-style gameplay.
  - Introduced an advanced feature while maintaining the overall system integrity.
- **Unit Testing Framework Enhancement:**
  - Expanded the unit testing framework with relevant test cases for new functionalities.

Testing and Quality Assurance

- **Comprehensive Unit Testing:** Unit testing plays a pivotal role in ensuring the reliability and correctness of our project. It involves subjecting individual code components, modules, and classes to a battery of tests to verify their functionality and behavior.
- **Validation of Functional Requirements:** Our unit tests serve as concrete proof that we've met the project's functional requirements. We can point to specific test cases and results that demonstrate how our code fulfills the specified criteria. For instance, we can showcase tests that validate map editing, gameplay, and startup phases.
- **Quality Assurance Practices:** We've also embraced quality assurance practices during development. These practices include code reviews, pair programming, and adherence to coding conventions. Consistent coding standards contribute to code quality and maintainability.
- **Continuous Integration and Deployment (CI/CD):** Our project is future-ready, as we've considered integrating it into a CI/CD pipeline. These processes will automate the testing and deployment of our software, ensuring that changes are thoroughly validated and quickly deployed, streamlining our development workflows.
- In summary, comprehensive unit testing is the cornerstone of our quality assurance efforts, providing us with the confidence that our project meets its functional requirements and is well-prepared for future development phases and deployments.

Conclusion

- **Strategic Evolution:** Build 3 marks a strategic evolution in the project, focusing on advanced programming practices and systematic code enhancement.
- **Adapter Pattern Integration:** The incorporation of the Adapter pattern has significantly enhanced our file handling capabilities. Seamlessly switching between "domination" and "conquest" file formats reflects a strategic approach to meet diverse user needs
- **Player Behavior Strategies:** The application now boasts dynamic player behavior strategies, enhancing the gaming experience through the implementation of the Strategy pattern.
- **Tournament Mode Implementation:** The introduction of a Tournament Mode showcases our commitment to providing users with diverse and engaging gameplay options.
- **Map Save/Load Efficiency:** Users can now efficiently manage maps with improved save/load functionalities, supporting different file formats.
- In conclusion, Build 3 reflects not only a progression in coding practices but a step forward in delivering a comprehensive and high-quality software solution.

Conclusion

- **Challenges:**
  - Integrating the Adapter pattern for different file formats posed challenges in ensuring seamless integration without disrupting existing functionalities - Thorough testing and staged integration helped identify and address compatibility issues, ensuring a smooth transition.
  - Introducing the Tournament Mode presented challenges in orchestrating multiple games automatically without user interaction - Extensive testing and incremental development allowed us to fine-tune the logic and handle diverse scenarios within the tournament framework.
  - Refactoring the codebase to incorporate new design patterns could potentially introduce unforeseen bugs or break existing functionalities. - Rigorous testing, version control, and continuous integration played a pivotal role in identifying and rectifying any unintended consequences.

Despite these challenges, the team's commitment to innovation, collaboration, and adaptability has led to the successful execution of Build 3, laying a solid foundation for the project's continued evolution.

References

Warzone game

Domination game maps

As specified in the Project Build 3 document

Thank You