Inhaltsverzeichnis

1. Lösungsidee

Es soll der (bereits in der Übung begonnene) Grafikeditor um eine Navigationsleiste sowie einige brauchbare Funktionen erweitert werden. Nach der Übungseinheit konnten bereits einzelne Shapes ausgewählt und gezeichnet (aufgezogen) werden. In weiterer Folge wurden Menüpunkte wie Pen, Brush, Background hinzugefügt, mit denen die Farben der nächsten Figur sowie der Hintergrund der gesamten Applikation geändert werden können.

In weiterer Folge wurde eine Sidebar implementiert, mit der ebenfalls die Shapes ausgewählt werden können. Die derzeit Ausgewählte Shape wird immer optisch in der Farbe gelb hervorgehoben. Die Sidebar wurde mit einem fix positionierten Rectangle gezeichnet. Wenn der Benutzer das Fenster vergrößern oder verkleinern sollte, so wird der Eventlistener on_size angestoßen und die Navigationsleiste wird auf die neue Höhe des Fensters angepasst. Die Breite bleibt immer konstant bei 40px. Die Icons in der Navigationsleiste wurden ebenfalls fix positioniert und haben eine Größe von 20 x 20 px. Selbstverständlich wird die aktuelle Shape in der Sidebar auch aktualisiert, wenn der Benutzer diese über das obere Menü ändert.

Für eine Selektion einer Shape wurde ein eigenes Symbol (Pfeil) eingeführt. Dieser Menüpunkt ist sowohl im oberen Menü als auch im linken Menü auswählbar. Nach der Selektion einer Shape mit dem Selektionstool wird die ausgewählte Shape mit einem eigenen unique_ptr behandelt und somit aus dem shapes vektor entfernt. Zeitgleich werden die Eckpunkte der ausgewählten Shape (4 x 4 px rectangles) eingezeichnet. Diese Selektionsmarker werden in einem eigenen unique_ptr vektor verwaltet. Sollte der Benutzer ein anderes Tool auswählen, so wird die selektierte Shape wieder in den Vektor hinzugefügt und die Selektionsmarker gelöscht.

Wenn eine Shape mit dem selektiert wurde, so können die Operationen Cut, Copy und Paste auf sie angewandt werden. Dazu wurde ein unique_ptr clipboard_shape eingefürt, der eine Shape in der Zwischenablage verwaltet. Wenn die die Operation cut durchgeführt werden soll, so wird eine Ressourcenübergabe der unique_ptr (selected_shape -> clipboard_shape) durchgeführt. Für die Copy Funktion hingegen ist jedoch die Implementierung einer neuen Methode pro Shape notwendig. Diese Methode erstellt eine Kopie der eigenen Shape und retourniert den entstandenen unique_ptr. Nach dieser Operation wird die Selektion der Shape aufgehoben. Die Shape wird natürlich wieder dem shapes vector hinzugefügt. Dieser unique_ptr wird wiederum dem clipboard_ptr zugewiesen. Sollte die Funktion Paste ausgeführt werden, so wird die Shape aus dem „Clipboard“ dem shapes vector hinzugefügt.

2. Code

Main.h

```
#include "appl.h"

int main(int argc, char* argv[]) {
  draw_application app;
  app.run(argc, argv);
}
```

Shapes.h

```
#pragma once
#include <ml5/ml5.h>

struct shape : ml5::object {
  using context = ml5::paint_event::context_t;
  shape(wxPoint point, const wxPen& pen, const wxBrush& brush);

  void draw(context& con);
  auto contains(wxPoint pos) const noexcept -> bool;
  auto empty() const noexcept -> bool;
  void set_right_bottom(wxPoint point);
  void move(wxPoint offset);
  void set_pen(const wxPen &pen);
  void set_brush(const wxBrush &brush);
  virtual std::unique_ptr<shape> create_copy() = 0;

  wxRect aabb;
  wxPen pen;
  wxBrush brush;

private:
  virtual void draw_(context&) const = 0;

};

/* ============= line ============ */

struct line final : shape {
  using shape::shape;
  std::unique_ptr<shape> create_copy();
private:
  void draw_(context& cont) const override;
};

/* ============= ellipse ============ */

struct ellipse final : shape {
  using shape::shape;
  std::unique_ptr<shape> create_copy();
private:
  void draw_(context& cont) const override;
```

```
};

/* ============= rectangle ============ */

struct rectangle final : shape {
  using shape::shape;
  std::unique_ptr<shape> create_copy();
private:
  void draw_(context& cont) const override;
};
```

Shapes.cpp

```
#include "shapes.h"

/* ============== shape ============== */

shape::shape(wxPoint point, const wxPen & pen, const wxBrush & brush )
  : aabb{point, point},
    pen{pen},
    brush{brush}
  {}

void shape::draw(context & con) {
  con.SetPen(pen);
  con.SetBrush(brush);
  draw_(con);
}

auto shape::contains(wxPoint pos) const noexcept -> bool {
  return aabb.Contains(pos);
}

auto shape::empty() const noexcept -> bool {
  return !aabb.GetWidth() && !aabb.GetHeight();
}

void shape::set_right_bottom(wxPoint point) {
  aabb.SetRightBottom(point);
}

void shape::move(wxPoint offset) {
  aabb.Offset(offset);
}

void shape::set_pen(const wxPen& pen) {
  this->pen = pen;
}

void shape::set_brush(const wxBrush& brush) {
  this->brush = brush;
}
```

```
/* ================================== */
/* ================================== */


/* =============== line =============== */

void line::draw_(context& cont) const {
  cont.DrawLine(aabb.GetLeftTop(), aabb.GetBottomRight());
}

std::unique_ptr<shape> line::create_copy() {
  std::unique_ptr<shape> copied_shape = std::make_unique<line>(aabb.GetLeftTop(), pen,
brush);
  copied_shape->set_right_bottom(aabb.GetBottomRight());
  return copied_shape;
}


/* =============== ellipse =============== */

void ellipse::draw_(context& cont) const {
  cont.DrawEllipse(aabb);
}

std::unique_ptr<shape> ellipse::create_copy() {
  std::unique_ptr<shape> copied_shape = std::make_unique<ellipse>(aabb.GetLeftTop(), pen,
brush);
  copied_shape->set_right_bottom(aabb.GetBottomRight());
  return copied_shape;
}


/* =============== rectangle =============== */

void rectangle::draw_(context& cont) const {
  cont.DrawRectangle(aabb);
}

std::unique_ptr<shape> rectangle::create_copy() {
  std::unique_ptr<shape> copied_shape = std::make_unique<rectangle>(aabb.GetLeftTop(), pen,
brush);
  copied_shape->set_right_bottom(aabb.GetBottomRight());
  return copied_shape;
}
```

Appl.h

```
#pragma once
#include "shapes.h"
#include <ml5/ml5.h>

struct draw_application final : ml5::application {
  auto make_window() const->std::unique_ptr<ml5::window> override;
```

```
private:
  struct window final : ml5::window {
    window();
  private:

    // some types
    enum class shape_type { rectangle, ellipse, line  };
    shape_type next_shape{ shape_type::line };
    enum class operation { none, moving, dragging };
    operation current_operation{ operation::none };

    // basic functions
    void on_init() override;
    void on_menu(const ml5::menu_event& event);
    auto make_shape(wxPoint pos) const->std::unique_ptr<shape>;
    auto make_shape(wxPoint pos, shape_type c_type, wxBrush c_brush, wxPen c_pen) const-
>std::unique_ptr<shape>;
    void on_paint(const ml5::paint_event& event) override;
    auto get_topmost_shape(wxPoint pos)->std::unique_ptr<shape>;
    void on_mouse_left_down(const ml5::mouse_event& event) override;
    void on_mouse_left_up(const ml5::mouse_event& /*no name for param because not used*/)
override;
    void on_mouse_move(const ml5::mouse_event& event) override;
    void on_size(const ml5::size_event& event) override;

    /* selection functions */
    void init_left_bar();
    void show_selected_shape(int icon_id);
    void handle_select(const wxPoint& pos);
    void reset_selected_shape();

    /* edit functions */
    void on_copy_shape();
    void on_paste_shape();
    void on_select_shape();
    void on_cut_shape();

    wxPoint last_move_pos;
    wxBrush brush{ *wxGREEN_BRUSH };
    wxPen pen{ *wxBLACK_PEN };
    ml5::vector<std::unique_ptr<shape>> shapes;
    std::unique_ptr<shape> new_shape;

    /* left bar */
    std::unique_ptr<shape> left_bar{nullptr};
    ml5::vector<std::unique_ptr<shape>> left_bar_icons;

    /* select tool active */
    bool select_tool_active{false};
    std::unique_ptr<shape> selected_shape;
    ml5::vector<std::unique_ptr<shape>> selection_marker;
```

```
    /* cut, copy, paste */
    std::unique_ptr<shape> clipboard_shape{ nullptr };
  };
};
```

Appl.cpp

```
#include "appl.h"

//draw_application
auto draw_application::make_window() const->std::unique_ptr<ml5::window> {
  return std::make_unique<window>();
}

//window
draw_application::window::window() : ml5::window{"ML5.Draw"} {}


/* ============= left-nav-bar ============ */

void draw_application::window::init_left_bar() {

  /* position of left-nav-bar */
  wxPoint top_left{ 0 , 0 };
  wxPoint bottom_right{ 40 , get_height() };

  /* create left-nav-bar */
  left_bar = make_shape(top_left, shape_type::rectangle, *wxLIGHT_GREY_BRUSH,
*wxLIGHT_GREY_PEN);
  left_bar->set_right_bottom(bottom_right);

  /* Rectangle, Ellipse, Line, Arrow(select tool) */
  shape_type icon_type[] = {
                shape_type::rectangle , shape_type::ellipse , shape_type::line,   // rect,
ell, line
                shape_type::line, shape_type::line , shape_type::line       // arrow
               };

  /* icon position */
  wxPoint icon_top[]    = { { 10 , 10 } , { 10 , 40 } , { 30 , 70 } , { 30 , 100 } , { 30 ,
100 } , { 30 , 100 } };
  wxPoint icon_bottom[] = { { 30 , 30 } , { 30 , 60 } , { 10 , 90 } , { 10 , 120 } , { 25 ,
100 } , { 30 , 105 } };

  /* create icons */
  for (int i = 0; i < 6; i++) {
    next_shape = icon_type[i];
    new_shape = make_shape(icon_top[i],icon_type[i], *wxLIGHT_GREY_BRUSH,pen =
*wxBLACK_PEN);
    new_shape->set_right_bottom(icon_bottom[i]);
    left_bar_icons.add(std::move(new_shape));
  }
```

```
}

void draw_application::window::show_selected_shape(int icon_id) {

  /* reset current selection */
  for (int i = 0; i < 6; i++) {
    left_bar_icons[i]->set_pen(*wxBLACK_PEN);
    left_bar_icons[i]->set_brush(*wxLIGHT_GREY_BRUSH);
  }

  /* set new icon highlighting */
  left_bar_icons[icon_id]->set_pen(*wxYELLOW_PEN);
  left_bar_icons[icon_id]->set_brush(*wxYELLOW_BRUSH);
  if (icon_id == 3) {
    for (int i = 1; i <= 2; i++) {
      left_bar_icons[icon_id + i]->set_pen(*wxYELLOW_PEN);
      left_bar_icons[icon_id + i]->set_brush(*wxYELLOW_BRUSH);
    }
  }
  refresh();
}


/* ============= edit (copy, paste, select, cut) ============ */

void draw_application::window::on_copy_shape() {
  if (selected_shape) {
    clipboard_shape = selected_shape->create_copy();
    set_status_text("shape in clipboard, paste to insert");
    reset_selected_shape();
  } else {
    set_status_text("<< error >> no shape selected");
  }
}

void draw_application::window::on_paste_shape() {
  if (clipboard_shape) {
    shapes.add(std::move(clipboard_shape));
    set_status_text("shape was pasted");
  } else {
    set_status_text("<< error >> no shape in clipboard");
  }
}

void draw_application::window::on_select_shape() {
  select_tool_active = true;
  show_selected_shape(3);
}

void draw_application::window::on_cut_shape() {
  if (selected_shape) {
    clipboard_shape = std::move(selected_shape);
```

```
    set_status_text("shape in clipboard, paste to insert");
    reset_selected_shape();
  } else {
    set_status_text("<< error >> no shape selected");
  }
}

/* ============= make shapes ============ */

auto draw_application::window::make_shape(wxPoint pos) const -> std::unique_ptr<shape> {
  switch (next_shape) {
    case shape_type::line: return std::make_unique<line>(pos, pen, brush); break;
    case shape_type::ellipse: return std::make_unique<ellipse>(pos, pen, brush); break;
    case shape_type::rectangle: return std::make_unique<rectangle>(pos, pen, brush); break;
    default: throw std::logic_error{ "unknown shape type" };
  }
}

auto draw_application::window::make_shape(wxPoint pos, shape_type c_type ,wxBrush c_brush,
wxPen c_pen) const -> std::unique_ptr<shape> {
  switch (c_type) {
  case shape_type::line: return std::make_unique<line>(pos, c_pen, c_brush); break;
  case shape_type::ellipse: return std::make_unique<ellipse>(pos, c_pen, c_brush); break;
  case shape_type::rectangle: return std::make_unique<rectangle>(pos, c_pen, c_brush);
break;
  default: throw std::logic_error{ "unknown shape type" };
  }
}


/* ============= select function ============ */

void draw_application::window::reset_selected_shape() {
  if(selected_shape) shapes.add(std::move(selected_shape));
  selection_marker.clear();
  select_tool_active = false;
  show_selected_shape(int(next_shape));
  refresh();
}

void draw_application::window::handle_select(const wxPoint &pos) {

  if (selected_shape = get_topmost_shape(pos)) {

    // get corners from original shape
    wxPoint bottom_l = selected_shape->aabb.GetBottomLeft();
    wxPoint bottom_r = selected_shape->aabb.GetBottomRight();
    wxPoint top_l = selected_shape->aabb.GetTopLeft();
    wxPoint top_r = selected_shape->aabb.GetTopRight();

    std::unique_ptr<shape> new_marker{ nullptr };
    int offset = 4;
```

```
    // calculate selection marker coordinates with offsets
    wxPoint start_points[] =  { bottom_l , bottom_r , top_l , top_r };
    wxPoint end_points[] =    {
                    {bottom_l.x - offset, bottom_l.y + offset} ,
                    {bottom_r.x + offset, bottom_r.y + offset} ,
                    {top_l.x - offset, top_l.y - offset} ,
                    {top_r.x + offset, top_r.y - offset}
                  };

    // add selection markers to vector
    for (int i = 0; i < 4; i++) {
      new_marker = make_shape(start_points[i], shape_type::rectangle, *wxRED_BRUSH,
*wxRED_PEN);
      new_marker->set_right_bottom(end_points[i]);
      selection_marker.add(std::move(new_marker));
    }
    refresh();
  }
}


/* ============= event handlers ============ */

void draw_application::window::on_mouse_left_down(const ml5::mouse_event& event) {
  assert(!new_shape);
  assert(current_operation == operation::none);

  const auto pos{ event.get_position() };

  /* if select tool is active */
  if (select_tool_active) {
    handle_select(pos);
    return;
  }

  /* set operation move/dragging */
  if (new_shape = get_topmost_shape(pos)) {
    last_move_pos = pos;
    current_operation = operation::moving;
  } else {
    new_shape = make_shape(event.get_position());
    current_operation = operation::dragging;
  }
  refresh();
}

void draw_application::window::on_mouse_left_up(const ml5::mouse_event& event) {
  if (select_tool_active) return;
  if (!new_shape) return;
  if (new_shape->empty()) new_shape.reset();
  else shapes.add(std::move(new_shape));
```

```
  current_operation = operation::none;
  refresh();
}

void draw_application::window::on_mouse_move(const ml5::mouse_event & event) {
  wxPoint curr_pos = event.get_position();

  if (!new_shape) return;

  // limit draw area
  if (curr_pos.x <= 40      || curr_pos.x >= get_size().x ||
    curr_pos.y >= get_size().y  || curr_pos.y <= 0)       {
    return;
  }

  const auto pos{ event.get_position() };
  switch (current_operation) {
    case operation::moving: {
      const auto offset{ pos - last_move_pos };
      last_move_pos = pos;
      new_shape->move(offset);
    } break;
    case operation::dragging: {
      new_shape->set_right_bottom(event.get_position());
    } break;
    default: throw std::logic_error{ "unknown state" };
  }
  refresh();
}

void draw_application::window::on_paint(const ml5::paint_event &event) {
  auto& con{ event.get_context() };
  /* draw shapes on board */
  for (const auto& s : shapes) s->draw(con);
  /* draw current shape */
  if (new_shape) new_shape->draw(con);
  /* draw left-nav-bar */
  if (left_bar) left_bar->draw(con);
  for (const auto &s : left_bar_icons) s->draw(con);
  /* draw selection marker */
  for (const auto& s : selection_marker) s->draw(con);
  /* draw selected shape */
  if (selected_shape) selected_shape->draw(con);
}

void draw_application::window::on_size(const ml5::size_event& event) {
  /* set new right bottom point of left-nav-bar (resize) */
  wxPoint bottom_right{ 40 , get_height() };
  left_bar->set_right_bottom(bottom_right);
}
```

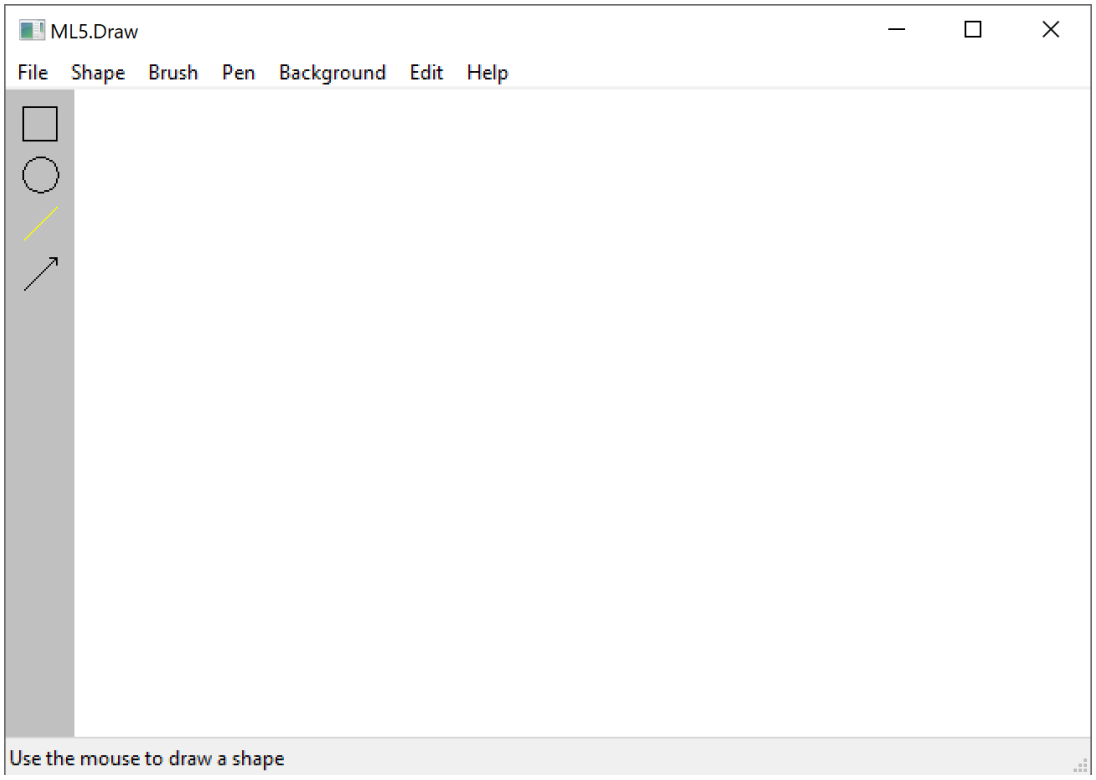
```
void draw_application::window::on_menu(const ml5::menu_event& event) {

  const auto item{ event.get_title_and_item() };
  if (item == "Shape/Rectangle") {
    next_shape = shape_type::rectangle;
    show_selected_shape(0);
  }
  else if (item == "Shape/Ellipse") {
    next_shape = shape_type::ellipse;
    show_selected_shape(1);
  }
  else if (item == "Shape/Line") {
    next_shape = shape_type::line;
    show_selected_shape(2);
  }

  if (item == "Brush/Blue")       brush = *wxBLUE_BRUSH;
  else if (item == "Brush/Green")     brush = *wxGREEN_BRUSH;
  else if (item == "Brush/Red")     brush = *wxRED_BRUSH;
  else if (item == "Brush/White")     brush = *wxWHITE_BRUSH;
  else if (item == "Brush/Light Gray")  brush = *wxLIGHT_GREY_BRUSH;
  else if (item == "Brush/Black")     brush = *wxBLACK_BRUSH;

  if (item == "Pen/Blue")     pen = *wxBLUE_PEN;
  else if (item == "Pen/Green") pen = *wxGREEN_PEN;
  else if (item == "Pen/Red")   pen = *wxRED_PEN;
  else if (item == "Pen/White") pen = *wxWHITE_PEN;
  else if (item == "Pen/Black") pen = *wxBLACK_PEN;

  if (item == "Background/Blue")        set_prop_background_brush(*wxBLUE_BRUSH);
  else if (item == "Background/Green")    set_prop_background_brush(*wxGREEN_BRUSH);
  else if (item == "Background/Red")      set_prop_background_brush(*wxRED_BRUSH);
  else if (item == "Background/White")    set_prop_background_brush(*wxWHITE_BRUSH);
  else if (item == "Background/Light Gray") set_prop_background_brush(*wxLIGHT_GREY_BRUSH);
  else if (item == "Background/Black")    set_prop_background_brush(*wxBLACK_BRUSH);

  if (item == "Edit/Copy")    on_copy_shape();
  else if (item == "Edit/Paste")  on_paste_shape();
  else if (item == "Edit/Select") on_select_shape();
  else if (item == "Edit/Cut")  on_cut_shape();

  refresh();
}

void draw_application::window::on_init() {
  add_menu("&Shape", {
    { "&Line"   , "<< next shape: line >>"    },
    { "&Ellipse"  , "<< next shape: ellipse >>" },
    { "&Rectangle"  , "<< next shape: rectangle >>" }
    });
```

```
  add_menu("&Brush", {
    { "&Blue"   , "<< next brush: blue >>"    },
    { "&Green"    , "<< next brush: green >>"   },
    { "&Red"    , "<< next brush: red >>"   },
    { "&White"    , "<< next brush: white >>"   },
    { "&Light Gray" , "<< next brush: light gray >>"},
    { "Blac&k"    , "<< next brush: black >>"   }
    });

  add_menu("&Pen", {
    { "&Blue"   , "<< next pen: blue >>"    },
    { "&Green"    , "<< next pen: green >>"   },
    { "&Red"    , "<< next pen: red >>"     },
    { "&White"    , "<< next pen: white >>"   },
    { "Blac&k"    , "<< next pen: black >>"   }
    });

  add_menu("Back&ground", {
    { "&Blue"   , "<< next background blue >>"      },
    { "&Green"    , "<< next background green >>"     },
    { "&Red"    , "<< next background red >>"     },
    { "&White"    , "<< next background white >>"     },
    { "&Light Gray" , "<< next background light gray >>"  },
    { "Blac&k"    , "<< next background black >>"     }
    });

  add_menu("&Edit", {
    { "C&ut"  , "<< cut selected shape >>"      },
    { "&Copy" , "<< copy selected shape >>"     },
    { "&Paste"  , "<< paste shape from clipboard >>"  },
    { "&Select" , "<< select shape >>"          }
    });

  set_status_text("Use the mouse to draw a shape");
  init_left_bar();      // create left-nav-bar
  show_selected_shape(2);   // highlight initial shape (rectangle)
}
```

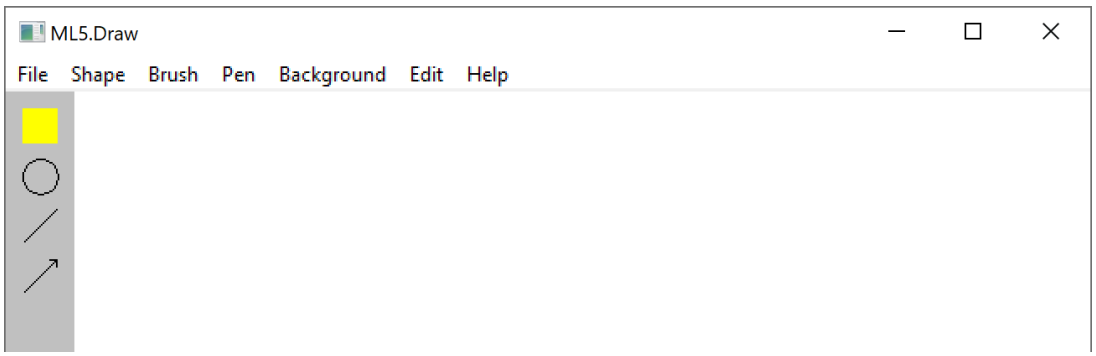
```
/* ============= helper function ============ */

auto draw_application::window::get_topmost_shape(wxPoint pos) -> std::unique_ptr<shape> {

  // reset selection
  if (selected_shape) reset_selected_shape();

  // select shape with left bar
  for (int i = 0; i < 6; i++) {
    if (left_bar_icons[i]->contains(pos)) {
      next_shape = shape_type(i); // select correct enum by indec
      refresh();
      show_selected_shape(i);
      return {};
    }
  }

  std::unique_ptr<shape>* tmp{ nullptr };
  for (auto& s : shapes) {
    if (s->contains(pos)) {
      tmp = &s;
    }
  }

  if (!tmp) return {};
  std::unique_ptr<shape> result{ std::move(*tmp) };
  shapes.remove(*tmp);
  return result;
}
```

3. Testfälle

Benutzeroberfläche des Grafikeditors
Der folgende Screenshot zeigt die Benutzeroberfläche mit der Implementierung einer Sidebar wie auf der Angabe. Die aktuell ausgewählte Shape wird in gelber Farbe hervorgehoben. Der Benutzer kann sowohl mit Hilfe der Sidebar als auch mit dem Hauptmenü die Shape ändern.

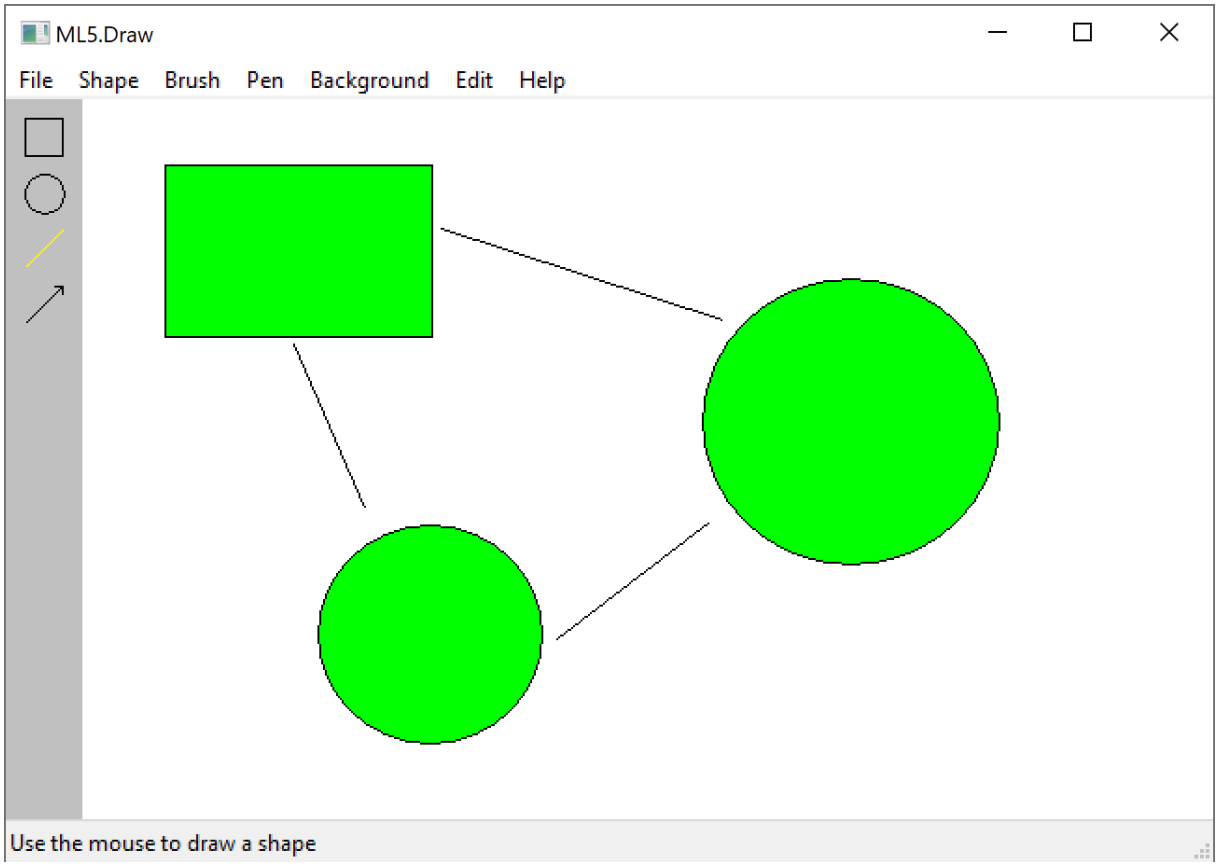


Nach der Auswahl einer anderen Shape, sieht die Sidebar wie folgt aus:



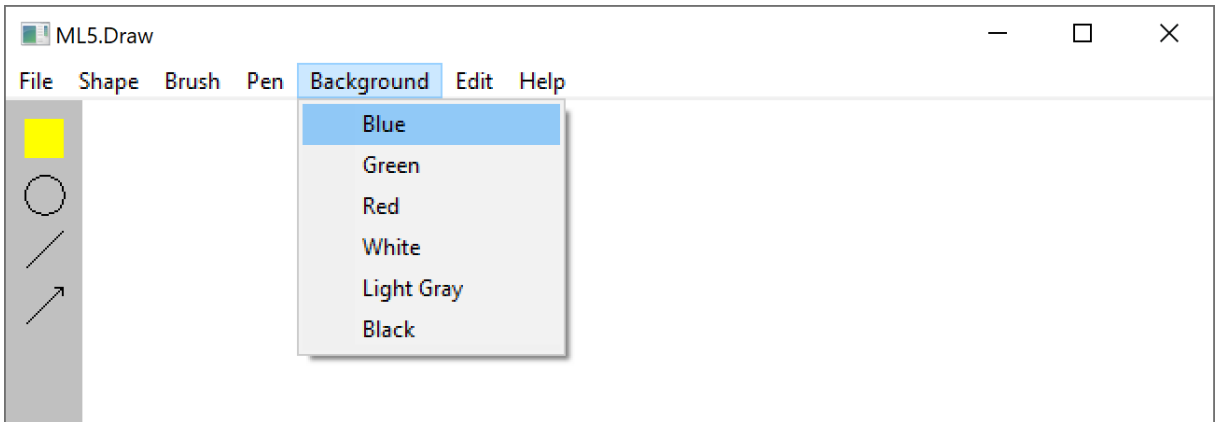
Zeichnen von Shapes

Der nachstehende Screenshot soll das Zeichnen mit verschiedenen Shapes illustrieren. Eine Shape wird immer dann gezeichnet wenn ein valides Tool ausgewählt wurde und beim Linksklick die Maus bewegt wird.

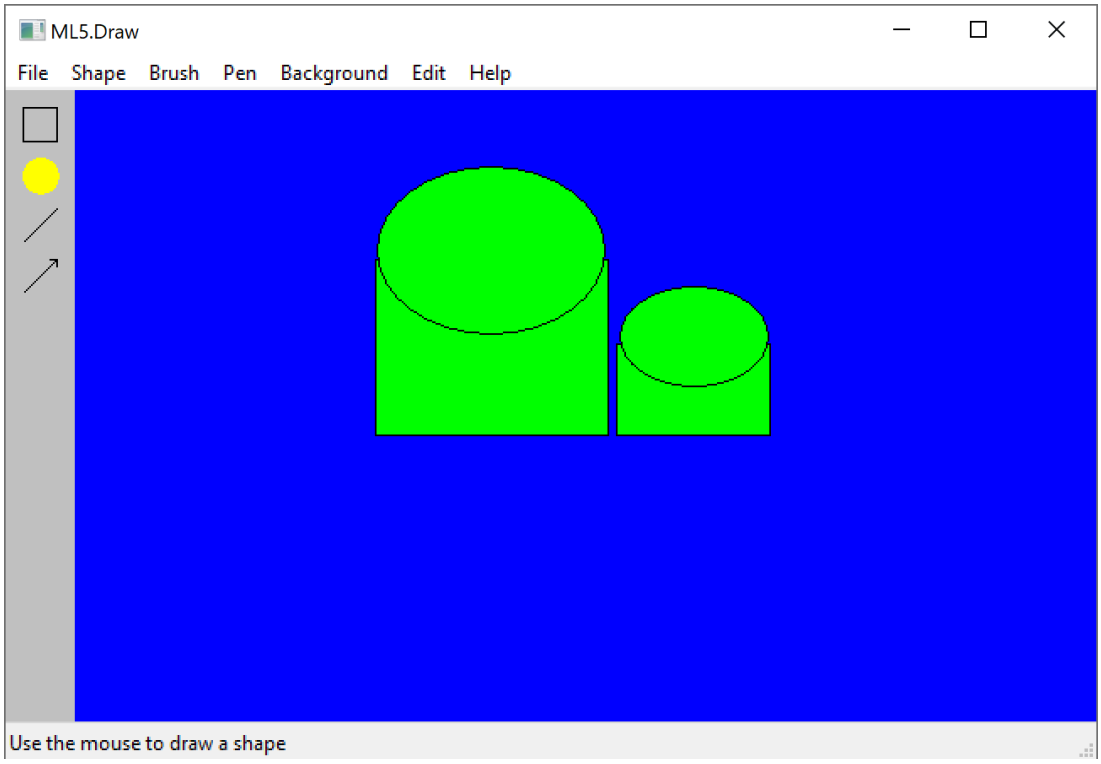


Ändern von Farben

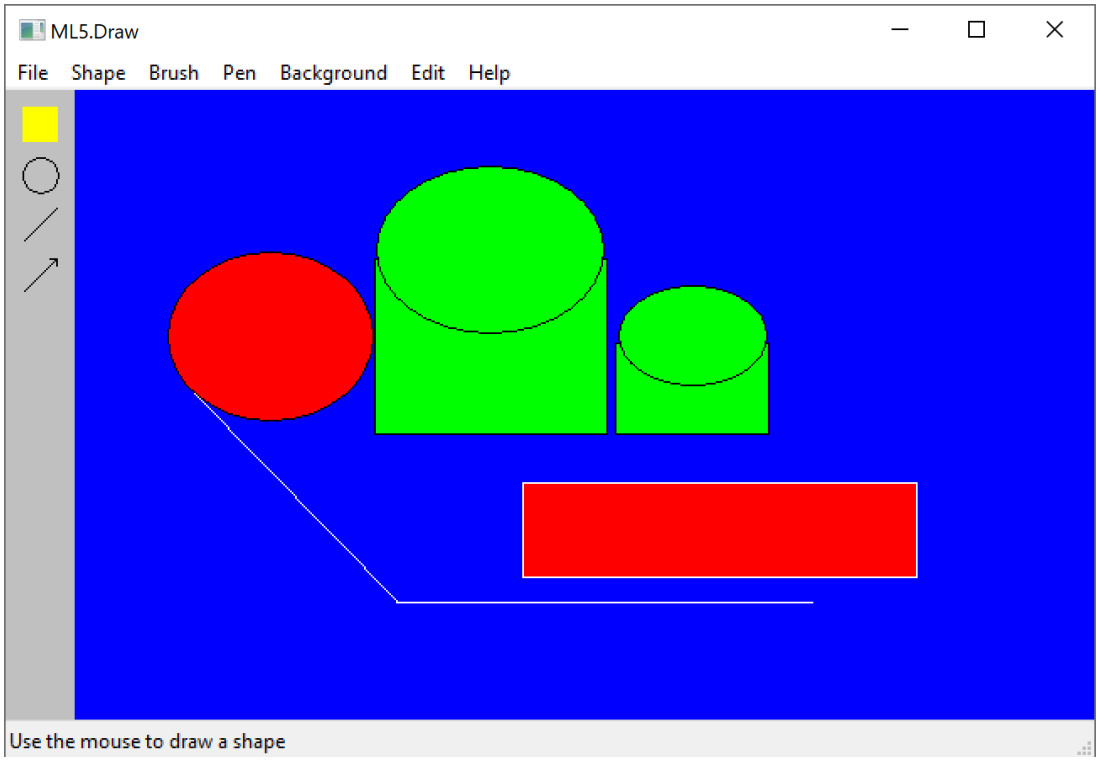
Der folgende Screenshot zeigt, wie eine Farbe geändert werden kann. Hier wurde die Hintergrundfarbe geändert. Die Menüs für das Ändern der Füllung bzw. des Stiftes sehen ident aus.



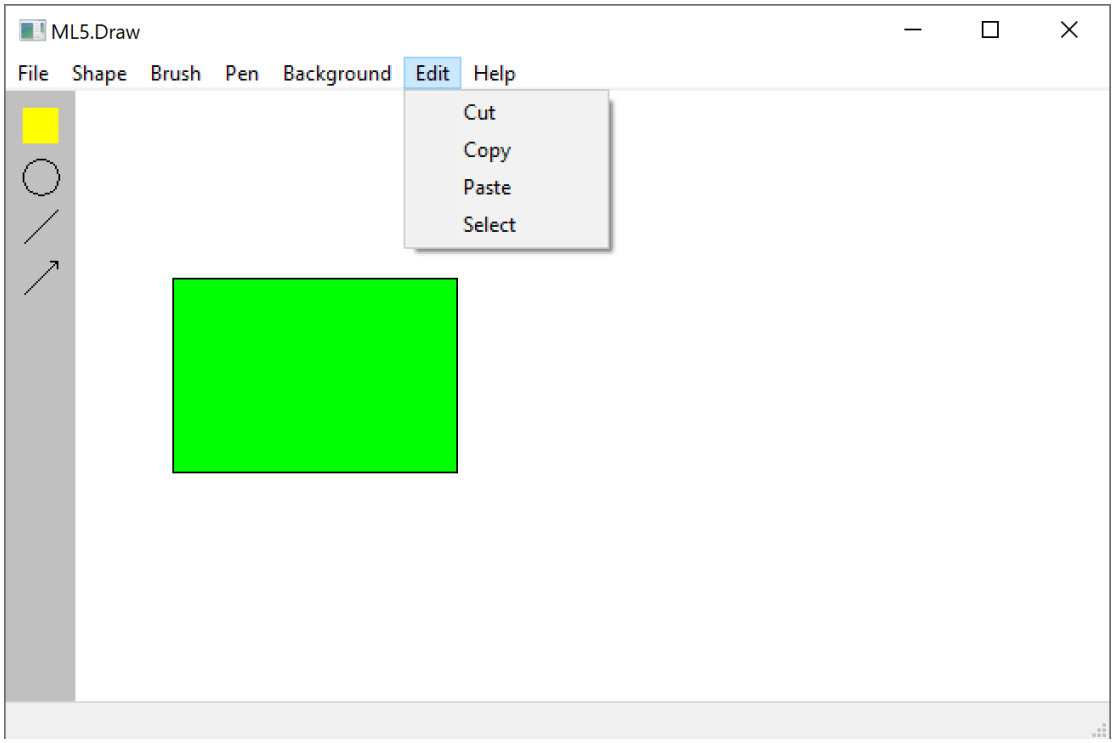
Der Nachfolgende Screenshot zeigt ein Gemälde vom Künstler Michael N. mit geänderter Hintergrundfarbe.



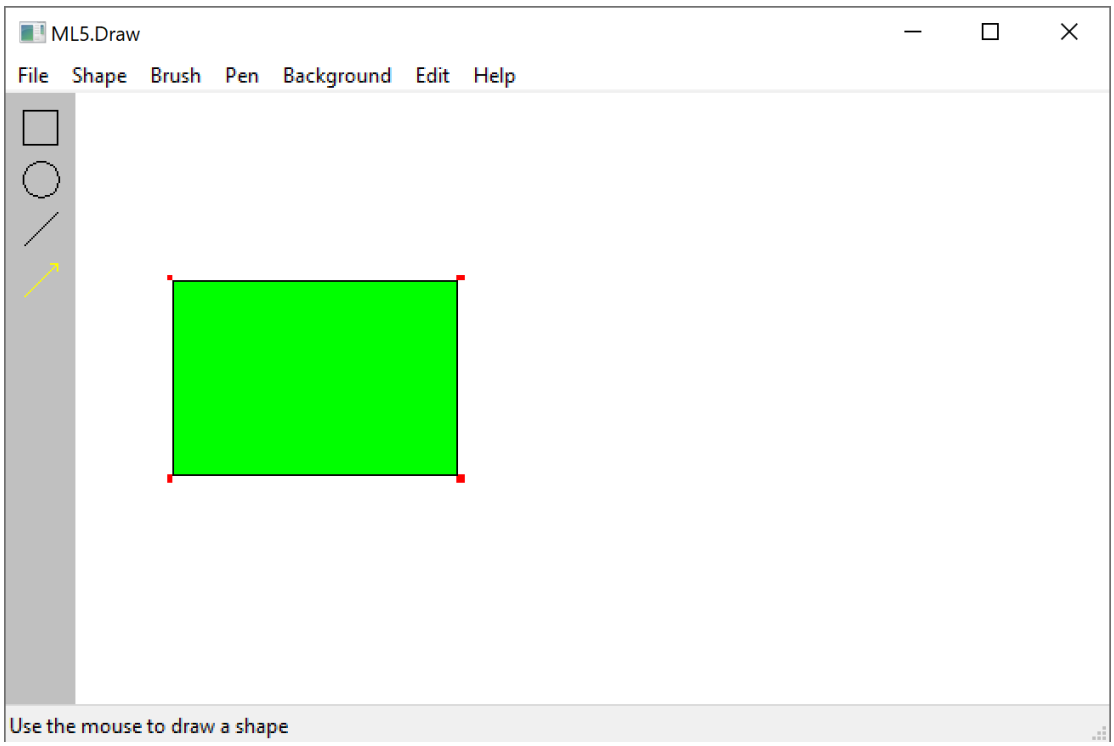
Hier mit geänderter Füllung:



Selektieren von Shapes
Der nachfolgende Screenshot zeigt das Menü, indem das Tool „Selektieren einer Shape“ zur Verfügung steht.

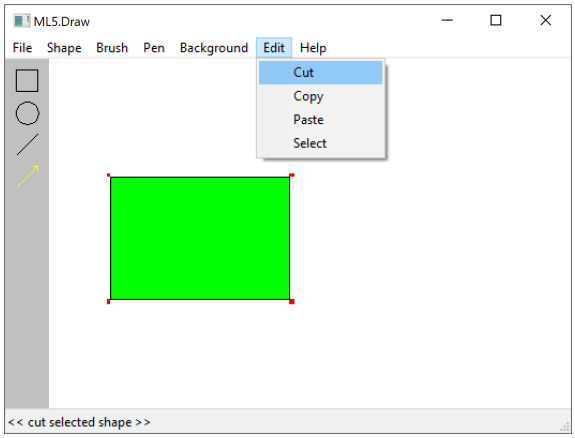


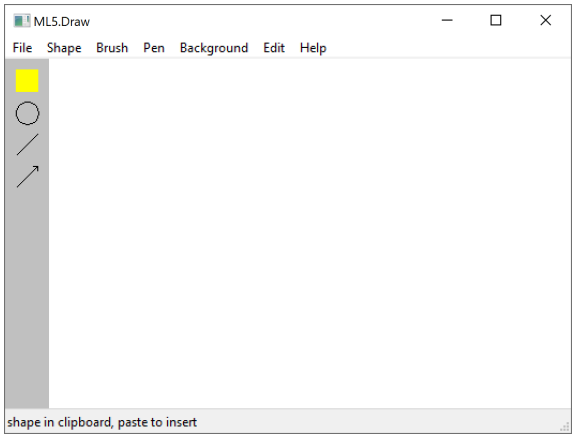
Wird das Tool ausgewählt und klick der Benutzer im Anschluss auf eine Shape, so werden die Eckpunkte dieser Shape mit roten Markierungen hervorgehoben. (Im Code sind die Markierungen gleich Groß ;) => wahrscheinlich „anti aliasing“ ausgeschaltet)

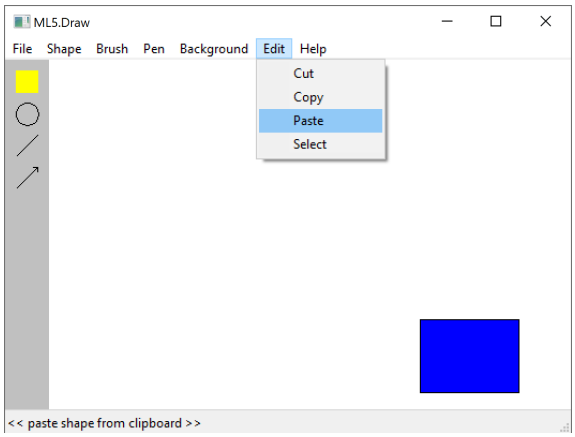


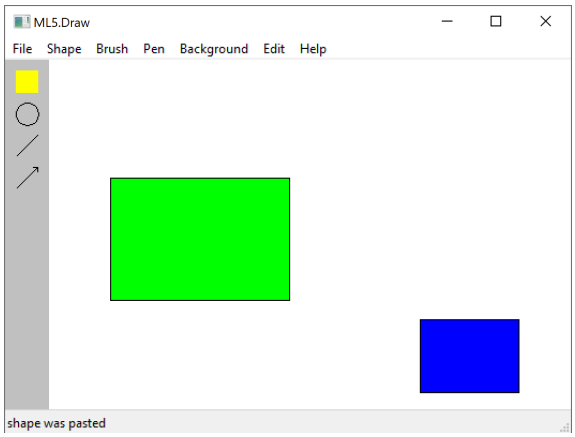
Operationen Cut & Paste der ausgewählten Shape

Die nachfolgenden Screenshots zeigen die Operationen Cut und Paste die auf die Ausgewählte Shape angewandt werden können. Nach der Auswahl von Cut wird Selektion zurückgesetzt und es kann mit der ursprünglichen Shape weitergezeichnet werden.









Operationen Copy & Paste der ausgewählten Shape

Die nachfolgenden Screenshots zeigen die Operationen Copy und Paste der ausgewählten Shape. Nach dem einfügend er kopierten Shape liegt diese klarerweise über der originalen Shape auf dem Screenshot rechts im Eck wurde die eingefügte Shape schon etwas verschoben, damit die Operation eindeutiger wird.