ELEC-A7151 Project Plan

leevi.kahri@aalto.fi
nicke.nordstrom@aalto.fi
andrei.aaltonen@aalto.fi
martin.j.haggblom@aalto.fi

1. Features

The goal is to create a graphical user interface representation of turtle graphics where an image is drawn based on user instructions. A rough sketch of the GUI is shown in picture below. The user inputs command into a command line. The command can be for example “walk 100” or “turn 50”. Each time the user is informed whether the command was successful or not.

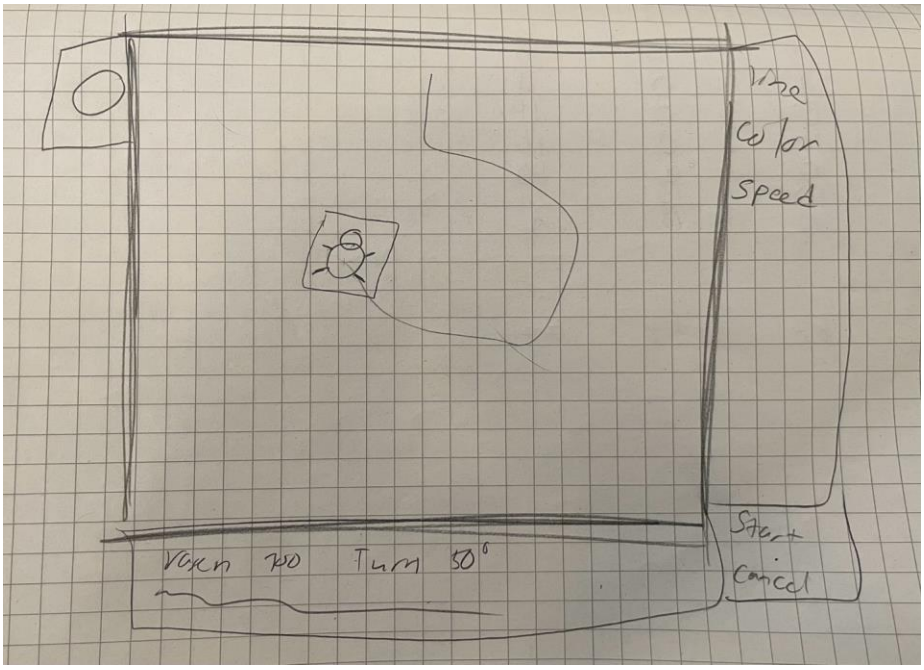
When all commands have been input the user can press the start button in the bottom right corner and the turtle starts moving in the big window. The command being executed is shown at the top of the screen. If the user is dissatisfied with the result he can cancel the execution via the cancel button in the bottom right corner. This assures that a long execution gone south in the beginning doesn’t have to be viewed all the way through.

Command “remove” removes the previous input command and command “remove all” removes all commands.

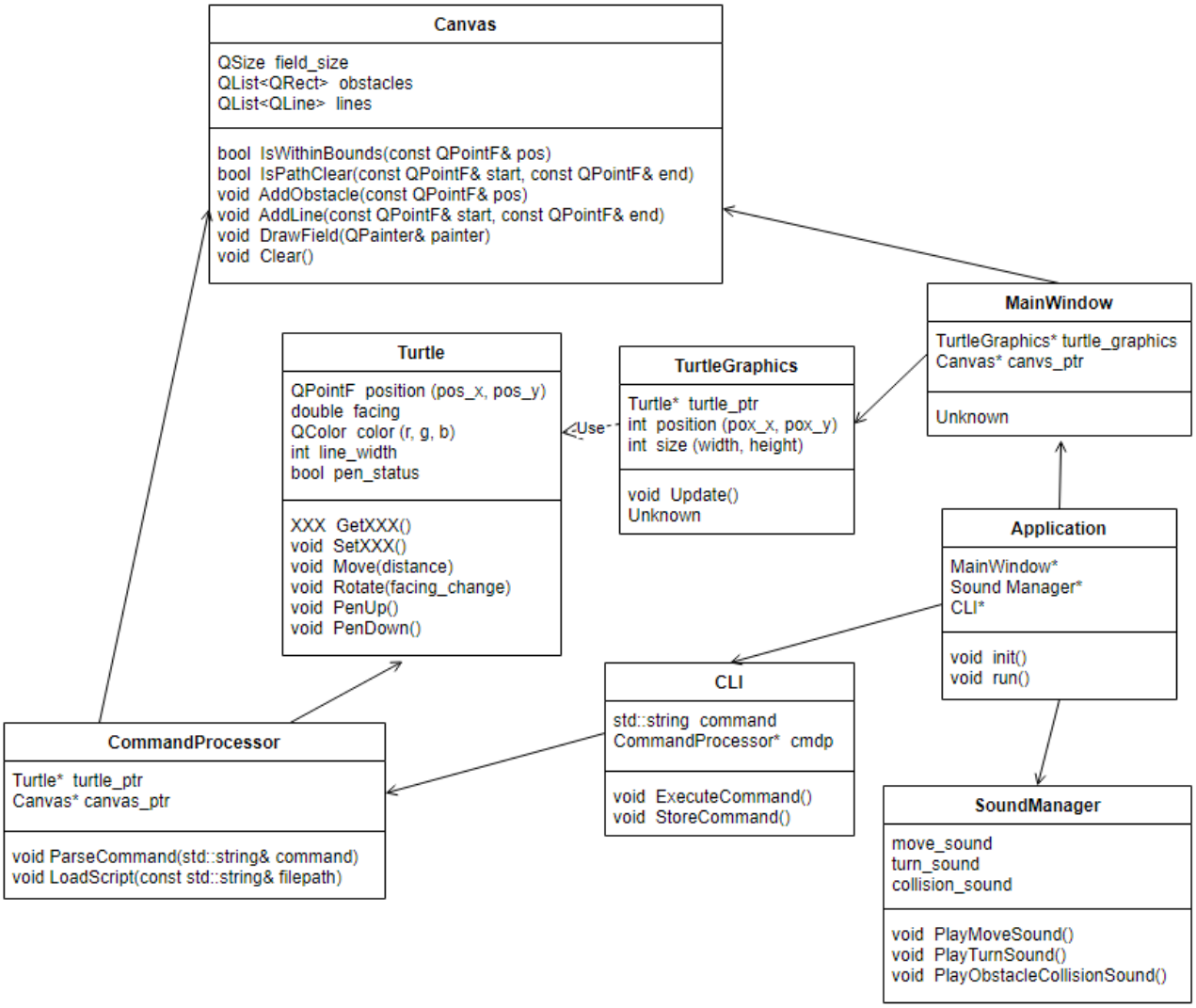
As the turtle walks it leaves behind a line. It is possible to modify the speed of turtle, color and width of the line and possibly some sound effect linked to certain commands. All these options are selected from a menu on the right edge of the GUI.

If the turtle hits the edge of the drawing board, it bounces of in the same angle as it arrived. So, it is possible to draw some interesting shapes just by letting the turtle run straight for a long time.

A list of commands can be saved to a file and retrieved from a file trough the command line. The commands for such actions are just ”save – filename” and “load –filename”. A list of files containing instructions for basic shapes is listed on the left side of the GUI.



2. Classes



- The specific relationship (arrows) between classes is not entirely clear yet.
- To separate the CLI and CommansProcessor appears like a reasonable choice as it allows the CLI class to focus on the graphical appearance, and with reading and writing.
- MainWindow implementation is not yet clear at this stage.
- The separation of Turtle from TurtleGraphics may not be necessary.
- Application should be the highest-level object in the program.
- What class should handle 3D effects is still uncertain, however it could be Canvas to e.g. add shadows to line.

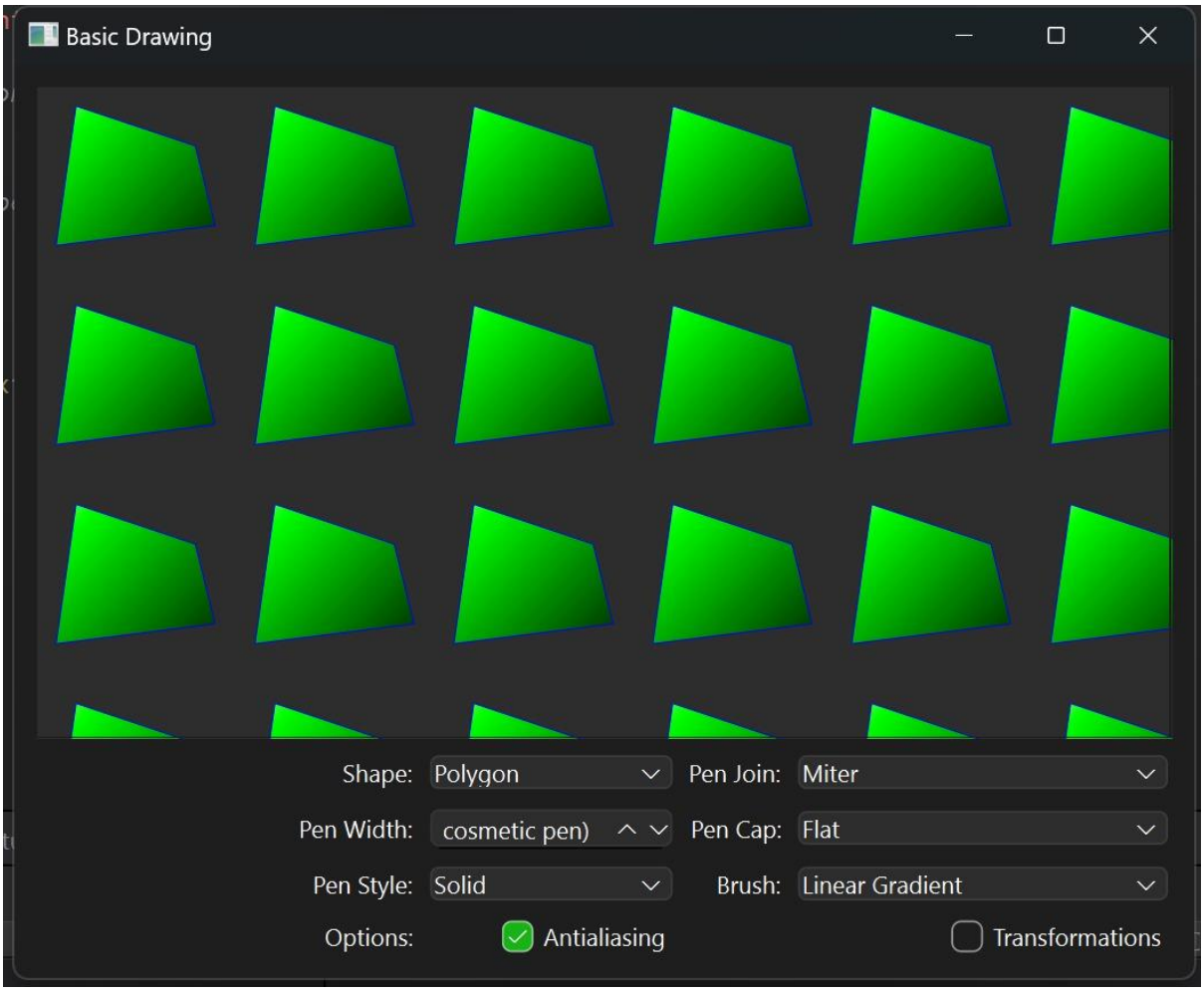
The class diagram above is a rough sketch of how the program could be organized. There are multiple questions still unanswered, but it gives a acceptable starting point for development. The first classes that need to be implemented would be: Application, MainWindow, Turtle (& TurtleGraphics), Canvas.

3. Libraries

Qt offers a lot of preset tools consisting of many useful classes and functions. The following will be utilized in our project:

- **Qt Widgets**
  - Provides a set of UI elements to create desktop-style user interfaces.
  - Widgets can display data and status information, receive user input, and provide a container for other widgets that should be grouped together.
  - Interesting classes for us to use within the Widgets include (but not limited to):
    - QWidget: The base class of all user interface objects.
    - QApplication: Manages the GUI application's control flow and main settings.
    - QLabel: Text or image display.
    - QPushButton: Command button.
    - QLineEdit: One-line text editor.
    - QSlider: Vertical or horizontal slider.
    - QCheckBox: Checkbox with a text label.
    - QProgressBar: Horizontal or vertical progress bar.

- **Qt Paint System**
  - Primarily based on the QPainter, QPaintDevice, and QPaintEngine classes, which enable painting on screen.
  - Other interesting classes for us to use within the paint system include (but not limited to) :
    - QPoint: Defines a point in the plane using integer precision.
    - QImage: Handles image data.
    - QLine: Describes a finite length line on a two-dimensional surface.
    - QPainterPath: provides a container for painting operations, enabling graphical shapes to be constructed.
    - QPainterPathStroker: Used to generate fillable outlines for a given painter path.
    - QPen: Defines how a QPainter should draw lines and outlines of shapes.
    - QBrush: defines the fill pattern of shapes drawn by QPainter.
    - QPixmap: An off-screen image representation that can be used as a paint device.
    - QColor: Provides colors based on RGB, HSV or CMYK values.



4. Spring Planning

The project is divided into four sprints as following:

- **Sprint 1: Project planning** (18.10. - 1.11.). By the end of the sprint the project plan should be pushed to git repository
- **Sprint 2: Initial implementation** (1.11. - 15.11.). By the end of the sprint there should be an initial running version of the project.
- **Sprint 3: Features complete** (15.11. - 29.11.). By the end of the sprint the main features should be implemented, even if not finalized.
- **Sprint 4: Finalization** (29.11. - 13.12.). The project output, including documentation, is finalized, and tested.

The tasks in prioritization order for each sprint are as following:

Sprint, DL Min. target	Task	Basic/Additional/Advanced/Extra feature
#1, **1.11**	Project Plan	NA
#2, **15.11** **Min. target:** Turtle draw hard coded figure	Board & MainWindow 2D	Basic
	TurtleGraphics	Basic
	Drawing options	Additional
#3, **29.11** **Min target:** Turtle draw from command line	Current command executed	Basic
	Command line	Basic
	Load and save file	Additional
	3D Graphics	Advanced
	Sound effects	Advanced
	Obstacles	Additional
	Cancel execution	Extra
	Animated turning	Extra
	Rainbow pen	Extra
#4, **13.12**	Finalization + documentation	NA

Table 1. High-level tasks per sprint in prioritization order

It is planned that if some of the Sprint #2 features are not finalized by the DL 15.11, those features will be finalized in the beginning of Sprint #3. Since the tasks are listed in prioritization order per Sprint, it might be that some of the lower prioritized features listed in Sprint #3 might not be implemented if time limitations occur. The team is tough at this point optimistic that all tasks listed in Table 1, will be implemented. From Table 1 it can also be noted that the absolute minimum target is to achieve a software where the Turtle draw from the command line after Sprint 3.

Regarding way of working the group has decided to:

- Focus the lion part of the work of the project during the beginning of the weeks i.e. Mondays and Tuesdays.
- Despite the main work is done during Mondays and Tuesdays, the group will follow the Discord channels during the rest of the weeks.
- As a starting point, the group will try out pair programming during sprint 2.

Project progress tracking will be implemented as following:

- Weekly remote sprint meetings. Group
  - Mondays @ 9:00-10:00
- Donatello rolling meeting memo for easy collaboration during meetings. Group
- Discord channels.
  - “Donatello. OOP C++” for continuous Group discussions
  - “turtle-group3” channel for continuous Group + Advisor + Qt discussions
- Issue board for tracking implementation of features. Group (+ Advisor)
- Biweekly remote sprint review meetings. Group + Advisor
  - Sprint 1 review: Tue 29.10 @ 10:00-10-45

    - Sprint 2 review: Tue 12.11 @ 9:00-10:00
    - Sprint 3 review: Tue 26.11 @ 9:00-10:00
- Project demo at campus. Group + Advisor (+ Qt)
    - Wed 4.12 @ 9:00-10:00