Milestone 4

SYSC 3110A: Software Development Project

Team 21:
Marc Aoun, 101263718
Iman Elabd, 101232751
Amreen Shahid, 101306199

Submitted on: December 5, 2025

1. UML Diagram

2. Human Player Sequence Diagram

3. Explanation of Data Structures and Design Used

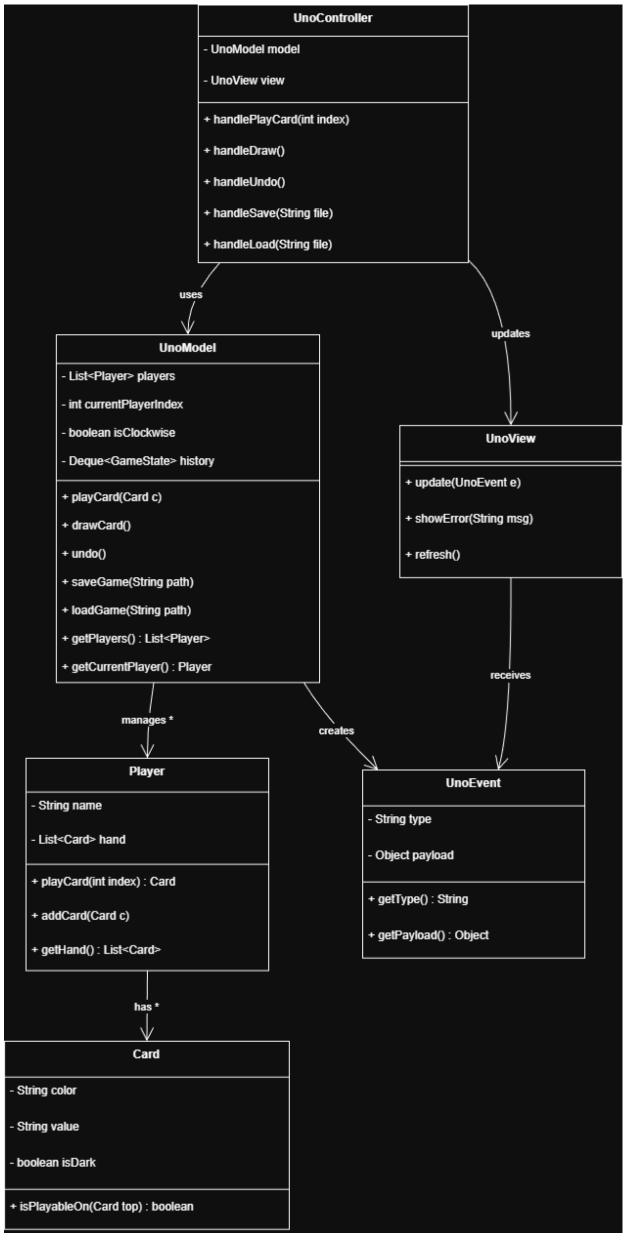
From Milestone 1 to 3, the project changed a lot. firstly, the old GameFlow class was doing everything in one place, like handling turns, checking rules, printing messages, and taking user input. Because it controlled so much on its own, the code became messy and difficult to build on. When we redesigned the project, we removed GameFlow completely and moved the whole game logic into UnoModel. UnoModel now looks after the turn order, which cards can be played, the top card, scoring, and all the special card effects. We also cleaned up the data by using enums for colours and values, and we updated the Player class so each player manages their own cards properly. Switching to MVC also changed our UML diagrams because the model, view, and controller are now clearly separated. UnoView became the interface for the GUI, and UnoController handles the button actions instead of Scanner input. The model updates the GUI right away through its observer methods, which makes the game feel smoother. During this stage, we also added the AI player, which follows simple rules: it plays any legal card it has, draws when it needs to, and usually uses special cards. After these changes, the project became much cleaner, manageable, and better suited for the GUI compared to the original text-based version.For Milestone 4, the project became even more complete because we added replay, undo/redo, and saving and loading the game. Before this milestone, the game could only play one round and everything reset after closing the program. Now players can start new rounds after someone wins, and they can also begin a new game when a player reaches 500 points. We also added full undo and redo with multiple levels, so the player can go back or forward through the game state, and the GUI updates right away. Another important update was serialization and deserialization, which lets the user save the whole game and load it later. Only the model gets saved, and the GUI rebuilds itself when loading. We also added error-handling so the game stays stable if the file is missing or corrupted. Because of these new features, the UML had to be changed again, and new sequence diagrams were made for undoing a move and loading a saved game. We also added JUnit testing for undo/redo and saving/loading to make sure everything works

4. Breakdown of Project Tasks and Responsibilities of Each Member

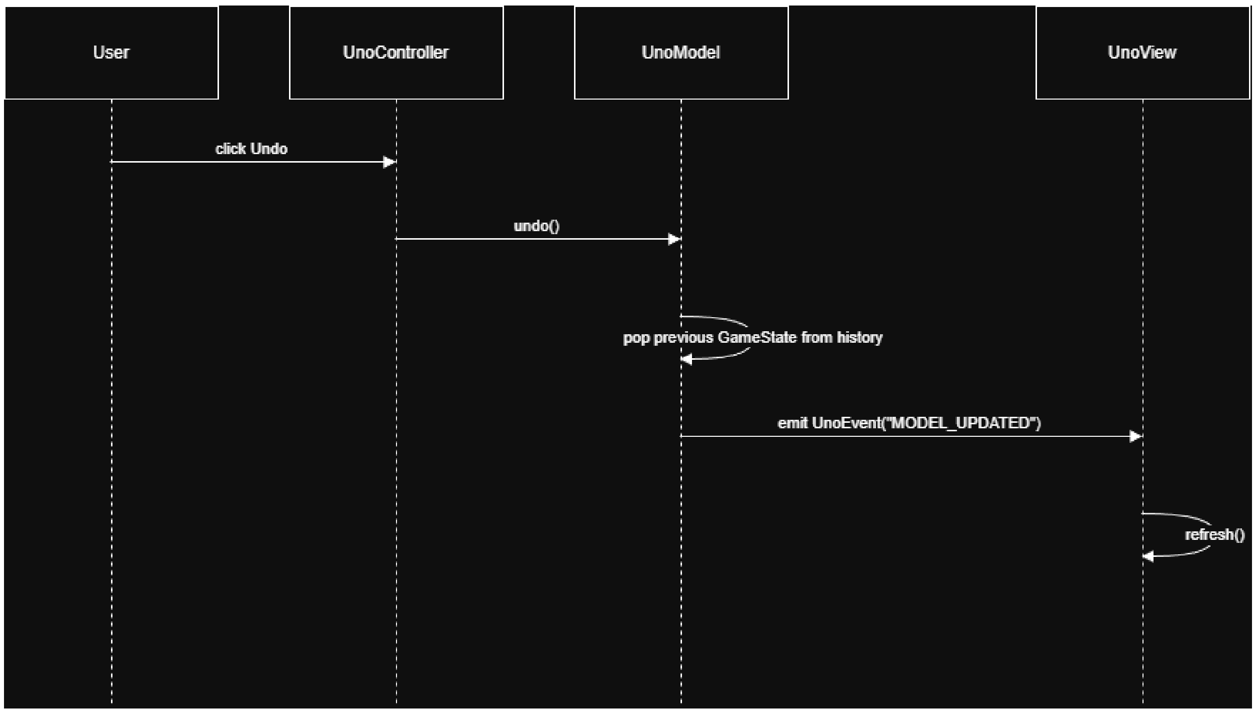
Name:	Tasks and Responsibilities:
Marc Aoun	Section 3 – Serialization / Deserialization Functionality 1. Serialization Functionality: The user is able to save a game 2. Deserialization Functionality : After saving, closing and reopening the GUI, the user is able to reload a saved game 3. Implementation : Only the required model classes are serialized 4. Error-Handling : Evidence of error-handling is present in the code for serialization and deserialization Section 5 – UML Modeling and Data Structures 1. UML Modeling: Provide well-structured class diagrams with complete variable and method signatures. Test classes are not required in the UML modeling. 2. Sequence Diagrams: Include sequence diagrams for important game scenarios: a. Undoing a move b. Deserializing / loading a saved game 3. Data Structure Explanation: a.UML Class Diagrams: Document the changes you made to your UML and data structures from Milestone 3 and explain why.
Iman Elabd	JUnit Test Coverage : Evidence of JUnit testing for this milestone. a. Evidence of JUnit testing for undo/redo functionality b. Evidence of JUnit testing for serialization/deserialization functionality Section 6 – Documentation 4. User Manual : Supply a user manual that explains to users how to run the jar file, how to play the game, and how to use the features implemented in all milestones. 5. Readme File: Supply a well-written readme file that explains the rest of the deliverables, lists the contributions of each team member for current and past milestones, and explains any known issues. 6. JavaDocs and Code Comments : JavaDocs are present and well-written for classes and methods, and code comments have been added where necessary. 7. Proper Use of GitHub: Individuals use clear and descriptive commit

	messages and branch names during the development process.
Amreen Shahid	Section 1 – Replay Functionality 1. Rounds: Players can play additional rounds after a Player has an Uno 2. Games: Players can play additional games after a Player reaches 500 points 3. Replay Implementation : The functionality for playing multiple rounds/games is implemented with minimal faults and with appropriate visual indications to the users. Section 2 – Undo / Redo Functionality 1. GUI Implementation : Undo and Redo Options are available to the user and nicely presented on the GUI 2. Undo Functionality : Undo functionality can successfully undo a move 3. Redo Functionality : Redo functionality can successfully redo a move 4. Multi-level Undo/Redo : Users can undo and redo multiple times in a row while maintaining proper game state 5. Performance Considerations : Undo and redo functionalities are implemented without significant impacts to game performance

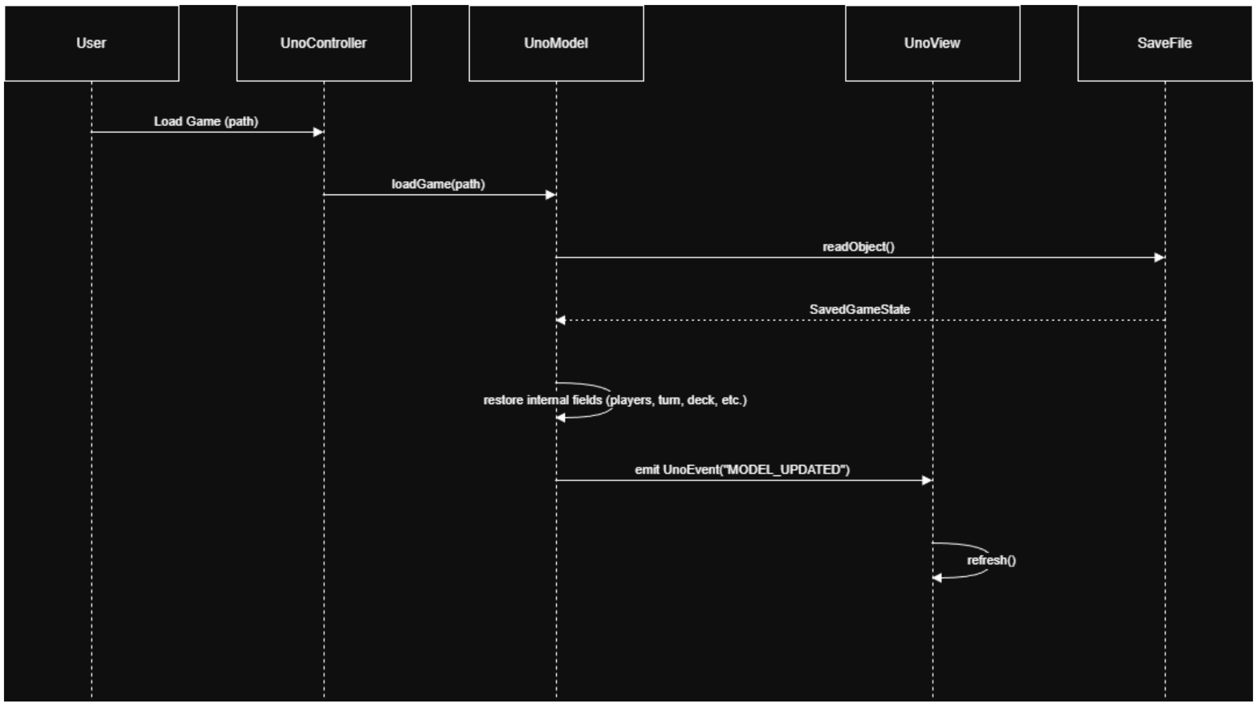
UML


UnoController
- UnoModel model
- UnoView view
+ handlePlayCard(int index)
+ handleDraw()
+ handleUndo()
+ handleSave(String file)
+ handleLoad(String file)
uses
updates
UnoModel
- List<Player> players
- int currentPlayerIndex
- boolean isClockwise
- Deque<GameState> history
+ playCard(Card c)
+ drawCard()
+ undo()
+ saveGame(String path)
+ loadGame(String path)
+ getPlayers() : List<Player>
+ getCurrentPlayer() : Player
UnoView
+ update(UnoEvent e)
+ showError(String msg)
+ refresh()
receives
manages *
creates
Player
- String name
- List<Card> hand
+ playCard(int index) : Card
+ addCard(Card c)
+ getHand() : List<Card>
UnoEvent
- String type
- Object payload
+ getType() : String
+ getPayload() : Object
has *
Card
- String color
- String value
- boolean isDark
+ isPlayableOn(Card top) : boolean

Sequence Diagram: Undoing a move


User
UnoController
UnoModel
UnoView
click Undo
undo()
pop previous GameState from history
emit UnoEvent("MODEL_UPDATED")
refresh()

Sequence Diagram: Deserializing / loading a saved game


User
UnoController
UnoModel
UnoView
SaveFile
Load Game (path)
loadGame(path)
readObject()
SavedGameState
restore internal fields (players, turn, deck, etc.)
emit UnoEvent("MODEL_UPDATED")
refresh()