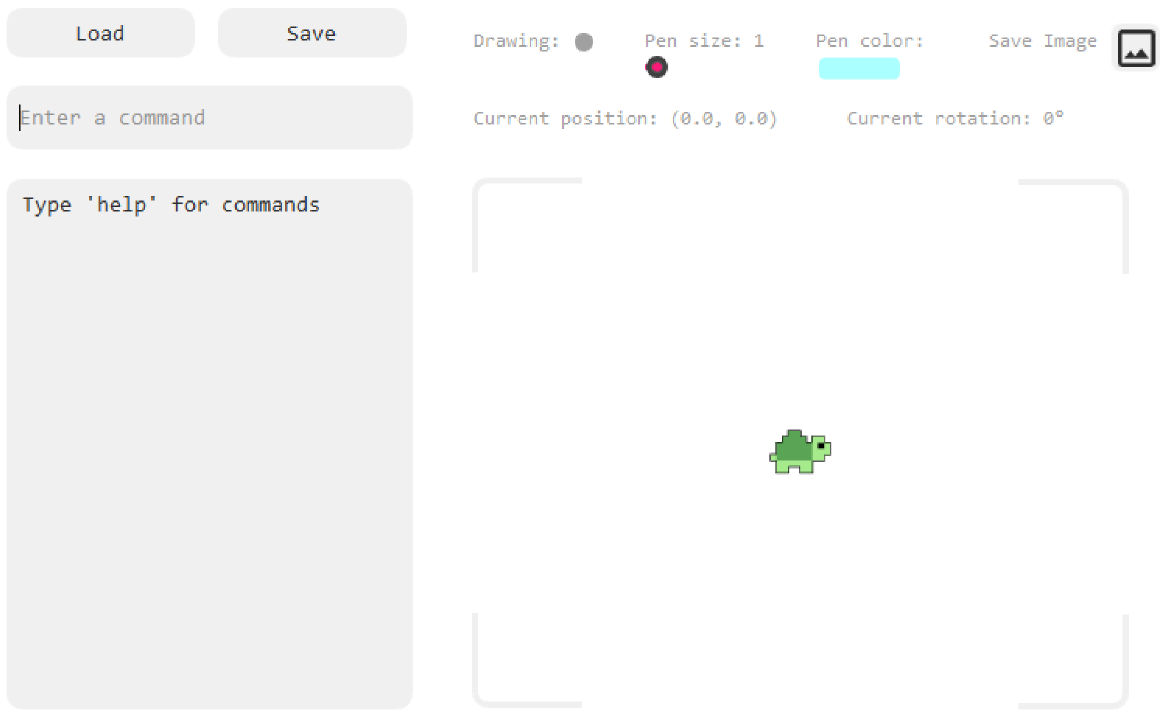
9.12.2024

Turtle Graphics

Project documentation

Mathias Castrén mathias.castren@aalto.fi
Hugo Tamm hugo.tamm@aalto.fi
Xiwei Zhao xiwei.zhao@aalto.fi
Giang Le giang.1.le@aalto.fi

[**Turtle Graphics**]



Type 'help' for commands

Overview

The software is a turtle graphics application that allows users to visually control and direct a virtual "turtle" using text-based commands. Users can input commands to move the turtle, draw geometric shapes, and interactively create graphics. The application features a graphical interface that visually represents the turtle's actions based on the commands entered, along with a command history and support features for user convenience.

What the software does:

The software provides an interactive graphical environment using the Qt library where users can control a turtle using a variety of commands. The turtle's position and angle are accurately displayed on a 2D field, allowing users to visualize its movements and rotations in real-time. Through a simple command-line interface, users can enter instructions to manipulate the turtle, such as moving it forward, turning it by a specified angle, or toggling between pen-up and pen-down modes, which control whether the turtle draws while moving.

Additionally, the software offers the ability to adjust pen attributes like color and line width, providing greater flexibility in creating drawings. The turtle can execute several predefined drawing commands, including shapes like stars, triangles, squares, rectangles, circles, hexagons, and even more complex figures like a house or random patterns. Users can also make the turtle spin by specifying the number of sides for the shape.

The program includes features that allow users to save their work, including the turtle's current state and the drawn image, and later load them for continued work. Furthermore, users can execute command scripts stored in files, enabling batch processing of instructions for automated or repetitive tasks.

In addition, the application plays a sound effect each time the turtle begins a movement, enhancing the interactivity of the experience. This sound is triggered when the turtle starts moving, providing audible feedback to the user for actions such as moving forward or turning, making the experience more engaging and dynamic.

What the software does not do:

The software is limited to predefined turtle movements and cannot interpret complex programming logic or scripts. It also does not include real-time collaborative editing or advanced graphics processing (for instance, 3D graphics). Enhancing the drawing field with obstacles that block turtle movement is not implemented.

Software structure

Overall Architecture

The program is developed using the Qt framework, which simplifies the creation of GUI components. Its modular design comprises the following key components:

- **`main.cpp`**: The entry point of the program. This file initializes the application and sets up the main window.
- **`mainwindow.cpp/h`**: This handles the graphical user interface (GUI). It includes the input field for commands, the command history display, and the main canvas where the turtle's movements are rendered.
- **`turtle.cpp/h`**: This core component processes turtle commands, updates its position and state (for instance, pen up/down), and calculates the path to be displayed on the main canvas.
- **`storage.cpp/h`**: Manages the saving and loading of command history. This allows users to save their session to a text file for later use or external editing.

Class Relationships

The UML diagram (see Figure 2) illustrates the interactions between the main components:

- **`mainwindow`** aggregates both **`turtle`** and **`storage`** classes to coordinate their functionalities.
- **`turtle`** class handles core logic, including movement and drawing.
- **`storage`** class interacts with the **`mainwindow`** to save or load the command history.

Interfaces to External Libraries

The program uses the Qt library to create and manage the graphical user interface elements, including windows, input fields, graphics rendering and event handling. No additional external libraries are required.

Instructions for building and using the software

How to compile the program ('make' should be sufficient), as taken from the git repository.

1. Clone the Repository

Ensure Git is installed, then run:

```
git clone <repository_url>
cd <repository_directory>
```

2. Install Dependencies

The program requires the Qt framework and CMake.
- Qt Framework:

In Linux, run command `sudo apt install qt6-default`. In Windows/macOS, install **Qt Creator** from the official Qt website.
- CMake:

In Linux, run `sudo apt install cmake`. In macOS, run command `brew install cmake` and in Windows, download from **cmake.org**.

3. Build the Program

In the project directory, run:

`cmake .` configure the build process based on the `CMakeLists.txt` file in the project directory.

`make` compiles the source files and generates an executable file.

4. Run the Program

After building, run command `./turtle_1`

How to compile the program using Qt Creator:

1. Open the project by selecting the `CMakeLists.txt` file.
2. Build the project using the "Build" button, then run it via the "Run" button.

External Library Requirements

Qt Framework: Required for GUI development. Install the Qt SDK via the official website or a package manager (for example, qt6-default on Linux).

CMake: Used for project configuration and building.

How to Use the Software: A Basic User Guide

1. Launch the application: Either run the program directly from Qt Creator or execute the compiled binary.
2. Interact with the application: Use the command window to input commands:

- General Commands:

  `help`: Display usage instructions.

  `reset`: Clear the canvas and reset the turtle.

  `pen up / pen down`: Toggle drawing mode.

  `gameify`: Set a random target position for the turtle.

- Movement Commands:

  `forward [number]`: Move forward by a specified distance.

  `turn [degrees]`: Rotate the turtle by a specified angle.

  `go [x] [y]`: Move directly to a specified point.

- Drawing Commands:

  `star`: Draws a 5-pointed star using forward and turns.

  `triangle`: Draws an equilateral triangle.

  `square`: Draws a square with 90-degree turns.

  `rectangle`: Draws a rectangle with 90-degree turns.

  `circle`: Draws a circle by incrementally turning.

  `cyclohexane`: Draws a hexagon by repeating 6 sides.

  `house`: Draws a house with a rectangular base and triangular roof.

  `random`: Generates random turtle movement commands for drawing patterns.

  `spinning [number]`: Draws a rotating shape with a specified side.

3. Track and Modify Commands: View command history in the command window to track past actions or edit commands.
4. Save and Load Sessions: Use the Save option to store the command history in a **`.txt`** file and reload a saved session using the Load option, which replays the saved commands.

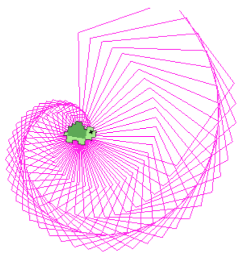


Figure 1: "spinning 5"
Spinning function creates a figure from shapes containing the user defined amount of corners.

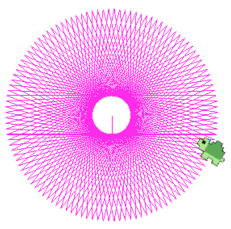


Figure 2: "random"
Function that always draws a different outcome.

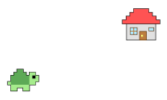


Figure 3: "gameify"
Function spawns a house at a random location and the user needs to guide the turtle back home.

Figures above were created to visualize different commands and they were captured by the programs image saving feature.

Testing

How the different modules in software were tested

The software modules were tested using a mix of automated unit tests and manual validation to ensure functionality and reliability. The **`Storage`** module was tested by verifying commands added to history, checking the order and size of the history list, clearing the history, and switching between the help display and history. The **`Turtle`** module tests focused on confirming the turtle's initial position, accurate movement, rotation handling, and reset functionality. Manual tests were performed on non-automated features, such as pre-drawn shapes, game-winning conditions, command queue processing, and color application in paths.

Description of the methods

Unit tests were created for each key feature, ensuring validation of command addition, history clearing, and turtle movement. The **Storage** module passed tests for history commands and model updates, while the **`Turtle`** module validated movement, reset, and rotation behaviors. Non-automated tests ensured correct visual elements and game mechanics. Tools like **`Valgrind`** were used for memory leak detection, and cross-platform compatibility was confirmed by testing on different systems.

Outcomes

The tests were successful, with all automated tests passing and confirming the expected behavior of both modules. These automated unit tests were ran every time the program was run ensuring functionality for key components. The tests would have always outputted a specific error text that tells what failed. If nothing failed, the tests outputted a success text informing that the program is viable. Non-automated tests addressed potential gameplay issues early in development. Memory leak detection was passed, and the project's cross-platform functionality was validated, ensuring the software's robustness for final release.

Features of the program:

- Turtle visualization with proper location and angle on a 2D field
- Basic commands: forward, turn, (pen) down, (pen) up
- Simple command line interface for entering the commands
- Different pen attributes such as color, line width
- Ability to save an image and turtle state, and later load it
- Ability to read command scripts for execution from file

Additional Features:

- Using QtMultimedia for sounds
- Enhancing the drawing field with a goal where the turtle needs to go

How does the program work?

The program is built using the Qt library, which simplifies the creation of GUI windows and populating them with widgets such as input fields and labels. The program begins in the main.cpp file, where the main window is instantiated. This main window contains an input field and a history box, which displays all commands issued to the turtle, as well as any errors or tooltips. Additionally, the main window hosts the "map" where the turtle graphics are displayed (see figure 1 below).

The user interface consists of two key files: mainWindow.cpp and inputController.cpp. These handle input validation, command history, and layout management. The actual turtle graphics are rendered in map.cpp, which manages location handling, grid formation, and line drawing. Turtle movements—such as "Forward", "Turn", "Pen up", and "Pen down" are processed in turtle.cpp.

Saving and loading the turtle's state is handled in storage.cpp, which uses the command history to track actions. Commands are saved in an ordered array of strings, allowing easy display of past commands and enabling quick restoration of previous states by replaying the command array. This array can be stored in a .txt file, which also allows external editing.

How is the program used?

1. Summarization:

The program allows users to control the turtle by entering executable commands. Users provide specific instructions to adjust turtle's movements and actions accordingly.

2. User Interface of the application:

The user interface of the application consists of two primary windows:
- Main window is the large display are where the movements of the turtle are shown. It visually represents how user commands are executed, displaying how the turtle moves according to given instructions and illustrating the path created.
- Command window is the smaller window that receives the command lines from users. It also shows a history of previous commands, allows users to monitor their inputs and modify or re-execute commands as necessary. User also can use "help" command to get support and read the instructions how to use the application efficiently.

3. Interaction between user and application

Users interact with the application by entering commands into the command window. The movements and actions of the turtle within the main window are controlled by these commands. The step by step process is as follows:

- Input command: Users enter the line of instruction into the command window.
- Command Processing: The application reads the input line, queues them for execution. The input controller makes ensuring that commands follow the right logic and syntax.
- Execution: The turtle executes the queued commands in the correct order and the main windows dynamically displays the movements of the turtle.
- History and Support: Users can view the command history to track their previous inputs and make adjustments to the turtle's path, access support if needed.

The high-level structure of the software

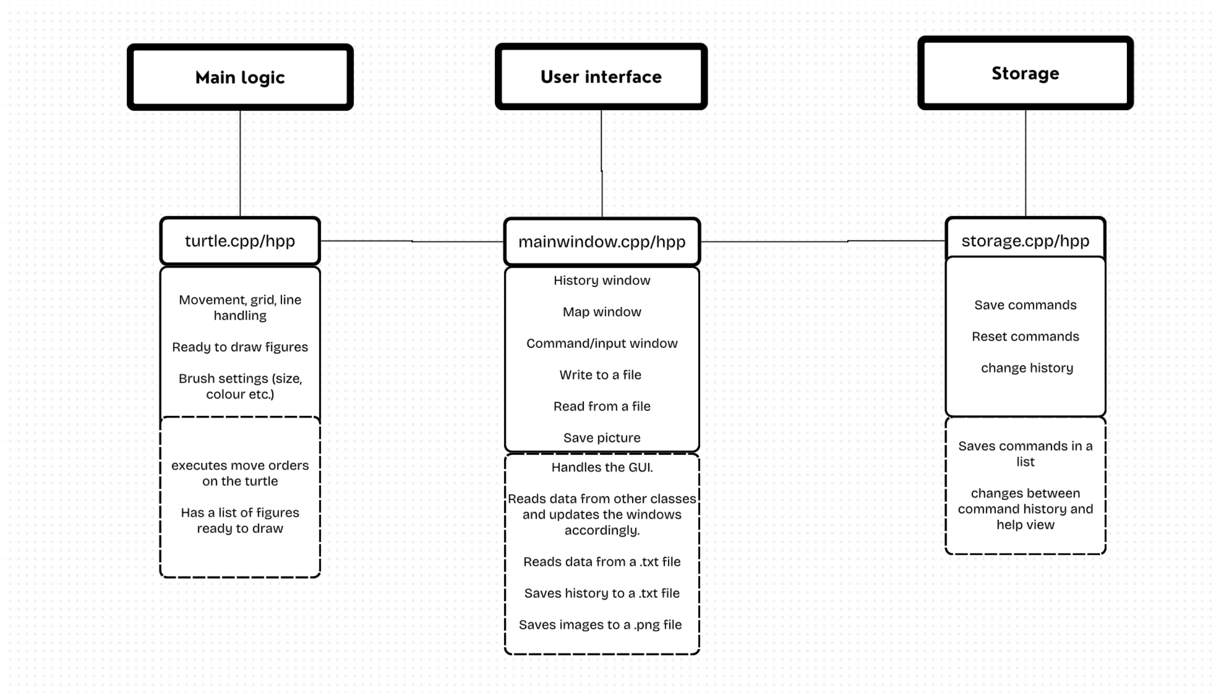

Main logic
User interface
Storage
turtle.cpp/hpp
Movement, grid, line handling
Ready to draw figures
Brush settings (size, colour etc.)
executes move orders on the turtle
Has a list of figures ready to draw
mainwindow.cpp/hpp
History window
Map window
Command/input window
Write to a file
Read from a file
Save picture
Handles the GUI.
Reads data from other classes and updates the windows accordingly.
Reads data from a .txt file
Saves history to a .txt file
Saves images to a .png file
storage.cpp/hpp
Save commands
Reset commands
change history
Saves commands in a list
changes between command history and help view

Figure 4: The basic structure and interaction between the classes.
The storage, main logic and user interface blocks represent modules.

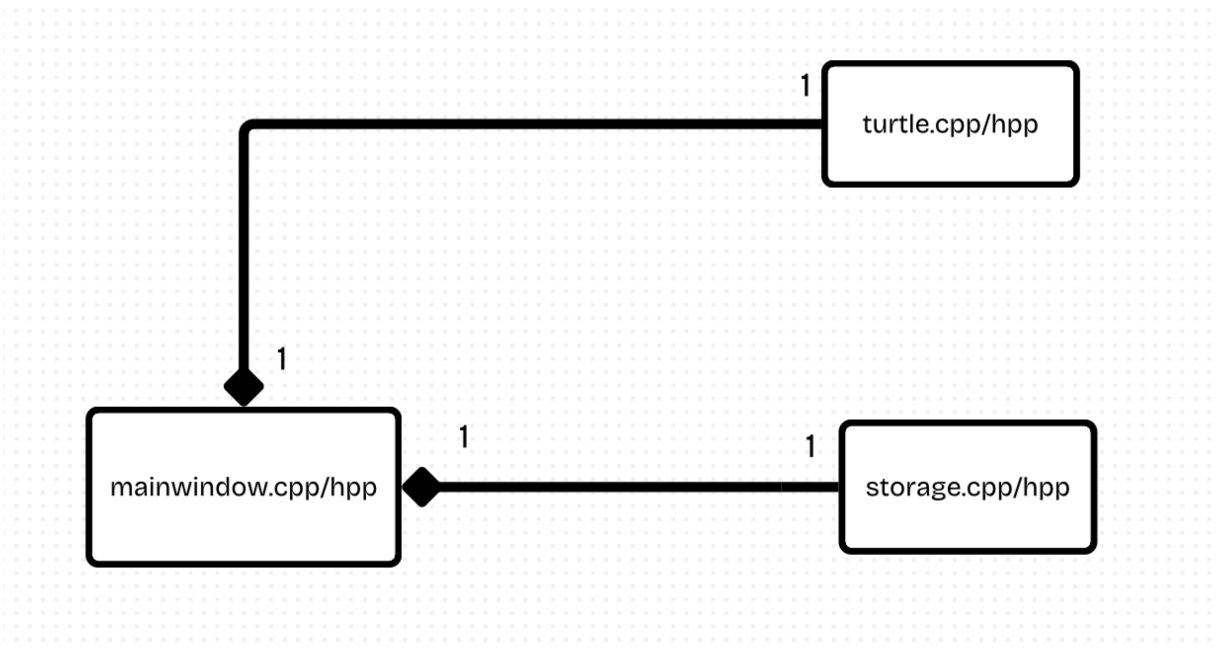


Figure 5: UML-class chart of Turtle graphics

The planned use of external libraries

1. Basic Qt tutorials

- Introduction to Qt Quick
- Introduction to QML
- Building with Cmake: Getting Started with CMake and Qt
- QML Integration Basics
- QML Modules
- How to Expose C++ to QML
- Introduction to Qt Quick 3D

2. GitHub example templates and tutorials

- ProgrammingPrinciplesAndPracticeUsingQt
- C++ Qt Game Tutorial 7 - Adding Sound Effects/Music

More detailed plans for the sprints

Our regular long meeting happens every Thursday afternoon. Here is the table denoting our phase goal and schedule:

Week	**Goal**	**Tasks Details**
Week 01 (Oct.23 ~ Oct.31)	Complete Project Plan	1. Installed the Qt; 2. Software architecture completed; 3. Project plan completed.

Week	**Goal**	**Tasks Details**
Week 02 (Nov.01 ~ Nov.07)	Complete basic functions	1. User interface implemented; 2. Make the turtle move using the keyboard;
Week 03 (Nov.08 ~ Nov.14)	Optimize basic functions	1. Present the turtle trajectory; 2. Start and end control of the turtle; 3. Polish user interface.
Week 04 (Nov.15 ~ Nov.21)	Complete advanced functions	1. Change the colour and line features; 2. Detect if the turtle has reached the boundary and send warnings; 3. Add sounds and a 3D turtle.
Week 05 (Nov.22 ~ Nov.28)	Optimize advanced functions	1. Add some fixed drawing features; 2. Optimize storage functions.
Week 06 (Nov.29 ~ Dec.05)	Submit demo to advisor	1. Meeting with advisor; 2. Polish project.
Week 07 (Dec.06 ~ Dec.12)	Project final commit to git	1. Make the final submission.

Work log for the sprints

The previous plan was followed well, here is the detailed description of the practical work including the completed tasks, time investment and members' contribution.

Stage	Tasks completed	Members' contribution
Plan the project (Oct.23 ~ Oct.31)	1. Configured environment; 2. Software architecture completed; 3. Project plan completed.	All members completed.
Basic features (Nov.01 ~ Nov.07)	1. User interface implemented; 2. Made the turtle move using simple commands; 3. Made the trajectory show up; 4. Loaded and saved figures;	**Hugo:** CMakeLists.txt, Forward & Turn, Turtle Image, UI Styling **Giang:** Created load function to read commands from file **Mathias:** Created command history, storage class and history window **Xiwei:** Visualized the trajectory line.
Optimize basic features (Nov.08 ~ Nov.14)	1. Made turtle move smoothly; 2. Added more commands functions; 3. Boundary detection; 4. 'help' function;	**Hugo:** UI tweaks, Drawing, go function **Giang:** Fixed upload feature and add sound **Mathias:** Help function, fixed input validation **Xiwei:** Made turtle move smoothly.
Complete advanced functions (Nov.15 ~ Nov.30)	1. Changed the colour and line size; 2. Added sounds and a 3D turtle; 3. Designed 'gameify';	**Hugo:** Boundary detection, restart command, fixed non-axial movement bug, queue for commands **Giang:** Designed drawing

Stage	Tasks completed	Members' contribution
	4. Designed variable automatic drawing shapes.	functions to draw star, circle, rectangular, triangle, cyclohexane, house, random and spinning, fix few bugs **Mathias:** unit tests for storage and turtle classes, save image function, gameify function **Xiwei:** Changed the colour and line size
Project final validation and testing (Dec.01 ~ Dec.12)	1. Valgrind test; 2. Functions polishment; 3. Final documentation completed; 4. Added sound when the movement starts.	**Hugo:** Added a house image, little tweak to gameify, fixed valgrind errors, save state function, created doxygen.pdf **Giang:** Adding sound feature, wrote overview, software structure, instructions for building and using software, testing description for final documentation. **Mathias:** UML & high-level structure chart, game testing, fixed small gameplay bugs. **Xiwei:** Final documentation completed.