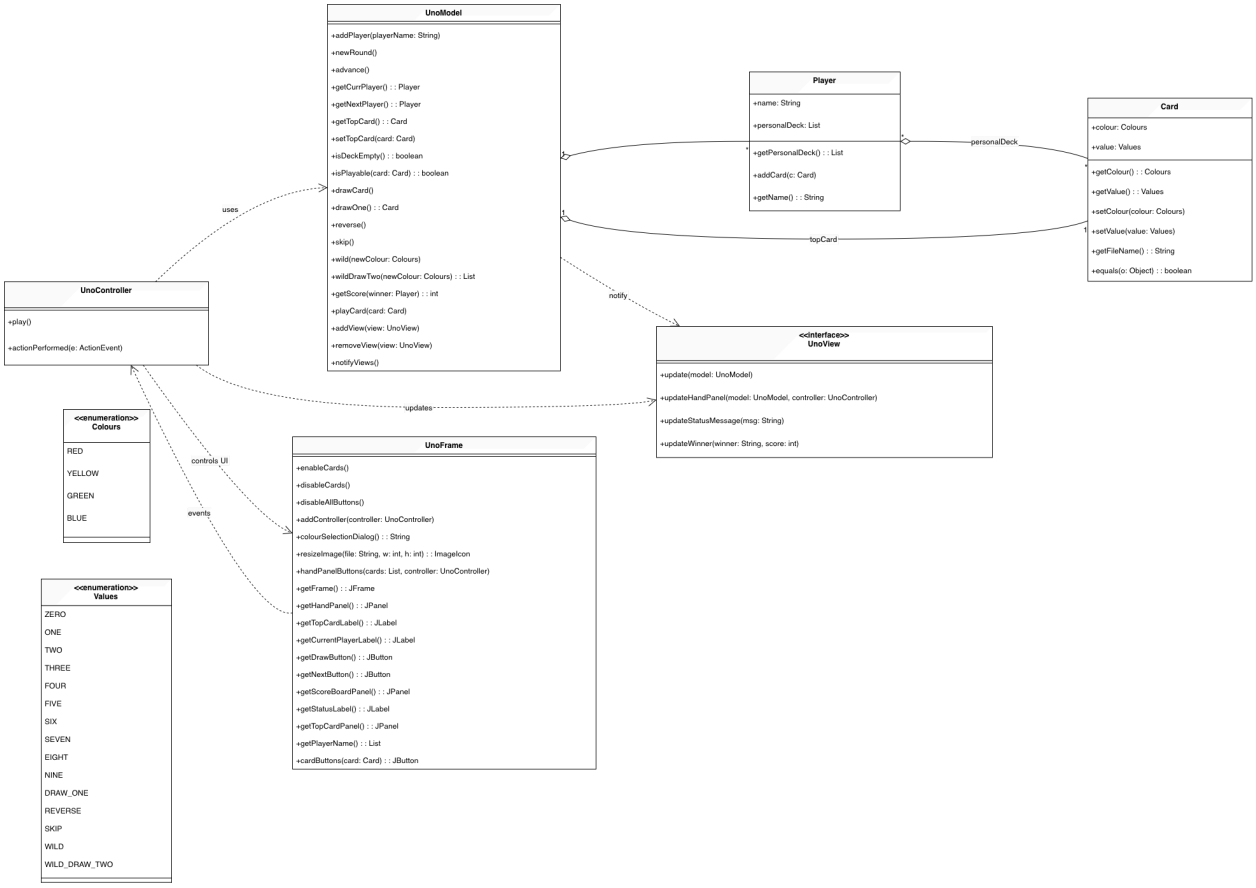
SYSC 3110 - A
Software Development Project

Milestone 2

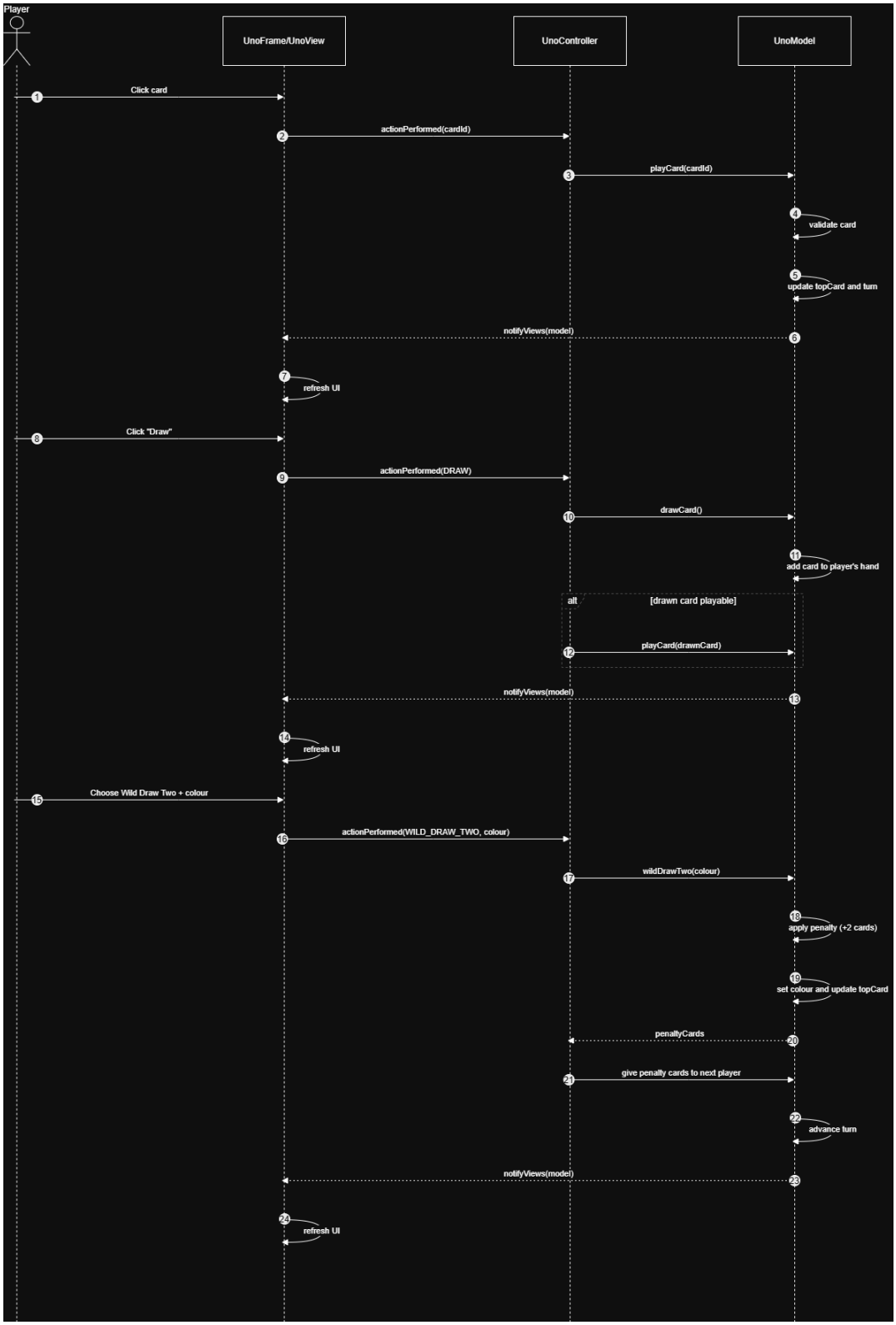
Team 21:
Marc Aoun, 101263718
Iman Elabd, 101232751
Naima Mamun, 101276213
Amreen Shahid, 101306199

Submitted on: November 10, 2025

1. UML Class Diagram



2. Sequence Diagrams



3. Explanation of Data Structures and Design Used

From Milestone 1 to Milestone 2, the data structures and UML were changed to support a full MVC architecture and a graphical user interface instead of the old console-based, procedural GameFlow system. The original GameFlow class mixed input, output, and rules in one place, so it was replaced by a dedicated UnoModel class that stores game state, enforces rules, manages players, tracks the current player, updates the top card, handles personal decks, manages scores, and notifies views. Scanner-based input and while-loops were removed and replaced with an event-driven structure using UnoController, which listens for button presses and forwards actions to the model. We ensured that our MVC View was an interface that UnoFrame implements. This way we maintain a clear separation between the model, view, and controller components of our MVC architecture. This design allows the controller to interact with the view through a clear contract without being tightly linked to the specific GUl implementation. The interface we created ensures that any class implementing it must provide the necessary update methods, enabling flexibility while preserving the MVC pattern's communication structure.The Card data structure was also upgraded by turning colours and values into strongly typed enums so the GUI can reliably map them to images and prevent invalid card states. Player was expanded to fully manage its own List<Card> personalDeck, allowing cleaner deck operations needed for the GUI. The old ActionCards helper class was removed and its logic (skip, reverse, wild, draw two) was merged into UnoModel to simplify rule handling. The model also gained an observer pattern using addView and notifyViews so the GUI updates automatically whenever the state changes. Overall, the system changed from a linear text-based loop to a modular, event-driven, GUI-compatible design with cleaner class responsibilities and safer data structures.

4. Breakdown of Project Tasks and Responsibilities of Each Member

Name:	Tasks and Responsibilities:
Marc Aoun	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. 3. Data Structure Explanation: Document the changes you made to your UML and data structures from Milestone 1 and explain why. 4. Readme File: Supply a well-written readme file which explains the rest of

	the deliverables, lists the contributions of each team member for current and past milestones, and explains any known issues. 5. JavaDocs and Code Comments: JavaDocs are present and well-written for classes and methods, and code comments have been added where necessary.
Iman Elabd	General Code Quality 1. Data Consistency and Synchronization: The information displayed on the view is accurately updated based on the current state of the game in the model 2. Error Handling: Appropriate recovery from invalid user actions 3. Junit Test Coverage: Thorough testing of all model-related classes and methods, and game logic.
Naima Mamun	1. JFrame: The graphical display is implemented within a JFrame 2.Player Range: The ability to select 2–4 players 3. Visibility of Game State: Display the cards held by the player and the top card 4.Visibility of Player Input: Options to play a card, draw a card, progress to the next player, and select a colour for a Wild Card are shown when applicable 5. User friendly: The user interface is organized and easy to understand 6. Responsive: The user interface responds to user interaction via mouse clicks
Amreen Shahid	1. MVC Pattern: Correct implementation of the MVC pattern as stated in the lecture slides 2. Java Event Model: Correct implementation of the Java Event Model for major updates to the game state 3. Separation of Concerns: A clear separation exists between the responsibilities of the Model, View and Controller components 4. Component Communication: Proper communication between Model, View and Controller