Johannes Gutenberg-Universität Mainz
Institut für Informatik
Einführung in die Softwareentwicklung
Prof. Dr. Michael Wand
Dr. Christian Ali Mehmeti-Göpel
Visual Computing Group

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Einführung in die Softwareentwicklung

ÜBUNGSBLATT 03: Vektorgraphik Teil 1 – Freestyle

12. Mai 2025

Über dieses Übungsblatt

Dieses Übungsblatt behandelt das erste Projekt, welches wir in EIS in Angriff nehmen wollen: Wir programmieren ein vektor-orientiertes Zeichenprogramm (bzw. einen kleinen Prototypen einer solchen Anwendung). Wie in der Vorlesung erklärt, geht es darum, die Entwicklung einer etwas komplexeren interaktiven Anwendung (komplexer als eine einfache Teilaufgabe eines Übungszettels) anzupacken, und dabei typische Probleme und deren Lösungen durch verschiedene architektonische Tricks, die in den letzten Jahrzehnten entwickelt wurden, kennenzulernen.

Das gesamte Projekt wird sich über sechs Wochen und drei Übungszettel ziehen:

- **Übungsblatt 03 (dieses Blatt):** Wir erstellen eine Grundversion des Programms. Nutzen Sie die Programmiertechniken, mit denen Sie vertraut sind und versuchen die Aufgabe so gut umzusetzen, wie es ihnen mit Ihrem aktuellen Wissen gelingt. Das Übungsblatt gibt dazu auch ein paar Tipps.
- **Übungsblatt 04 (nächstes Blatt):** Wir werden die Architektur nochmal überarbeiten in Bezug auf objektorientierte Konzepte (falls nötig). Außerdem werden wir weitere Features einbauen und zusätzliche Programmiertechniken kennenlernen.
- **Übungsblatt 05 (übernächstes Blatt):** Nachdem wir in der Vorlesung die Trade-Offs von funktionalem und objektorientierten Design diskutiert haben, bauen wir ein zusätzliches Funktionales Interface ein, um neue Operationen zu ermöglichen, die vorher nicht realisierbar waren (dafür werden andere Features mit dem Interface erschwert). Auch hier wird es weitere Aufgaben geben, die andere Aspekte beleuchten.

Ich empfehle Ihnen, die Serie von Übungen zu nutzen, um Erfahrungen mit Softwareentwicklung zu sammeln. Auch wenn Sie auf diesem Blatt noch einige Dinge machen, die sich später als überflüssig herausstellen, sammelt man Erfahrungen, welche Vor- und Nachteile bestimmte Entwurfsmuster und –strategien haben.

Noch ein Tipp: Lesen Sie alle Teilaufgaben durch, bevor Sie mit der Bearbeitung anfangen!

Abgabe: (Abgabe 7. SW)

Für dieses Übungsblatt sind zwei Wochen Bearbeitungszeit vorgesehen. Das Übungsblatt muss entsprechend bis zum **25. Mai 2025, 23:59h** in LMS abgegeben werden. Laden Sie dazu den Source-Code aller Aufgaben hoch. Die Ergebnisse müssen in den Übungen zwischen dem 26.-30. Mai 2025 vorgestellt werden. Beachten Sie hierbei die Ausweichregelungen für die Feiertage!

Aufgabe 0: Design (unbewertet)

Lesen Sie sich alle Aufgaben durch, und überlegen Sie sich, wie Sie Ihr Programm strukturieren möchten. Dabei ist sicherlich ein Aspekt, dass man das Programm im Laufe des Semesters noch erweitern und ausbauen möchte (neue Features). Machen Sie sich schon einmal Gedanken darüber, wie Sie dem Rechnung tragen können.

Tipp: Es ist sinnvoll, bodenständig zu bleiben. Auch wenn es viele elegante architektonische Tricks gibt, fährt man oft am besten damit, die Werkzeuge und Techniken einzusetzen, die man gut kennt und mit denen man Erfahrung hat. Falls Ihre Lösung nicht besonders elegant ist, macht das nichts – bewertet wird nur, ob es funktioniert. Es besteht noch Gelegenheit, das ganze strukturell weiterzuentwickeln. Falls Sie schon Techniken kennen, die hier helfen können, spricht nichts dagegen, diese einzusetzen.

Aufgabe 1: Rahmenprogramm (10 Punkte)

Hinweis: Sie sollten sich natürlich gut überlegen, in welchen Schritten Sie die verschiedenen Ausbaustufen und Features erstellen wollen (hier gibt es sicher viele Wege zum Ziel); man muss diese Aufgabe nicht unbedingt als erste lösen (mir persönlich fiel es so am leichtesten).

Zunächst soll ein Rahmenprogramm erstellt werden, mit dem Dokumente verwaltet werden können. Dieses Rahmenprogramm kann auf der z.B. Lösung des letzten Übungsblattes aufbauen, und soll die folgenden Features bieten:

- Erstellen Sie ein Widget, auf dem Sie die Vektorgraphik darstellen können, die Sie in Aufgabe 2 erstellen.
- Stellen Sie eine Möglichkeit bereit, um Kommandos aufzurufen (z.B. via Menü, Toolbar oder mit anderen Buttons/Dialogen), um die später benötigten Test durchführen zu können.
- Es sollte möglich sein, das Programm ordnungsgemäß zu beenden.

Tipp: Aufgabenblatt 02 stellt einen sinnvollen Ausgangspunkt dar – die Musterlösung steht auf LMS zum Download bereit.

Aufgabe 2: Eine Bibliothek für Vektorgraphik (50 Punkte)

Diese Aufgabe stellt den Kern dieses Übungsblattes dar. Es geht darum, eine Bibliothek für Vektorgraphik zu erstellen, und diese zu testen. Wie in der Vorlesung diskutiert, sollte die Bibliothek ermöglichen, geometrische Objekte zu verwalten und zu „rendern", also als Bitmap-Graphik darzustellen.

Auf diesem Übungsblatt sollen drei Arten von Objekten unterstützt werden:

- Achsenparallele Rechtecke (mit beliebiger Position wie auch Breite/Höhe)
- Kreise (mit beliebigen Mittelpunkt und Radius)
- Sterne (mit fester Anzahl von „Zacken"). Der dritte Typ wird in Aufgabe (d) hinzugefügt.

a) Erstellen Sie Datenstrukturen, die

- Rechtecke und
- Kreise

repräsentieren, wobei man jedes Mal neben den geometrischen Parameter (Größe/Position) mindestens die Farbe für die Füllung (in RGB) und die Farbe für den Rand (auch in RGB) frei wählen kann. Erstellen Sie danach eine weitere Datenstruktur, die

- eine ganze Zeichnung repräsentiert.

Überlegen Sie sich, wie Sie das ganze am besten strukturieren können. Denken Sie daran, dass in Aufgabenteil (d) noch ein weiterer Typ hinzugefügt wird.

Tipps zur Umsetzung:

Am einfachsten kann man dies lösen (dies ist nicht die eleganteste Lösung), indem man in Python separate Klassen für alle drei Typen anlegt. Die dritte (gesamte Zeichnung) wird dabei eine Python-Liste enthalten, in der alle in der alle Objekte eingetragen sind, die in der Zeichnung vorkommen.

Beispiel:

```
class Rectangle:
    # ... define structure, e.g. using __init__ or using @dataclass ...


class Circle:
    # ... same here ...


class Scene:
    # ... contains some list ...
    # Tip: Python 3.10+ permits the MyPy-type: List[ Rectangle | Circle ]
    # for type annotations (if desired)
```

b) Nun soll die Szene dargestellt werden. Schreiben Sie dazu ein Unterprogramm (welches ggf. weitere Unterprogramme nutzt), welches eine gegebene Szene in ein Rasterbild umwandelt. Das Unterprogramm bekommt dabei (mindestens) die folgenden Parameter übergeben:

- Eine Referenz auf ein Zeichenwerkzeug
  (in Qt: vom Typ „`QPainter`“, in AWT: vom Typ „`Graphik`“)
- Einen Bildausschnitt, um den angezeigten Ausschnitt der Zeichnung festzulegen. Das kann man z.B. wie folgt definieren:
  - Breite und Höhe des Pixelbildes (z.B. Fenster) in Pixeln
  - Linke obere und rechte untere Ecke des Zeichnungsausschnitt in „Weltkoordinaten“

Das Unterprogramm soll den „Seiteneffekt“ haben (d.h., es soll das tun), mit dem Zeichenwerkzeug ein Bild der gesamten Szene zu erstellen.

Hinweise zur Umsetzung:

Wie in der Vorlesung angesprochen, müssen Sie eine „Viewport-Transformation“ durchführen, bei der abstrakte Weltkoordinaten der Rechtecke/Kreise/etc. in Pixelkoordinaten umgewandelt werden. Danach übernimmt `QPainter` (oder sein Pendent in SWING) das zeichnen. Ein „Clipping“ (Abschneiden an den Grenzen) ist nicht nötig – die Bibliothek kann bereits damit umgehen, wenn Koordinaten den sichtbaren Ausschnitt verlassen und schneidet selber entsprechend ab. Die Viewport-Transformation ist nochmals in den folgenden beiden Abbildungen illustriert:

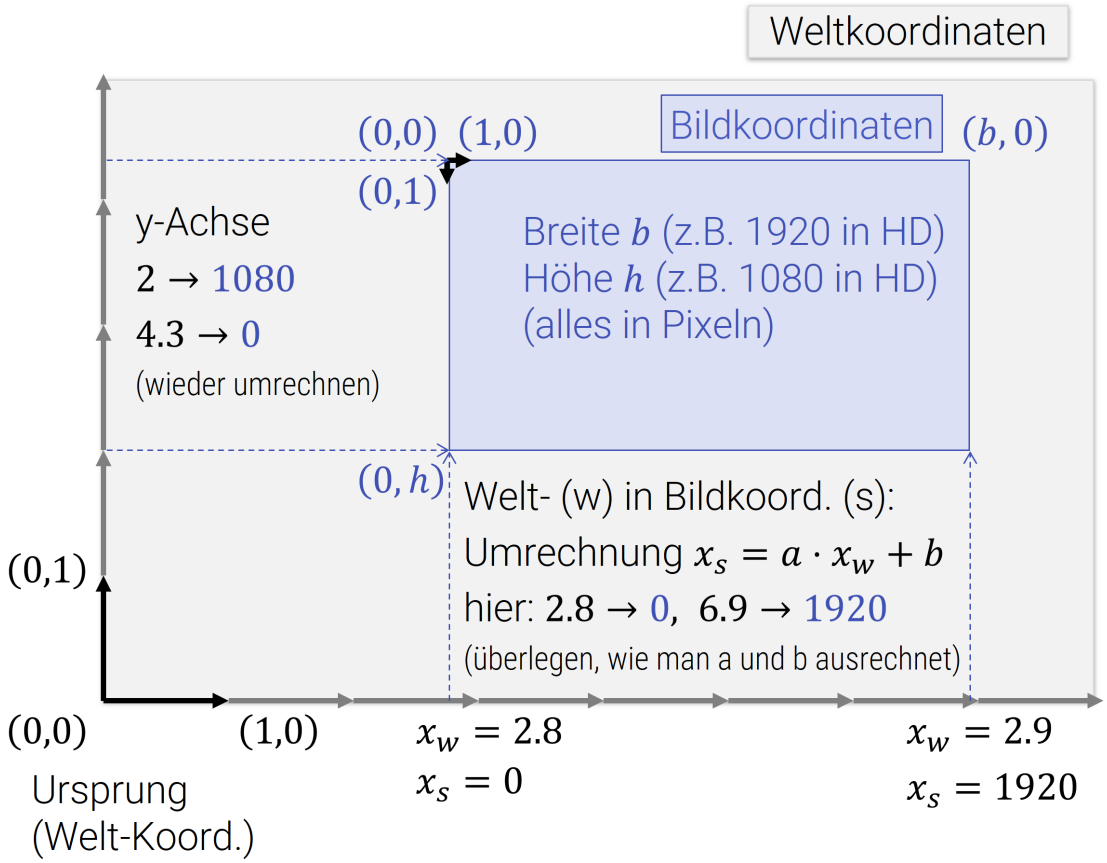

Abbildung 1: „View-Port-Transformation“

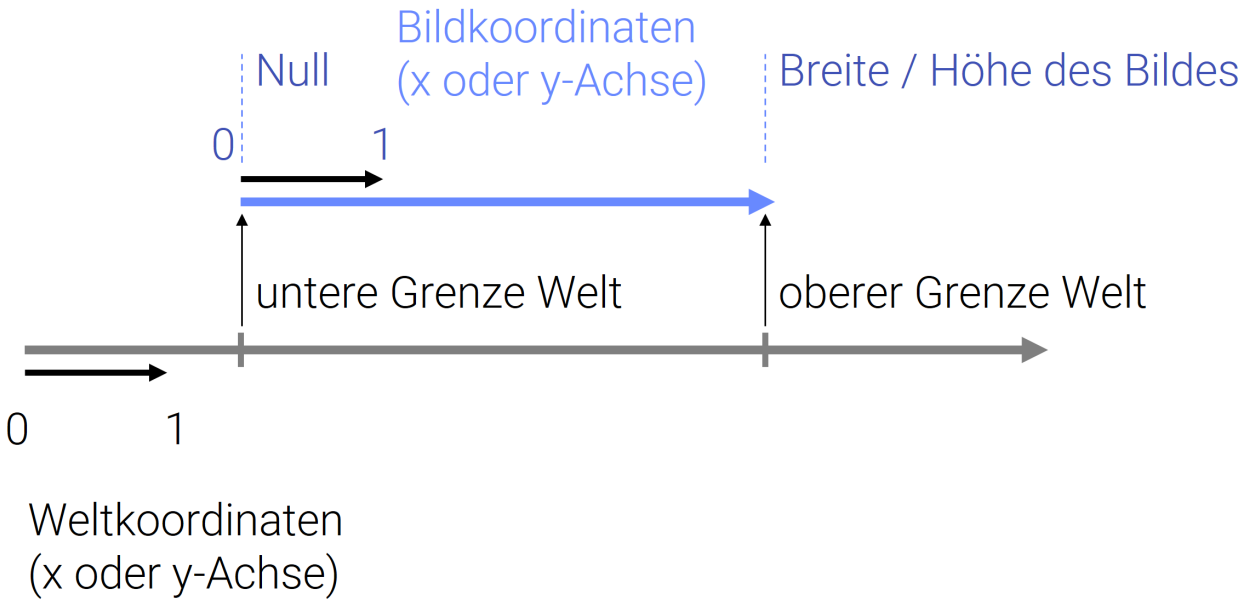

Abbildung 2: „View-Port-Transformation“ entlang jeweils einer der beiden Achsen – wenn man diese Abbildung lange genug anstarrt, sollte man auf die richtige Formel kommen :-)

Tipp: Bei der Umsetzung kann viel schiefgehen (z.B. Rechenfehler). Überlegen Sie sich gut, wie Sie ihren Code sinnvoll testen / schrittweise auf-/ausbauen können.

c) Fügen Sie nun Ihrem Rahmenprogramm eine Funktion (Bedienelement im GUI) hinzu, mit der Sie zwei verschiedene Testszenen erzeugen können. Diese sollten ungefähr die in Abbildung 3 dargestellte Formen haben (genaue Abmaße egal/freie Wahl; auch die Wahl der Farben ist beliebig, solange mehr als eine vorkommt):

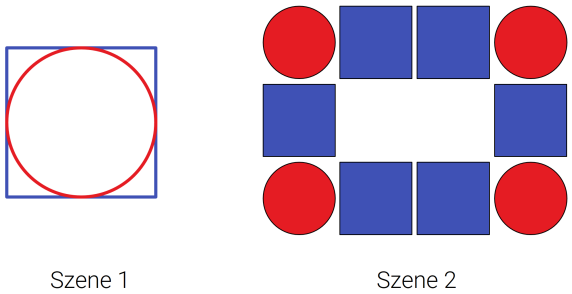

Abbildung 3: Zwei einfache Testszenen. Genaue Maße und Farben sind frei wählbar (mehrere Farben in der gleichen Zeichnung sind aber Pflicht)

Der Bildausschnitt, der vom Programm angezeigt wird, sollte so gewählt sein, dass die ganze Szene (gut) zu erkennen ist (also nichts abgeschnitten, und nicht extrem klein).

d) Erweitern Sie Ihr Programm nun so, dass ein „Stern“ als drittes Primitiv (möglicher Typ von Objekt in der Zeichnung) unterstützt wird. Auch hier sollte man Farbe von Füllung und Rand einstellen können, und der Mittelpunkt und die Größe des Sterns sollen frei wählbar sein. Die genaue Form ist

freigestellt (solange es ungefähr wie ein Stern aussieht). Es soll möglich sein, die Funktion im GUI zu testen (z.B. via Integration in die Testszene aus Aufgabenteil (b)).

Tipp: Die Punkte

$$\begin{pmatrix} R \cdot \cos\alpha \\ R \cdot \sin\alpha \end{pmatrix}, R \geq 0, \alpha = 0 \dots 360° \; (entspr. \, 0 \dots 2\pi)$$

Liegen alle auf einem Kreis um den Ursprung mit Radius R. Wählt man $\alpha = 2\pi\frac{k}{N}$ für $k = 1 \dots N$ und eine feste Zahl N, so bekommt man gleichmäßige Punkte auf dem Kreis heraus. Nun muss man nur noch den Radius zwischen zwei Werten hin- und herwechseln.

Hinweis: Es ist natürlich erlaubt, alle drei Primitive von Anfang an zu berücksichtigen. Ich empfehle aber, den Stern später „einzubauen" um zu sehen, wie viel Aufwand die Erweiterung des Programms macht. Notieren Sie Ihre Erfahrungen und tragen Sie diese in den Übungsgruppen vor (*unbewertet*).

Aufgabe 3: Interaktion – Visualisierung der Vektorgraphik (20 Punkte)

Erweitern Sie Ihr Rahmenprogramm so, dass man den Bildausschnitt frei einstellen kann: Welcher Ausschnitt der Zeichnung im Fenster angezeigt wird (Zoom-Faktor, Verschiebung des Ausschnitts) soll frei wählbar sein.

Die Gestaltung des GUIs ist freigestellt – Sie können das (elegant) mit Mausbewegungen steuern (z.B. ziehen am Fenster für Verschiebungen, ziehen mit anderer Maustaste oder in anderem Modus für Zoomen) oder auch ein Eingabefeld (z.B. `QDoubleSpinBox` für die Eingabe von Zahlen im Fließkommaformat) vorsehen. Es ist auch möglich, Knöpfe für rechts/links/oben/unten und rein-/rauszoomen vorzusehen, oder „Scrollbars" zu nutzen. Alles, was Verschieben und Zoomen voll unterstützt ist „richtig" im Sinne der Aufgabe.

Aufgabe 4: Fortgeschrittene Interaktion – Erzeugen von Zeichenobjekten (20 Punkte)

Erweitern Sie nun Ihr Programm so, dass Sie einer bestehenden Szene neue Primitive (Rechtecke, Kreise, Sterne) hinzufügen können. Hierzu stehen zwei Varianten zur Auswahl. Die erste ist einfacher, und gibt 10 Punkte, die zweite ist schwerer und gibt die vollen 20 Punkte:

- **Einfacher:** Es gibt eine Funktion im GUI (Button oder Menüpunkt) mit dem ein neues Primitiv (alle drei zur Auswahl) hinzugefügt werden kann. Mit geeigneten Bedienelementen (z.B. `QDoubleSpinBox` für die Eingabe von Zahlen im Fließkommaformat) kann dann eingestellt werden, wie das neue Objekt aussehen soll. Abbildung 4 zeigt ein „Mock-up" von einer möglichen Implementation.
- **Schwieriger:** Man wählt auf geeignete Weise einen Zeichenmodus (Rechteck, Kreis, Stern) im GUI (z.B. über eine Toolbar) und klickt direkt mit der Maus in das Fenster. Bei gedrückter Taste zieht man einen Bereich auf, in den das Objekt eingepasst wird (das Rechteck kann man so direkt zeichnen, bei Kreisen/Sternen würde der Radius durch das Ziehen der

Maus gesteuert). Ohne die Steuerung der Größe (nur Mittelpunkt wird angepasst) gibt es (auch) nur 10 Punkte.

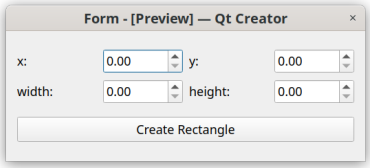

Abbildung 4: Link – so könnte ein einfacher Dialog aussehen, um ein neues Objekt zu erzeugen. Rechts: Mit der Maus „aufziehen" ist natürlich schicker, aber schwieriger zu programmieren.

Tipps zur Umsetzung der Interaktionen

In den Aufgaben 1c, 2, 3 und 4 sollen interaktive Werkzeuge programmiert werden. Es ist nicht trivial, eine Anwendung so zu strukturieren, dass dies übersichtlich und erweiterbar bleibt[1]. Wir werden in der Vorlesung noch einige Ansätze kennen lernen. Einige sehr einfach umsetzbare Methoden hier als Anregung in willkürlicher Reihenfolge (experimentieren Sie aber ruhig mit eigenen Ideen):

- Die Vogel-Strauß-Option: Bei einem kleinen Programm kann man es „hardcoden" wie in Blatt 01/02. (Einfach Member-Variablen für die UI-Elemente in das Hauptfenster und die Event-Behandlung programmieren). Für diese Blatt vielleicht gar nicht verkehrt.
- Der „`QAction`"-Mechanismus (Blatt 02), der in Qt schon eingebaut ist, hilft dabei Menüs und Toolbars besser zu strukturieren. Bei der Behandlung von selbstprogrammierten Mausereignissen / eigenen Widgets hilft das allerdings erstmal nicht weiter.
- Es ist sinnvoll, jedem Datenobjekt einen Besitzer zu geben, also hier z.B. die Szene in einem Fenster oder einem Widget (und nur dort) zu refrenzieren. Die Szene „besitzt" dann wiederum die Primitive.
- Zum Anzeigen der Szene sollte man, wie auf Blatt 02 geübt, ein eigenes Widget definieren, dessen Aufgabe es ist, Szenen anzuzeigen (vielleicht „`SceneViewerWidget`"?)
- Es kann hilfreich sein, dem Programm (oder auch Teilen davon, wie dem Widget, das die Szene anzeigt) einen Modus zu geben, z.B. Stern-generier-Modus, Rechteck-generier-Modus, Zoom-Modus, …). Weiterhin hätte jeder Modus wieder Unter-Zustände, wie z.B. Maustaste gedrückt/losgelassen beim Zeichnen eines Rechtecks. Im Laufe der Interaktion wechselt man auf bestimmte Wege durch diese Zustände durch (man nennt das „endliche Automaten" in der Theorie, und das wird für UIs gerne als Modell benutzt).

[1] Ich würde aus meiner persönlichen Perspektive soweit gehen, zu sagen, dass eine gute Strukturierung von GUIs im Sinne von erweiterbar und *wirklich einfach* umzusetzen vielleicht wirklich noch eine offene Forschungsfrage ist, aber ich bin kein Experte in HCI/UI-Research.