Task 1 - UML Reflection Points

Utility of the Domain Model vs. Specification

The domain model was useful because it made the ambiguous text from the assignment specs into a more structured, visual representation that is quickly comprehensible compared to reading a long page of text. The UML modelling stage of this assignment forced our team to make concrete decisions about cardinality/multiplicity, for example, 1..* vs 0..* and data types that the natural language specification left open to interpretation.

Natural Language vs. Conceptual Modelling

The natural language description in rulebook motivates conceptual modelling because human language is imprecise and prone to contradictions. This UML modelling step was needed to bridge the gap between Catan game rules and software logic.

Some observations I noted:

Visualization of Coupling: The diagram reveals how tightly coupled the Board is to its components (Tile, Intersection), allowing us to comprehend the complexity of the logic before writing a single line of code.

Validation of Responsibilities: It allows us to quickly look at specific scenarios. For example, asking where is the build logic, reveals whether it belongs in Player or Board before we commit to an architecture/structure.

Early Error Detection: Before implementing the logic, the UML model can help iron out the design flaws and potential bottlenecks early to save time and potentially money in the future.

Unmodelled Aspects

Game State + Turn Flow: Class diagram shows what objects are there, but not when they interact or show the sequences of every turn such as roll dice, trade, building, etc.

- We can model this using a Sequence Diagram for the turn logic.

Complex Validity Rules: Currently, the diagram cannot enforce the rules such as a new road must connect to an existing road of the same color.

- This logic is best modelled using an Object Constraint Language, allowing to write specific mathematical rules like invariants that the code must obey, right on the diagram.

OO Mechanisms in Design

Object Oriented mechanisms were critical in organizing the complexity of Catan:

Task 1 - UML Reflection Points

We used an inheritance hierarchy like for example, from superclass Piece extending to subclass Building and extending to subclass Settlement & City to reduce code duplication. We have also shared attributes like owner are defined once in the parent Piece class, rather than repeated in subclasses.

We have also utilized polymorphism heavily. Allows board to treat different objects the same. For example, Intersection hold Building references without needing to know if it's specifically a City or a Settlement until victory points are calculated.

Encapsulation was also utilized in the design. We made all class attributes private and provided public accessor methods instead of exposing the attribute information that can impact the integrity of the game. We protect internal states of the player from being corrupted by other classes like Board.

SOLID Principles

Applied SOLID principles to make design robust:

We changed Player.makeMove() to return a String instead of printing to the console. This ensures the Player class only handles game logic (such as deciding the move), while leaving the responsibility of printing/logging entirely to the Simulator.

Have used the open close principle. Piece and Building inheritance hierarchy allows the system to be extended (for example adding a ship or metropolis in an expansion) by creating new subclasses without modifying the existing core logic in Board or Intersection.

Also the Liskov Substitution Principle was applied. Our design ensures that a City can be used anywhere a Building is expected (like on an Intersection), ensuring that upgrading a piece does not break the board's storage structure.