Project Plan for Turtle Graphics Application

1. Scope of the Work

The program

The program consists of a 2D canvas and the Turtle, which can also be called the cursor. The Turtle is placed in the 2D-plane of the canvas and can be moved therein with a discrete set of commands. As the turtle moves, its path is marked on the canvas. This enables the programmatic drawing of ordered planar patterns among other things. The canvas will be created in its own window.

The application will be used by inputting commands through a command line interface (CLI) in the terminal where the program executable is run. Sequences of commands can be run by defining and executing a script file. Scripts may also include loops and simple functions to enable more advanced use. Users will see the turtle respond to the commands without delay by the animation of its movement and the drawing of its path on the canvas.

Turtle states can be saved to a file in a script format that can be later executed by the program. Since the program is simple and commands are fast to execute, a state can be recovered by simply executing all the commands that lead to the state. This also enables a user to easily understand and modify the saved states.

Features of the basic app

- Canvas
    - turtle and its path displayed on a 2D plane
    - current turtle location and rotation representation
    - dynamic resizing of the canvas window
- Command Line Interface: simple UI to input commands
- Basic turtle commands
    - move forward
    - turn
    - pen down/up
    - set pen color and width

Advanced Features that are added on top of the basic app

- Additional turtle commands:
    - goto (move the Turtle to the specified coordinates via the shortest path)
    - setdir (set the bearing of the Turtle relative to north)
    - setspeed (set the speed of Turtle movement)
    - show / hide turtle
- Command scripts:

    - scripts that can be read from a file and executed
    - loops, numerical variables and function definitions
- Collisions:
    - obstacles on a 2D-plane that interfere with Turtle movement
    - canvas borders
    - collision rules
        - reflect (set canvas edges to reflect the cursor)
        - stop (set canvas edges to stop the cursor)
        - modulo (set canvas edges to teleport the cursor to the other side)
- Save and Load Functionality:
    - the current drawing on the canvas can be exported to an image file
    - the whole state of the program (including all executed commands) can be saved to a file and loaded later

Tentative Features

- Command rewinding:
    - undo past commands
    - redo undone commands
- More turtle commands
    - teleport (same as goto but the path is not drawn)
    - undo, redo (past commands)
    - seticon (set a custom icon that represents the Turtle)
    - tilt (rotate the icon of the Turtle relative to cursor)
- Sound Effects: Integrate QtMultimedia to add sound effects to turtle movements.
- Customizations
    - cursor icon
    - background image
- Multithreaded mode: draw with multiple Turtles simultaneously to enable interesting animations
- 3D extension: Use QtQuick3D to create a 3D canvas

2. High-Level Structure of the Software

Diagram 1: High-level diagram of the application (see the appendix for a zoomed version)

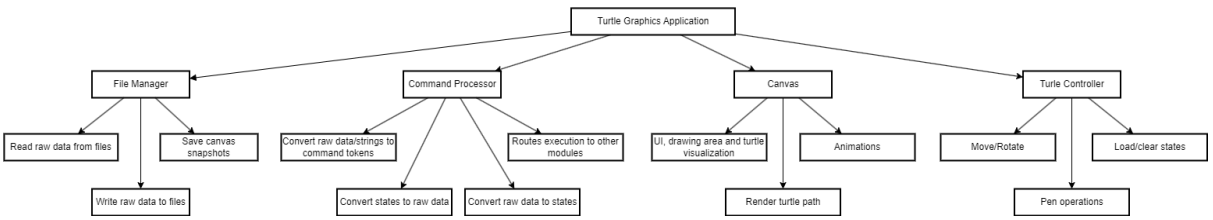
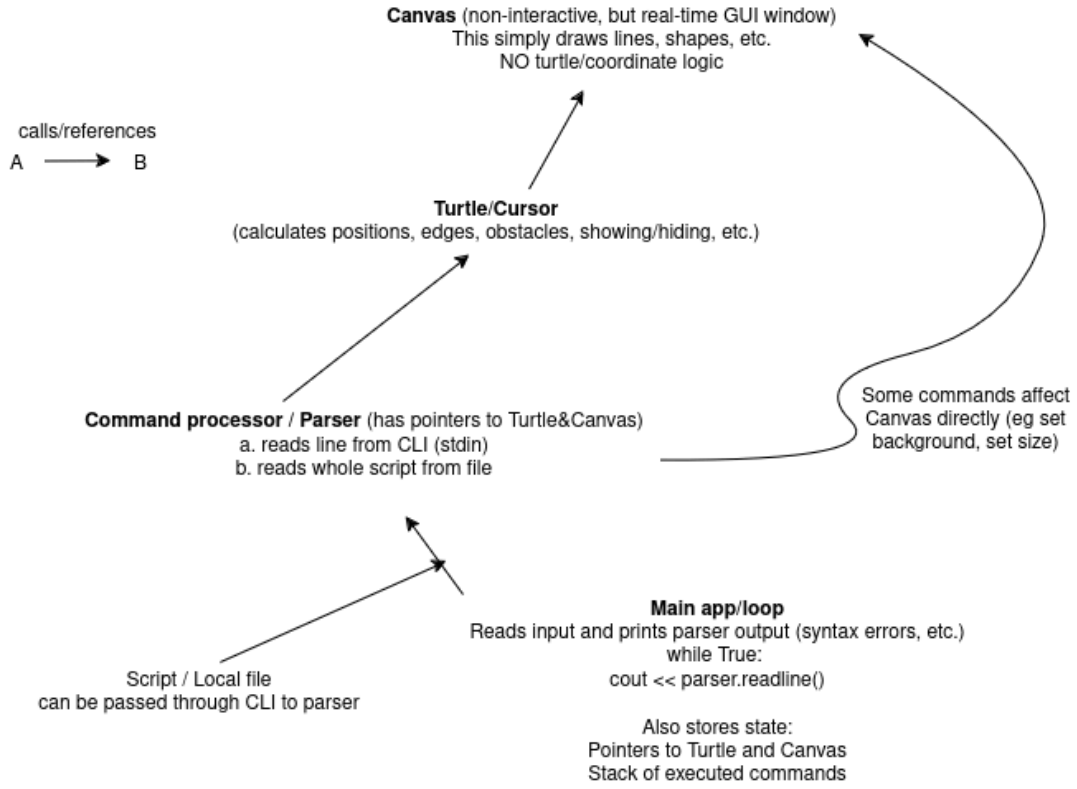


Diagram 2: Flow chart showing the high-level architecture of the program. The main modules are bolded and the arrows show how they connect together.



Main Modules

- **File Management**: Handles reading from files, saving and loading of turtle states and saving of canvas snapshots.
- **Command Processor**: Processes input and routes execution to corresponding modules.
- **Turtle Control**: Handles turtle logics, states and properties (position, angle, pen state).
- **Canvas**: Implemented with Qt Quick and QML. Manages the drawing area and rendering of the turtle's path. Can include animations and CLI if we decide to make it a part of the UI (instead of the default CLI).

Main Classes

- **Application/Main**: Launches and shuts down the program. Reads input from CLI.
- **Parser**: Parses data into command tokens or scripts. Handles syntax errors.
- **Router**: Routes execution to corresponding modules.
- **Turtle**: Represents the turtle with properties like position and angle. Includes methods to update turtle position and rotation, manage pen properties and handle turtle states.
- **Canvas**: Manages drawing operations and displays the turtle's movement. Has properties like size and background.
- **FileManager**: Handles file operations for saving and loading.

3. Planned Use of External Libraries

- Qt Core and Qt GUI: For basic application structure and GUI elements.
- QtMultimedia: For sound effects.
- QtQuick and QtQuick3D: For enhanced UI and potential 3D features.

4. Sprints

Duration: Each sprint will last 2 weeks.

Sprint Meetings: Weekly meetings every Monday at 12 PM for 15-30 minutes to discuss progress, challenges, and next steps. Potential additional meetings during critical phases.

Sprint 1

Start Date: 21.10.2024; End Date: 01.11.2024

Goals:

- Make a project plan
- Prepare the GitLab repository
- Assign tasks

Sprint 2

Start Date: 04.11.2024; End Date: 15.11.2024

Goals:

- Basic command execution via CLI
- Turtle forward movement and rotation
- Initial canvas and turtle visualization
- Implementation of the features described above for the “Basic App”
- Setting up automated tests

Sprint 3

Start Date: 18.11.2024; End Date: 30.01.2024

Goals:

- Extending the application to handle scripts
- Simplifying code from the prior sprint where possible
- Saving/Loading of turtle states and canvas snapshots

- Implementing obstacles and other features described in the “Advanced Features” section

Sprint 4

Start Date: 04.12.2024; End Date: 15.12..2024

Goals:

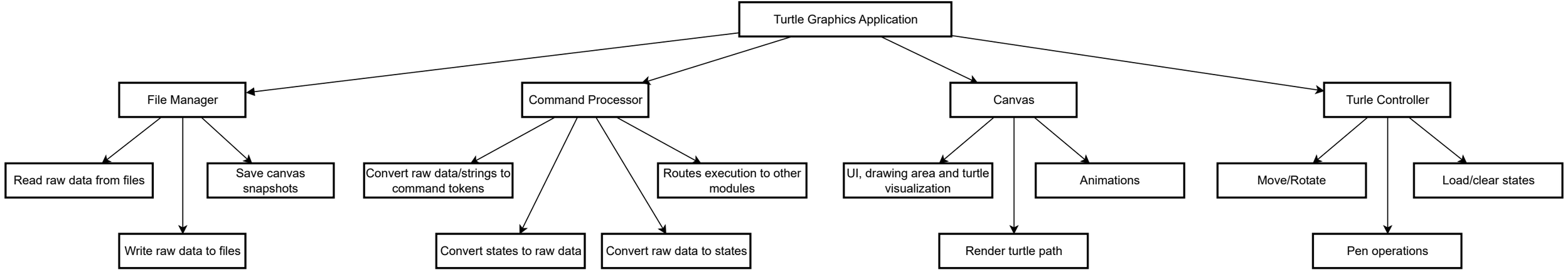
- Possible tentative features
- Polishing
- Testing
- Documentation

Progress Tracking

Progress will be tracked using the GitLab issue board.

Each task will be broken down into smaller tasks and assigned to team members.

Weekly meetings will serve to review tasks, update progress, and adjust plans as necessary.


Turtle Graphics Application
File Manager
Command Processor
Canvas
Turle Controller
Read raw data from files
Save canvas snapshots
Write raw data to files
Convert raw data/strings to command tokens
Routes execution to other modules
Convert states to raw data
Convert raw data to states
UI, drawing area and turtle visualization
Animations
Render turtle path
Move/Rotate
Load/clear states
Pen operations