Monopoly Game Project Report

Object-Oriented Programming (OOP) Course

Submitted by:

ABDULLAH BIN USMAN
Roll Number: 24L-0558

Instructor:

Usman Anwar

Submission Date: May 13, 2025

Abstract

This project implements a digital Monopoly board game using C++ and the SFML library. It supports core mechanics: player movement, property purchasing, rent collection, trading, house building, and chance/community chest cards. The purpose is to demonstrate Object-Oriented Programming (OOP) concepts—inheritance, polymorphism, composition, and aggregation—ensuring a modular and maintainable codebase.

Contents

1 Introduction

1.1 Problem Statement

The goal was to develop a digital Monopoly game replicating core mechanics using OOP principles. Physical board games require manual tracking, which a digital version automates for an engaging experience.

1.2 Objectives

- Implement a Monopoly game with a graphical interface using C++ and SFML.
- Support mechanics: property purchasing, rent payment, trading, house building, chance/community chest cards.
- Apply OOP concepts for scalability and maintainability.
- Provide an intuitive interface with clear visuals and controls.

1.3 Motivations

The project aimed to apply OOP to a complex application, enhancing skills in C++, SFML, and GUI design while creating a multiplayer game.

2 OOP Concepts Used

The game employs OOP principles:

- Inheritance: Type is a base class for Property, Railway, Utility, reusing attributes like typeName.
- Polymorphism: calculateRent is overridden in Property, Railway, Utility for type-specific calculations.
- Composition: Board contains Property, Railway, Utility; GameLogic composes Player, Board, and others.
- Aggregation: Player aggregates Property, Railway via ownership.

3 Class Diagrams

The UML class diagram illustrates the project's core architecture, focusing on 9 key classes and their main relationships (inheritance, composition, aggregation, association) for clarity and ease of understanding. Figure 1 presents the simplified diagram, designed with clear, clean arrows using TikZ. Inheritance uses double arrows, composition uses filled diamonds, aggregation uses hollow diamonds, and association uses solid lines, all in black with thin lines. A detailed textual description follows, covering all 13 classes and relationships.

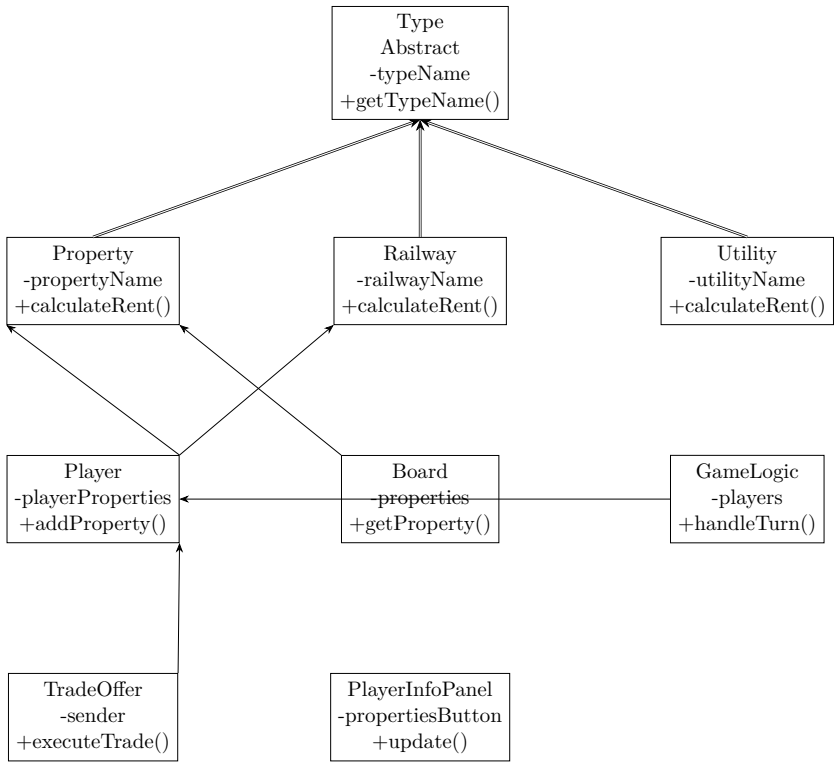

Figure 1: Simplified UML Class Diagram for Monopoly Game

3.1 Detailed UML Description

The UML diagram shows 9 core classes with 8 key relationships for simplicity. All 13 classes, their attributes, methods, and relationships (including those omitted from the diagram) are described below. Relationships include inheritance (double arrow), composition (filled diamond), aggregation (hollow diamond), and association (solid line). Visibility: (+) public, (-) private.

- Type (Abstract)
  - Attributes: -typeName: string, -isBuyable: bool, -typeIndex: int
  - Methods: +Type(string, bool, int), +getTypeName(): string, +getIsBuyable(): bool, +getTypeIndex(): int
- Property (Inherits Type)
  - Attributes: -propertyName: string, -propertyPrice: int, -propertyRent: int[6], -currentRentLevel: int, -housePrice: int, -color: string, -isSold: bool, +indexTable: int, +ColorTable: string*, +propertyCountTable: int*
  - Methods: +Property(), +Property(string, int, int, int[], int, string), +getPropertyName(): string, +getPropertyPrice(): int, +getCurrentRentLevel(): int, +getHousePrice(): int, +getColor(): string, +setCurrentRentLevel(int), +getPropertyRent(): int, +getIsSold(): bool, +setSold(bool), +canBuildHouse(): bool, +calculateRent(): int, -updateTable()

- Railway (Inherits Type)
  - Attributes: -railwayName: string, -railwayPrice: int, -railwayRent: int[4], -isSold: bool
  - Methods: +Railway(), +Railway(string, int, int), +getRailwayName(): string, +getRailwayPrice(): int, +getRailwayRent(int): int, +setSold(bool), +calculateRent(int): int
- Utility (Inherits Type)
  - Attributes: -utilityName: string, -utilityPrice: int, -isSold: bool, -rentMultiplier: int[2]
  - Methods: +Utility(string, int, int), +getUtilityName(): string, +getUtilityPrice(): int, +getIsSold(): bool, +setSold(bool), +calculateRent(int, int): int
- Player
  - Attributes: -playerName: string, -playerMoney: int, -playerPosition: int, -playerProperties: int[22], -playerRailways: int[4], -playerHouses: int[22], -playerHotels: int[22], -RailwayCount: int, -PropertyCount: int, -isBankrupt: bool, -inJail: bool, -jailTurns: int
  - Methods: +Player(), +Player(string), +subtract(Property&), +subtract(Railway&), +add(Property&), +add(Railway&), +getPlayerMoney(): int, +Bankrupt(), +isPlayerBankrupt(): bool, +Buy(T&, Player&), +addMoney(int), +subtractMoney(int), +setPlayerPosition(int), +getPlayerPosition(): int, +transferMoney(Player&, int), +transferTax(int), +setInJail(bool), +getInJail(): bool, +getTaxMoney(), +buyHouse(Property&, Board&), +showHouseBuyDialog(Property&, Board&), +ownsProperty(int), +ownsAllPropertiesOfColor(string, Board&), +getPlayerName(): string, +drawPlayer(sf::RenderWindow&, sf::Font&, int)
- Board
  - Attributes: -boardTexture: sf::Texture, -boardSprite: sf::Sprite, -properties: Property*, -railways: Railway*, -utilities: Utility*
  - Methods: +Board(), +draw(sf::RenderWindow&), +getPropertyAtPosition(int): Property&, +Board()
- Button
  - Attributes: -button: sf::RectangleShape, -buttonText: sf::Text, -font: sf::Font
  - Methods: +Button(float, float, float, float, string, sf::Color, sf::Color), +draw(sf::RenderWin +isMouseOver(sf::RenderWindow&), +setFillColor(sf::Color), +setText(string)
- Dice

- Attributes: -lastRoll: int, -isDoubles: bool
- Methods: +Dice(), +roll(): int, +getLastRoll(): int, +isDoublesRolled(): bool

- Chance
  - Attributes: -chanceCards: string[16]
  - Methods: +drawCard(): string

- CommunityChest
  - Attributes: -communityChestCards: string[16]
  - Methods: +drawCard(): string

- TradeOffer
  - Attributes: -sender: Player*, -receiver: Player*, -senderProperties: vector<Property*>, -receiverProperties: vector<Property*>, -senderMoney: int, -receiverMoney: int
  - Methods: +TradeOffer(Player*, Player*), +addSenderProperty(Property*), +addReceiverProperty(Property*), +setSenderMoney(int), +setReceiverMoney(int), +executeTrade(): bool, +showTradeDialog(Board&)

- GameLogic
  - Attributes: -players: Player*, -numPlayers: int, -board: Board, -dice: Dice, -currentPlayerIndex: int, -rollDiceButton: Button*, -endTurnButton: Button*, -diceRollText: sf::Text, -font: sf::Font, -hasRolledThisTurn: bool, -doubleRollCount: int
  - Methods: +GameLogic(int, string[]), +handlePropertyLanding(Player&, int), +movePlayer(), +drawGameControls(sf::RenderWindow&), +handleGameControls(sf::RenderWindow&, sf::Event&), +nextTurn(), +getCurrentPlayer(): Player&, +getPlayerByIndex(int): Player&, +getPropertyByIndex(int): Property&, +handleSpecialSquare(Player&, int), +processCommunityChestCard(Player&, string), +processChanceCard(Player&, string), +showTradeWindow(int), +GameLogic()

- PlayerInfoPanel
  - Attributes: -panel: sf::RectangleShape, -font: sf::Font, -playerTexts: sf::Text[4], -propertiesButton: Button*, -tradeOfferButton: Button*
  - Methods: +PlayerInfoPanel(GameLogic&), +update(GameLogic&), +draw(sf::RenderWind +getTradeOfferButton(): Button*, +getPropertiesButton(): Button*, +PlayerInfoPanel()

- PropertiesWindow
  - Attributes: -propertiesWindow: sf::RenderWindow, -font: sf::Font, -propertyTexts: sf::Text[22], -titleText: sf::Text, -propertyCount: int
  - Methods: +PropertiesWindow(GameLogic&), +update(GameLogic&, int), +show(), +isOpen(): bool, +getWindow(): sf::RenderWindow&, -setupText(sf::Text&)
- Relationships
  - Property, Railway, Utility inherit from Type (shown in diagram).
  - Board composes Property, Railway, Utility (Property shown in diagram).
  - GameLogic composes Player, Board, Dice, Button (Player shown in diagram).
  - PlayerInfoPanel composes Button, sf::Text (not shown in diagram).
  - PropertiesWindow composes sf::Text (not shown in diagram).
  - Player aggregates Property, Railway (shown in diagram).
  - TradeOffer associates with Player, Property (Player shown in diagram).

4 Test Cases

Test cases verify core functionalities:

1. Player Movement and Passing GO
   - Input: Player rolls 5 from position 38.
   - Expected: Moves to position 3, gains $200.
   - Observed: Marker moves to position 3, money updated.
2. Property Purchase
   - Input: Lands on “Lab 4” (price $60), clicks “Buy”.
   - Expected: Money decreases by $60, property added.
   - Observed: Money deducted, property shown.
3. Rent Payment
   - Input: Player 1 lands on Player 2’s “Lab 4” (rent $4).
   - Expected: Player 1 pays $4, Player 2 gains $4.
   - Observed: Money transferred, rent shown.

4. Chance Card
    - Input: Lands on Chance, draws “Advance to Go”.
    - Expected: Moves to position 0, gains $200.
    - Observed: Marker moves to GO, money updated.
5. Community Chest Card
    - Input: Lands on Community Chest, draws “Pay hospital $100”.
    - Expected: Money decreases by $100.
    - Observed: Money deducted, card shown.
6. Trade Execution
    - Input: Player 1 offers “Lab 4” and $100 for “Old Audi”, both confirm.
    - Expected: Properties swapped, money transferred.
    - Observed: Trade completed, properties updated.
7. House Building
    - Input: Owns Light Blue properties, buys house on “Mico Lab” (price $50).
    - Expected: Money decreases by $50, rent level increases.
    - Observed: Money deducted, house added.
8. Double Roll Handling
    - Input: Rolls doubles twice, then a third double.
    - Expected: Extra turns for two doubles, jail on third.
    - Observed: Extra turns, then jailed.

5 Future Directions

Enhancements based on the codebase:

- Optimize PropertiesWindow to show only owned properties.
- Support Railway, Utility trading in TradeOffer.
- Add auctions for declined purchases.
- Implement “Get out of Jail Free” card.

- Enhance UI to show houses/hotels on board.
- Enable saving/loading game states.