Kingdomino: Enter the Virtual World

Hai Duong, Tran
CSE, Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
hai.tran2@stud.fra-uas.de

Pham Minh Tuan, Bui
CSE, Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
pham.bui@stud.fra-uas.de

***Abstract*—This document provides a brief overview of our implementation of the board game "Kingdomino" and its associated algorithms. We discuss problem formulation, related work, and our teamwork structure. Our approach includes input/output techniques, a proposed algorithmic methodology, and implementation details with a graphical user interface. We present simulation results and statistical tests to evaluate performance under various conditions. Finally, we highlight the conclusions drawn from our development process and outline potential future work.**

***Index Terms*—Kingdomino, boardgame, implementation, game development, OOP**

I. Introduction

Kingdomino [1] is a strategic tile placement game where players act as lords expanding their kingdoms. The objective is to construct the most prosperous territory by selecting and placing tiles that represent various terrain types, such as wheat fields, lakes, and mountains. Each tile comprises two sections that must be connected to the existing kingdom based on matching terrain types.

A key mechanic in Kingdomino is the tile selection order, which is influenced by the quality of previous choices. High-value tiles offer strategic benefits but may result in players selecting tiles later in subsequent rounds, introducing tactical decision-making. Additionally, tiles with crowns multiply the value of the connected terrains, significantly impacting the final score.

The game concludes once each player fills their 5×5 grid or can no longer place tiles. Scoring is determined by the size of connected terrain groups and the number of crowns they contain. Kingdomino's blend of simple rules and strategic depth has garnered popularity, making it an excellent subject for developing and analyzing game-based algorithms and AI strategies.

II. Problem Description

The primary goal in Kingdomino is to construct the most prestigious kingdom by strategically placing tiles within a constrained 5×5 grid. Players must explore various terrain types—fields, lakes, mountains, forests, meadows, and swamps—and connect them to their existing kingdom while adhering to specific placement rules. Each tile consists of two sections, and at least one section must match the terrain type of an adjacent tile. Crowns on tiles act as multipliers to the score of connected terrain groups, encouraging players to prioritize valuable combinations.

Players compete for tiles through a selection mechanism that balances high-value tiles with the consequence of selecting later in subsequent rounds. This creates an optimization problem where players must maximize immediate gains while strategically positioning themselves for future moves. The game concludes when each player has filled their grid or can no longer place tiles, with scoring determining the winner based on terrain connectivity and crown placements.

A. Formal Description

The game involves the following components and constraints:

1) Components:

- **Tiles**: Each tile consists of two sections, each representing one of the six terrain types (fields, lakes, mountains, forests, meadows, and swamps). Certain tiles include crowns, which serve as score multipliers.
- **Kingdom**: A 5×5 grid where tiles are placed. Each player starts with a central tile (wild type) and builds outward.
- **Selection Order**: Players select tiles in descending order of tile value (higher numbers first) and use their selection to determine the order in subsequent rounds.

2) Rules:

- **Placement**:
  - Tiles must connect to at least one adjacent tile with the same terrain type (horizontally or vertically).
  - The grid is limited to 5×5 dimensions; any tile that cannot be placed is discarded.
- **Scoring**:
  - Points are calculated as the product of the number of connected tiles of the same terrain and the number of crowns within the connected group.
  - Bonuses include:
    - +10 points for placing the central castle at the grid's center.
    - +5 points for completing a full grid.
- **Game Variants**:
  - A 7×7 grid variant for advanced play.
  - Multi-round gameplay (Dynasty mode) with cumulative scores.

3) Objective: Maximize the total score by constructing a kingdom that balances large connected terrain groups with crown placements, while competing against other players for optimal tile selection.

B. Examples of Gameplay

Gameplay is explain in Appendix A.

III. Related Work

Board games like Carcassonne [2] and Patchwork [3] offer valuable insights into the mechanics of tile placement and scoring, making them relevant to our implementation of Kingdomino. Carcassonne challenges players to build cities, roads, and fields by placing tiles based on terrain type, a mechanic that closely aligns with Kingdomino's terrain-matching rules. Similarly, Patchwork emphasizes the efficient use of a constrained grid, much like Kingdomino's 5×5 board, where players must optimize placement for maximum scoring. These games demonstrate how simple mechanics can result in complex decision-making, a feature we aim to replicate in our project.

In addition to inspiration from traditional board games, the implementation of Kingdomino leverages several key computational and architectural techniques. The scoring mechanism, for instance, utilizes a flood-fill algorithm [4] to evaluate the connected terrain groups and their associated scores. This algorithm, commonly employed in image processing and graph traversal [5], provides an efficient way to traverse and calculate properties of contiguous regions. Its application in Kingdomino ensures accurate and performant scoring calculations, even as the board becomes increasingly complex during gameplay.

The game is designed using an event-driven architecture [4], [6], where interactions, where interactions such as tile placement and scoring updates are managed asynchronously through an EventManager. This approach is prevalent in modern game frameworks and engines, such as LibGDX, Unity, and Unreal Engine, and ensures a clear separation between game logic and user input. By decoupling these components, the design achieves improved modularity and scalability, enabling future enhancements such as AI integration or multiplayer features. The intention was to implement Entity-Component-System (ECS) architecture [7] to further improve the game's performance and scalability. But we not strictly follow this architecture for flexibility and simplicity.

Furthermore, the aesthetic design of Kingdomino draws from pixel-art-inspired games, including Balatro [8], which use vibrant colors and minimalistic textures to create a visually engaging experience. This style has been adopted to maintain the simplicity of the board game while enhancing the digital adaptation with a modern and nostalgic visual appeal.

Finally, the use of the LibGDX framework [9] streamlines development, particularly in rendering, asset management, and input handling. By leveraging tools such as TextureAtlas [10] for managing game assets and InputMultiplexer for handling player interactions, the framework supports an efficient implementation of Kingdomino's mechanics and aesthetics. This integration aligns with industry-standard practices for creating scalable, interactive applications.

Incorporating these approaches and inspirations, the implementation of Kingdomino aims to balance simplicity, efficiency, and engagement, ensuring a faithful and enjoyable digital representation of the original board game.

IV. Teamwork

In general, the project was divided into two main parts: the game logic and the graphical user interface (GUI). We used GitHub Project to track and communicate with each other.

We ultilize SCRUM methodology to manage our project. For more details, see the GitHub Project board.

A. Roles and Responsibilities

For the game logic, Hai Duong Tran was responsible for almost all backend logic including the implementation of the tile placement rules, scoring mechanism, and the overall game flow. He also integrated the flood-fill algorithm for scoring and ensured the game logic adhered to the official Kingdomino rules. Moreover, he worked on shader to add an extra visual effect to the game.

Pham Minh Tuan Bui focused on the graphical user interface (GUI) and user interactions. He designed the layout, managed the rendering of game elements using the LibGDX framework, and implemented the event-driven architecture to handle user inputs and game state updates. Additionally, he worked on the visual aesthetics and ensured a smooth user experience.

Even though we had our own responsibilities, we often collaborated on various aspects of the project. We held regular meetings to discuss progress, brainstorm solutions to challenges, and ensure that our work was aligned. Code reviews were conducted to maintain quality and consistency, and we frequently pair-programmed to tackle complex problems together. This collaborative approach not only enhanced our productivity but also fostered a deeper understanding of the project as a whole.

B. Collaboration Tools and Workflow

GitHub Issues and Pull Requests to track bugs and feature requests. Each team member created branches for their respective tasks, ensuring that the main branch remained stable. Code reviews were conducted via Pull Requests, where we provided feedback and suggestions for improvements. We also used GitHub Actions for continuous integration, running automated tests to catch issues early.

For communication, we relied on Messenger for real-time discussions and meet each other at school or guest house for meetings. We documented our progress and decisions in a shared Google Drive, which included meeting notes, design documents, and brainstorming sessions. This collaborative approach ensured transparency and kept everyone aligned with the project's goals.

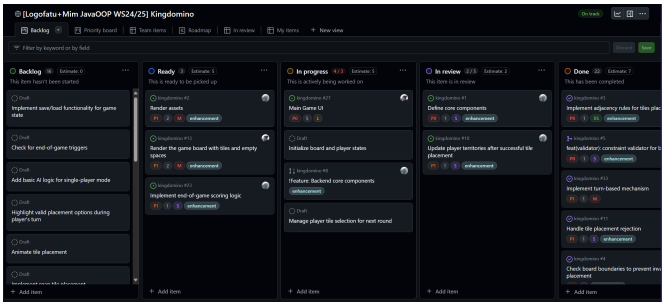

Fig. 1. Optional Process Flow Diagram (replace with your own).

V. Proposed Approaches

A. Input/Output Technique

B. Algorithm and Pseudocode

VI. Implementation Details

A. Application Structure

B. Graphical User Interface (GUI)

C. UML/Class Diagram

D. Used Libraries and Environment

E. Important Code Snippets

Fig. 2. UML/Class Diagram (replace with your UML figure).

```
// Pseudocode or short snippet
function calculateScore(board):
    score = 0
    for each territory in board:
        if territory is contiguous:
            score += territory.size * territory.crowns
    return score
```

VII. Experimental Results, Statistical Tests, Running Scenarios

A. Simulation Setup

B. Results and Evaluations

VIII. Conclusions and Future Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

A. Team Reflection

B. What We Learned

C. Future Development

Appendix A
Gameplay Mechanic

References

[1] Wikipedia contributors, "Kingdomino — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kingdomino&oldid=1243538299

[2] ——, "Carcassonne (board game) — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Carcassonne_(board_game)&oldid=1254238483

[3] ——, "Patchwork (board game) — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Patchwork_(board_game)&oldid=1212675183

[4] ——, "Flood fill — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/wiki/Flood_fill#Stack-based_recursive_implementation_(four-way)

[5] ——, "Graph traversal — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Graph_traversal&oldid=1250833600

[6] ——, "Event-driven architecture — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Event-driven_architecture&oldid=1262812523, 2024, [Online; accessed 19-January-2025].

[7] ——, "Entity component system — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Entity_component_system&oldid=1265223349

[8] ——, "Balatro (video game) — Wikipedia, the free encyclopedia," 2025, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Balatro_(video_game)&oldid=1270307614

[9] "Libgdx: A java game development framework," https://libgdx.com/.

[10] Wikipedia contributors, "Texture atlas — Wikipedia, the free encyclopedia," 2024, [Online; accessed 19-January-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Texture_atlas&oldid=1256529720