Assignment Report

SFWRENG 2AA4 (2026W)

Assignment 1

Author(s)

Alex Martinez	marta40@mcmaster.ca	400587889
Jalal Al-Sharif	alsharij@mcmaster.ca	400568628
Swar Thakar	thakas5@mcmaster.ca	400565632
Logan Sedore	sedorl1@mcmaster.ca	400566324

GitHub URL

`https://github.com/alephnull1678/settlers-of-catano`

February 13, 2026

1 Executive summary

Logan Sedore ▶*This assignment involved creating a design-focused program simulation for the settlers of the Catan board game. This implementation of the game required a system capable of manipulating players, resources, tiles, and building pieces while maintaining proper SOLID and OO design principles. With careful implementation using these principles, a UML diagram was created to capture the entities and relationships present in the game. This model was then translated into program code using interface-driven design and appropriate use of parent and child classes. Design decisions include the use of generic catalog abstractions for representing resource quantities, the SRP between entity classes, and the use of immutable snapshots to prevent unintended state mutations. A functioning simulation was successfully implemented from the UML design, capable of managing players, distributing resources, consuming building costs, and maintaining consistent game state transitions. This design prioritizes the correctness of program structure, ensuring the proper implementation of SOLID, OO, and GRASP principles.*◀

2 Requirements traceability

Req ID	Status	Implemented in	Design considerations
R1.1	Implemented	`Player`, `PlayerHand`, `MapPlayerHand`	Player entity encapsulates resources and building capabilities, delegating inventory handling to specialized components.
R1.2	Implemented	`Catalog<T>`, `MapCatalog<T>`	Generic catalog abstraction used to model quantities of resources and pieces uniformly, supporting extensibility and reuse.
R1.3	Implemented	`PieceHandler`, `Piece`, `Building`, `Road`, `Settlement`, `City`	Piece hierarchy models domain entities with specialization, enabling polymorphic handling of buildable structures.
R1.4	Implemented	`Board`, `HardWiredBoard`, `StaticBoard`, `Tile`, `Node`	Board topology is represented through tile and node relationships. A predefined configuration ensures deterministic simulator behaviour.
R1.5	Implemented	`Game`, `Dice`, `RegularDice`, `MultiDice`	Dice behaviour is abstracted through interfaces, allowing different dice implementations while maintaining rule-driven simulation.
R1.6	Implemented	`Validator`, `Action`	Validation logic is separated from gameplay execution to enforce rules independently and improve maintainability.
R1.7	Implemented	`Player.consumePiece`, `PlayerHand.removeHand`, `PieceHandler`	Building actions are treated as atomic transactions combining piece consumption and resource payment. Rollback ensures consistency if payment fails.
R1.8	Implemented	`Catalog.snapshot()`	Snapshot functionality provides read-only views of inventories, preserving encapsulation and preventing unintended mutation.
R1.9	Implemented	`Player.chooseAction()`	Randomized action selection supports autonomous simulator behaviour without requiring human interaction.
R1.10	Implemented	Full system integration	All domain entities interact to form a functioning simulation core consistent with Catan gameplay rules and constraints.

3 Design and domain modeling

Logan Sedore ▶*The program was designed based on the primary functions that exist within The Settlers of Catan and implemented using a UML diagram and object-oriented principles. SRP was heavily considered, and as such, Player, Tile, Node, and Piece were designed as separate components, each changing for only one primary reason. Resource and piece inventories are handled through the Catalog<T> abstract class. This was implemented so that different types of inventory could be managed without any primitive obsession. Furthermore, the MapCatalog implementation allows for a storage mechanism that does not violate any SOLID principles. The Piece interface handles all the different buildable structures, such as roads, settlements, and cities, and extends their common behaviour. The PieceHandler is associated with the Piece interface through composition and manages piece availability. The Dice interface has implementations RegularDice and MultiDice, allowing the system to adapt to different dice layouts without modifying the game logic. Furthermore, each action needs a validation check to see if the player can proceed with the action. This is done with the Validator class, keeping the rules in check and independent from the player. Building actions involve transactions of piece consumption and resource payment. If the validator claims an action to be invalid, the piece is returned to the inventory, ensuring the game state remains consistent. Overall, the design focuses heavily on SRP, OO, and object-oriented principles such as polymorphism and encapsulation to create a orginazed and modular program.*◀

4 Translating engineering models to program code

Logan Sedore ▶*As mentioned previously, the design was created and organised using a UML diagram. Each part of the game and its relationships, as identified in the model, were implemented into corresponding Java classes and interfaces. Abstract behaviours are demonstrated through the implementation of interfaces such as Catalogue, Dice, and Board, with their functionality being provided through classes like mapCatalog, RegularDice, and HardWiredBoard. This system created strong flexibility and easy extendability without modifying existing code significantly. Composition in the UML was implemented into the program through object interactions. For example, the Player class interacts with PlayerHand and PieceHandler to manage resources and building operations instead of creating a god class to handle all these entities. Inheritance relationships were made into structured class hierarchies. This is seen in the piece model, where specific structures such as roads and cities extend common behaviour. This allows buildable entities to be handled in a polymorphic way while still maintaining each components specialized functionality. Overall, the program structure and implementation follow the design of the UML diagram very closely, resulting in a clean and SOLID program.*◀

5 Using Generative AI

Swar Thakar ▶*Elaborate this phase. Follow the pointers in the assignment and share additional insights if applicable.*◀

6 Implementation

Logan Sedore ▶*Implementation of the game was done based on the previously mentioned UML diagram. The main functionalities of the game, including players, the board, tiles, nodes, and pieces, were translated directly into their corresponding classes to perform their role in the system. A large part of the game is resource and inventory management. This was implemented using the generic Catalog<T> abstraction, in conjunction with MapCatalong, allowing for polymorphic storing of such. Player resources are managed through PlayerHand, and the piece use and availability are managed by PieceHandler. The structures available for a player to place, such as roads, settlements, and cities, were implemented with a class hierarchy with classes extending the Piece abstraction. This allows for placable pieces to be handled properly while maintaining each piece's individual behaviour. Dice was implemented using the Dice interface and subclasses such as RegularDice and MultiDice, allowing proper resource distribution during gameplay. Another important component is the Validator, which was made to enforce gameplay rules independently of the player. Each action is checked against the class before execution to ensure a rule-following game. Building actions were implemented using a transactional system of resources. When a player decides to build, the system first checks if the piece is available, then takes the corresponding resources to build that entity. If the resource amount is insufficient to build the entity, the piece is returned to the catalog to maintain a consistent game state. The final system supports player interaction, resource distribution, building structures, and rule validation. Together, these elements form a functioning simulation that follows the intended gameplay model.* ◀

7 Reflection on the engineering process

Logan Sedore ▶*The development of The Settlers of Catan in software highlighted the cruciality of planning and design before implementation. The use of a UML diagram helped clarify and organise system responsibilities, reducing unknowns when converting the design into code. A key lesson from the assignment was the value of separating game functionalities and responsibilities into blocks. Assigning specific tasks to components (SRP), such as PlayerHand, PieceHandler, and Validator, prevented classes from becoming overly complex, making the system open to change and extendability. Furthermore, the use of interfaces and generic structures such as Catalog<T> created strong abstraction demonstrating how design decisions made early can simplify implementation later. These abstractions reduced duplication, allowing different parts of the system to interact fluently. The final important takeaway was the importance of state consistency. Many operations make unwanted changes in the system before they are meant to be executed in the actual game. This includes building operations as transactions. Snapshot views for read-only data helped prevent unintended side effects. Overall, the assignment reinforces the role of proper design in software engineering. A clear model, well-defined responsibilities, and modular components made the system easier to implement and provided a strong foundation for future extensions.* ◀

8 Roles and responsibilities

Logan Sedore ▶*All team members contributed equally across design, implementation, and documentation. Individual contributions are summarized below.*◀

- TODO: **John Doe** contributed to the conceptual design and the implementation of RQ1.1.
- TODO: ...