1. Introduction and Scope

Project Overview
This project is about creating a Mahjong solitaire game where players match pairs of tiles until no moves remain. The user interface (UI) will be built in QML for a responsive and interactive design, while the game logic will be handled with C++ for efficiency and flexibility. The finished product will be a desktop game that includes the classic "Turtle" Mahjong layout, lets players select and remove matching tiles, and may have extra features like shuffling and sound effects.

Functional Requirements

Must-Have Features:

- Show the Mahjong "Turtle" layout with tiles stacked in layers.
- Let players select and remove matching tile pairs if the tiles are "open."
- Block tile removal for tiles that are covered or surrounded.
- Provide visual feedback by highlighting selected tiles, dimming blocked tiles, and fully showing open tiles.

Should-Have Features:

- A shuffle button to rearrange tiles when players are stuck.
- Sounds for selecting tiles, removing pairs, and making invalid moves.

Could-Have Features:

- Animations for selecting or removing tiles.
- Basic 3D effects for enhanced visuals using QtQuick3D (if time permits).

Won't-Have Features:

- Guaranteed solvable layouts.
- Multiplayer or online play.
- AI or hint systems.

Audience:

- Casual Mahjong players who enjoy solitaire puzzles.
- Students and developers looking for examples of integrating QML and C++ with Qt.

How It Works:
The game opens with a Mahjong board displayed on the screen. Players pick two tiles. If the tiles match, they disappear, and a sound plays. If the tiles don't match, an error sound is played, and

the selection resets. Players continue removing pairs until no moves remain. The shuffle button can be used to mix up the tiles and create new possibilities.

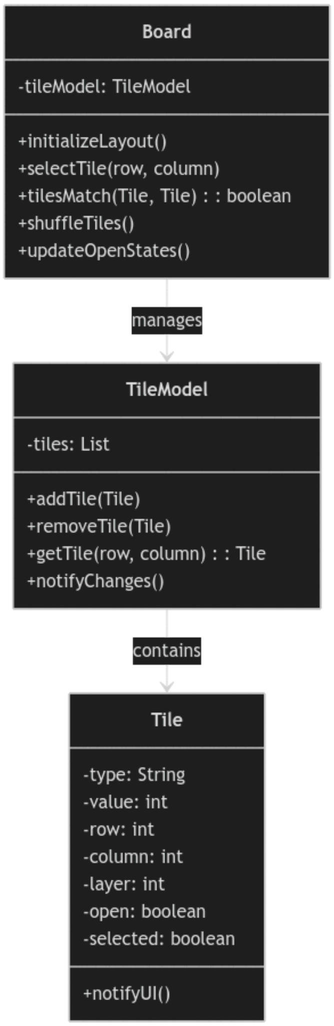
2. High-Level Software Structure

System Overview

The software is divided into two main parts:

1. User Interface (QML Layer):
   - Manages the layout, animations, and tile rendering.
   - Reacts to updates from the C++ backend.
   - Handles player inputs like mouse clicks and communicates with the backend.
2. C++ Logic Layer:
   - Tile Class: Represents each Mahjong tile with properties like type, position, and whether it's open or selected.
   - TileModel (QAbstractListModel): Manages a dynamic list of Tile objects. This model connects the backend logic to the UI by providing data to display and handling updates when tiles are removed or changed.
   - Board Class: Creates the Mahjong "Turtle" layout, calculates which tiles are open, manages tile selection and removal, and handles shuffling.

Class Diagram (Overview):



QML Interaction:
The QML layer binds directly to the TileModel and Board objects. Player interactions, like selecting tiles, are passed to the backend for validation and logic processing.

3. Use of External Libraries

Qt Modules Used:

- QtQuick/QML: For creating the UI, tile rendering, and managing interactions.
- QtMultimedia: For adding sound effects.
- QtCore/QtGui: For handling core features like event loops, data management, and property updates.

Why These Libraries?
These are part of Qt's core framework, which is reliable and well-documented. They allow us to cleanly separate UI design from backend logic.

External Dependencies:
No third-party libraries are needed. Tile images and sound effects will come from free resources (e.g., opengameart.org) and will be bundled into the app as part of its resource files.

4. Sprint Plan and Project Management

Development Approach:
We will use a simplified Agile process with four one-week sprints. Each sprint begins on Monday and ends on Friday with a review. Daily stand-up meetings will help us track progress and address any issues quickly.

Sprint Goals:

- Sprint 1 (Week 1):
  - Set up the project structure with directories like src, tests, and plan.
  - Create basic Tile and TileModel classes.
  - Display a grid of placeholder tiles using QML.
  - Deliverable: A basic app that shows placeholder tiles with no game logic.
- Sprint 2 (Week 2):
  - Add the Board class and generate the "Turtle" layout.
  - Implement logic to check which tiles are open or blocked.
  - Replace placeholders with real Mahjong tile images.
  - Deliverable: A static Mahjong board with visible open/blocked states.
- Sprint 3 (Week 3):
  - Add functionality to select, match, and remove tiles.
  - Implement sound effects for player actions.
  - Introduce the shuffle feature.
  - Begin testing the logic using unit tests.
  - Deliverable: A fully playable game with basic sound effects and shuffling.
- Sprint 4 (Week 4):
  - Polish the UI with better visuals and animations.
  - Perform extensive testing, including memory checks.
  - Finalize documentation, including the user manual and code comments.
  - Deliverable: A polished, tested game ready for release.

Meeting Schedule:

- Daily Stand-Ups: Every weekday at 9:00 AM for 15 minutes.
- Weekly Reviews: Fridays at 3:00 PM for 30 minutes.

Progress Tracking:

- Version Control: All code will be stored in a Git repository, using feature branches for new tasks.
- Task Management: Trello or GitHub Issues will track progress.
- Code Reviews: Every team member’s work will be reviewed by another before merging.

Team Roles:

- Member 1: Focus on backend logic, including Board and tile matching (40%).
- Member 2: Focus on QML integration and UI design (30%).
- Member 3: Focus on testing and documentation (30%).

5. Challenges and Adjustments

Possible Issues:

- The "Turtle" layout might be difficult to implement, so we may start with a simpler one.
- Animations and 3D effects are optional and will only be added if time permits.
- Sound effects and the shuffle feature are secondary priorities.

Mitigation Strategies:

- Focus on completing core gameplay first.
- Keep the code modular to allow easier changes.
- Use daily meetings to address blockers and adjust tasks as needed.

6. Documentation and Diagrams

Planned Documentation:

- Class Diagrams: To explain how Tile, TileModel, and Board work together.
- Sequence Diagrams: To show workflows like tile selection and removal.
- Architecture Diagrams: To highlight the separation of UI and backend logic.

Additional Documents:

- Doxygen-generated code documentation.
- A user manual with instructions for playing the game.

7. Conclusion

This plan outlines how we will create a Mahjong solitaire game, focusing on its features, structure, and development process. The Agile approach ensures flexibility, while clearly defined goals and roles keep the project on track. By the end, we aim to deliver a fully functional and polished game, along with comprehensive documentation.