Inhaltsverzeichnis

[1. Lösungsidee 2](#_Toc30933684)

[2. Code 3](#_Toc30933685)

[3. Testfälle 15](#_Toc30933686)

# 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.

# 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;  } |

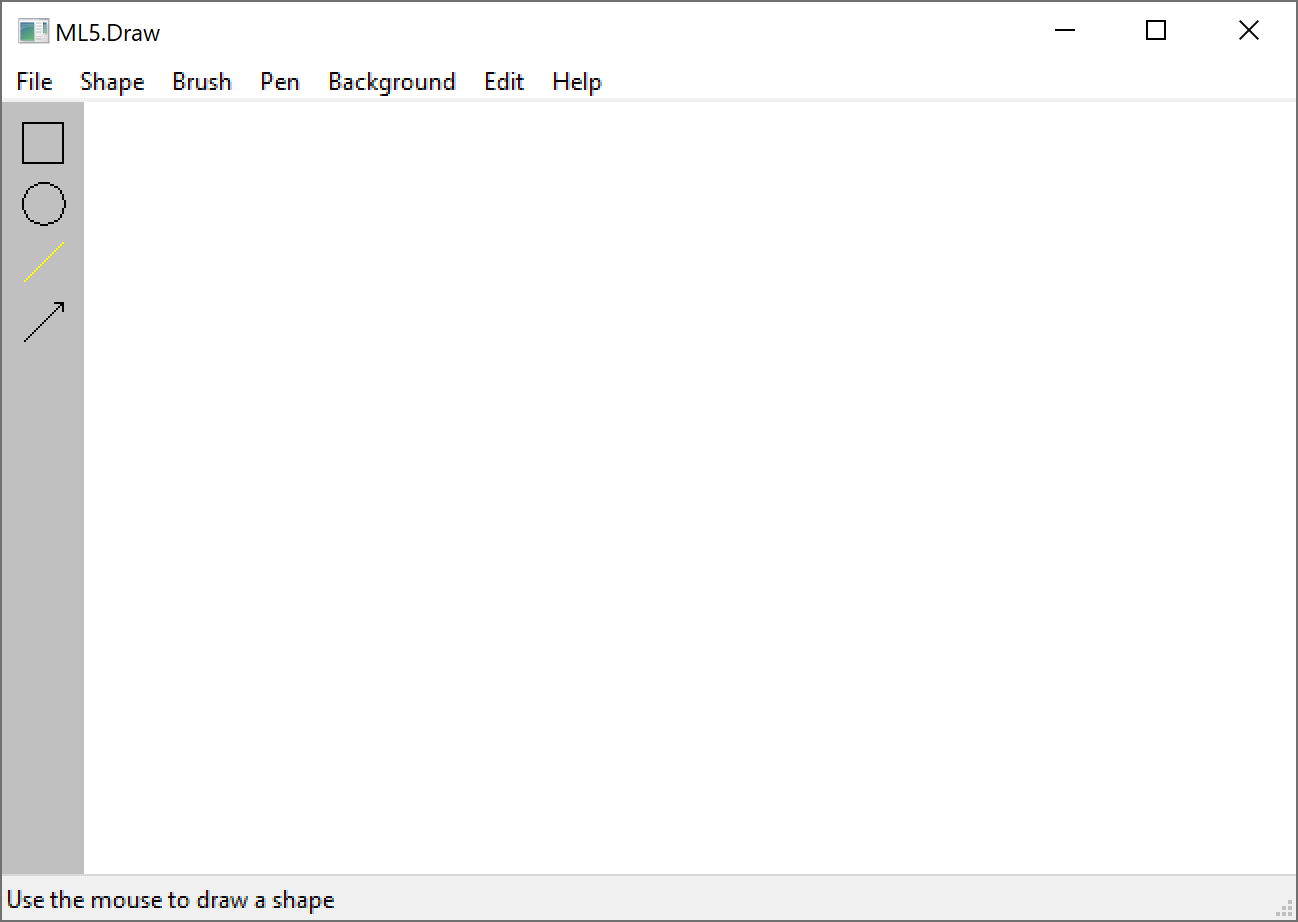
|  |
| --- |
| **Appl.h** |
| #pragma once  #include "shapes.h"  #include <ml5/ml5.h>  struct draw\_application final : ml5::application {  auto make\_window() const->std::unique\_ptr<ml5::window> override;  private:  struct window final : ml5::window {  window();  private:  // some types  enum class 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;  } |

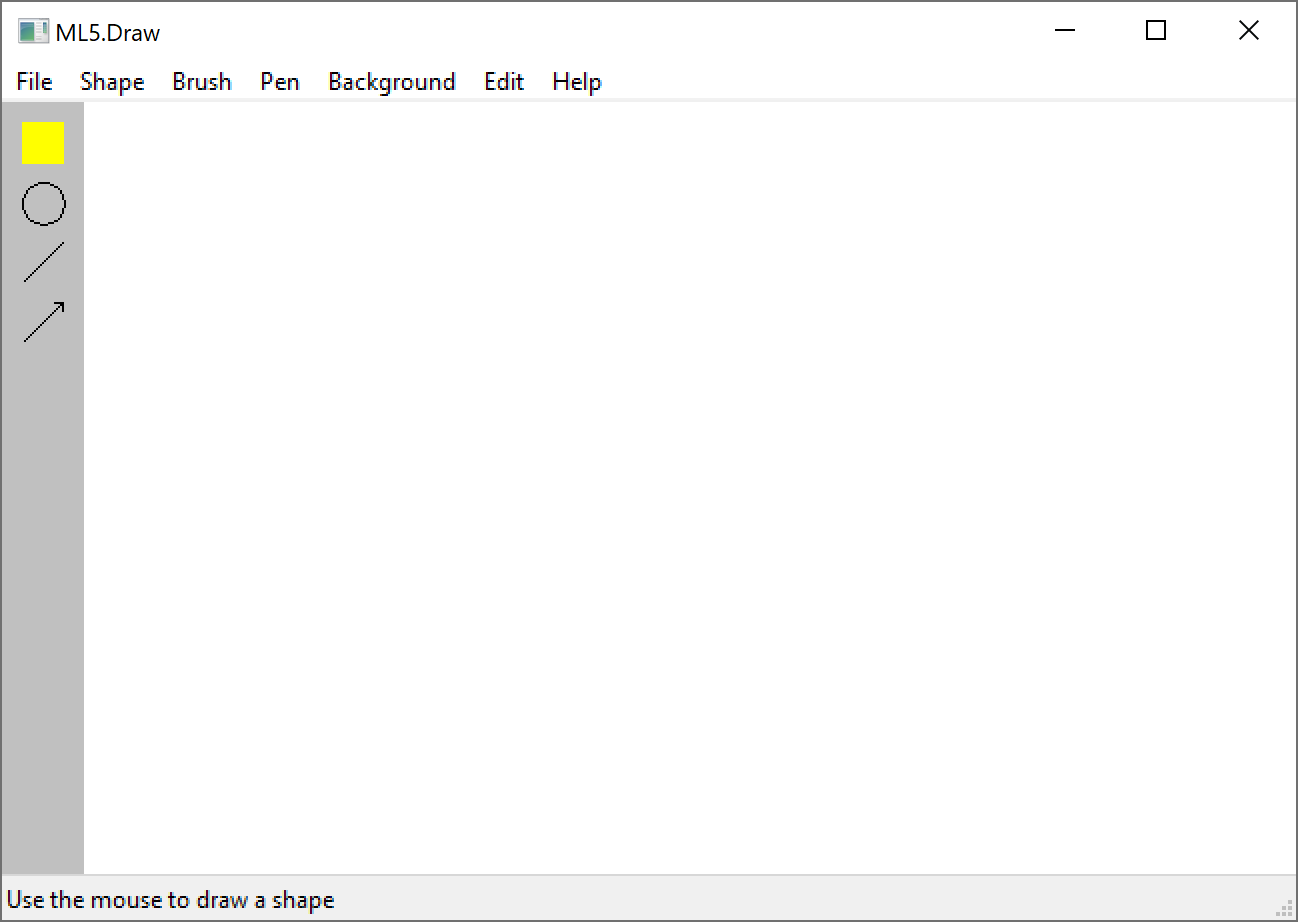
# Testfälle

**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.

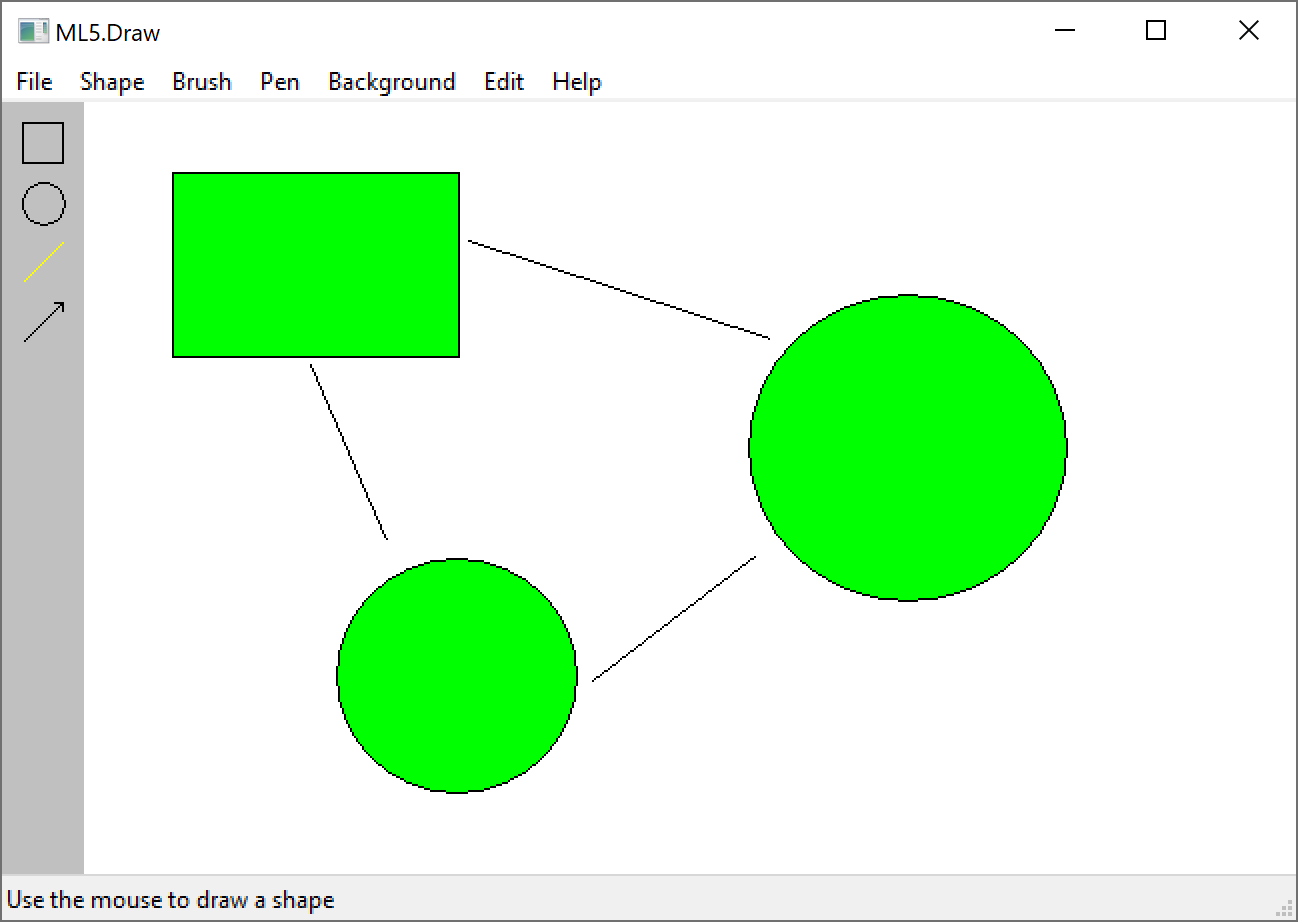


Nach der Auswahl einer anderen Shape, sieht die Sidebar wie folgt aus:



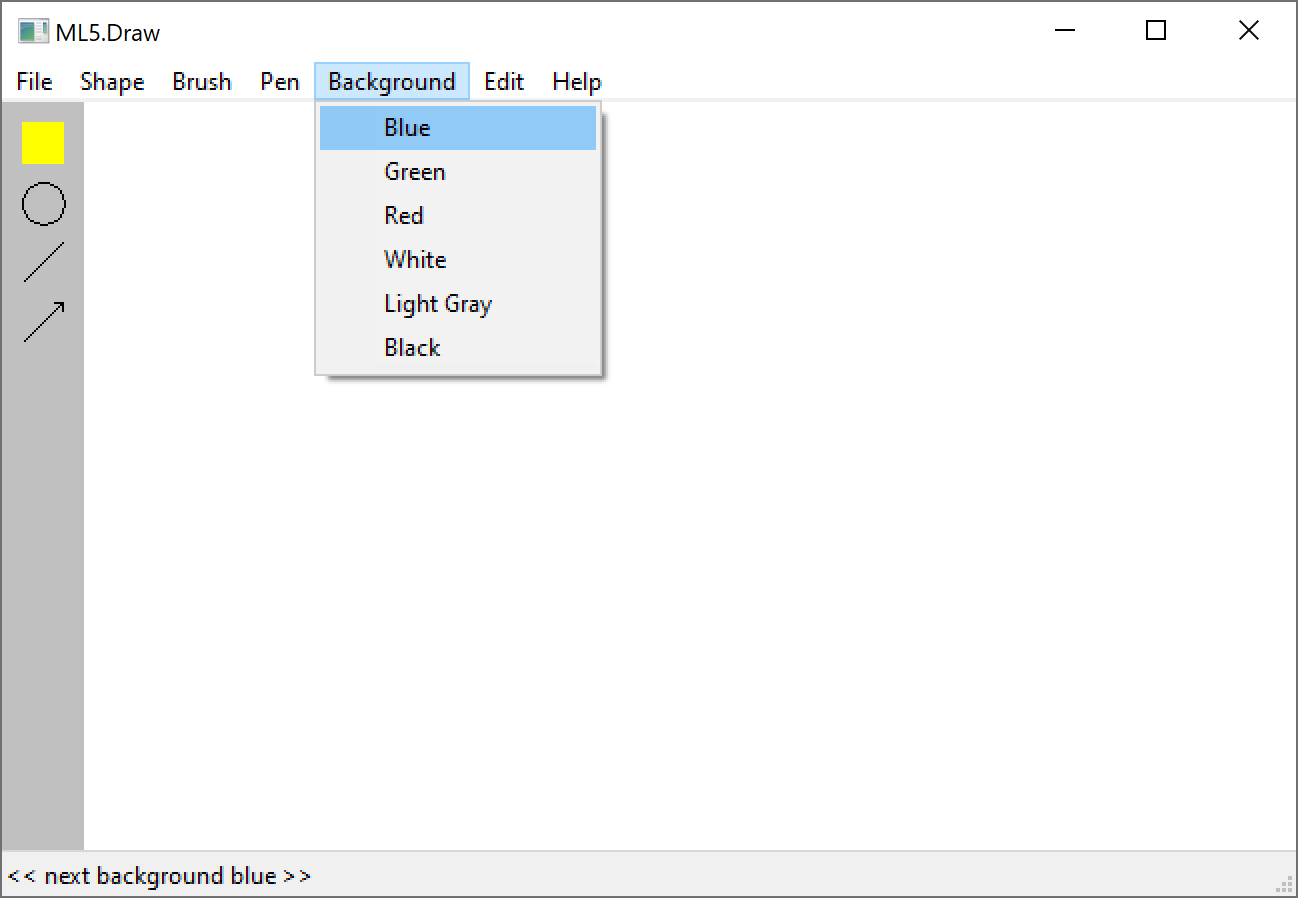
**Zeichnen von Shapes**

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.

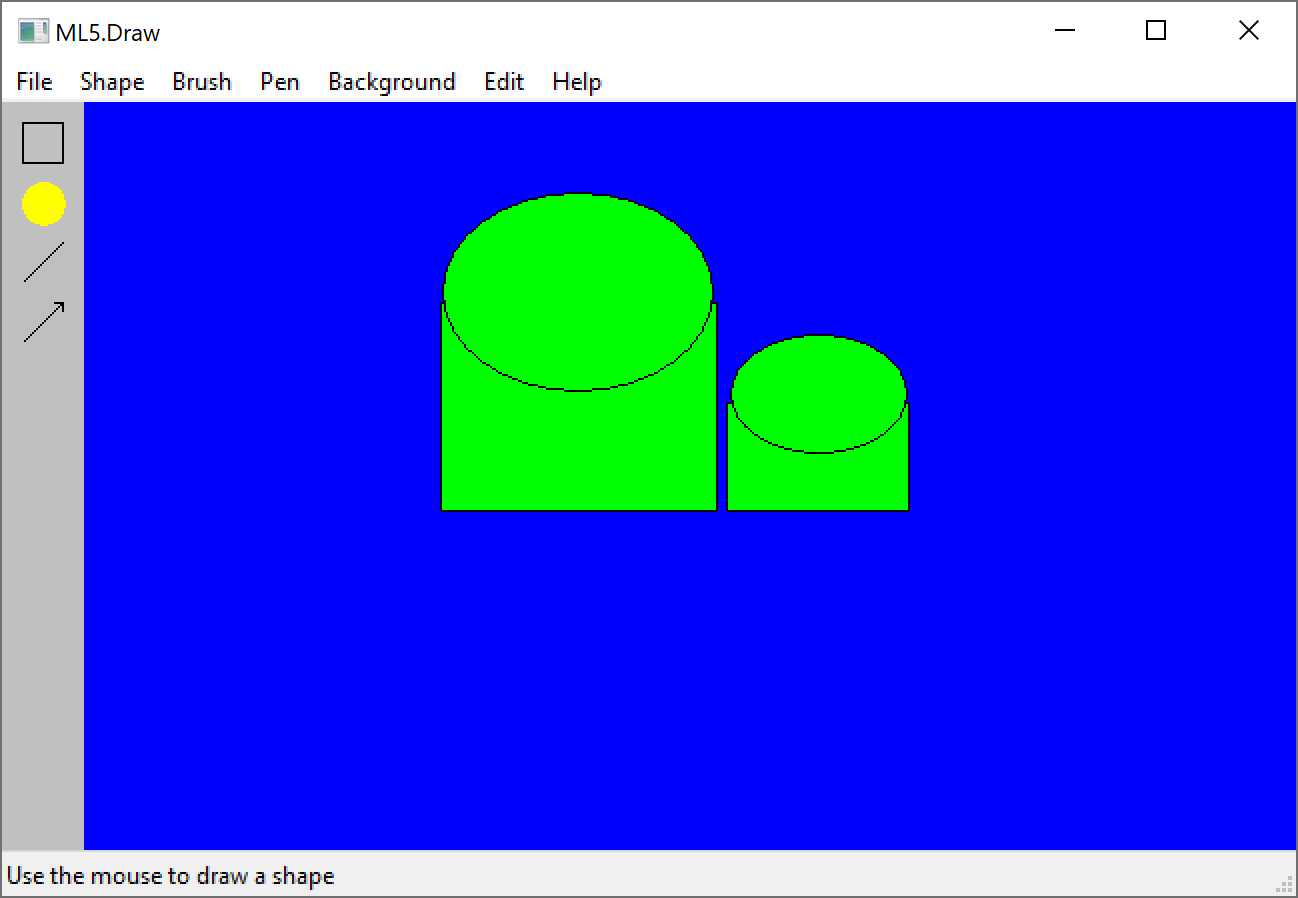


**Ändern von Farben**

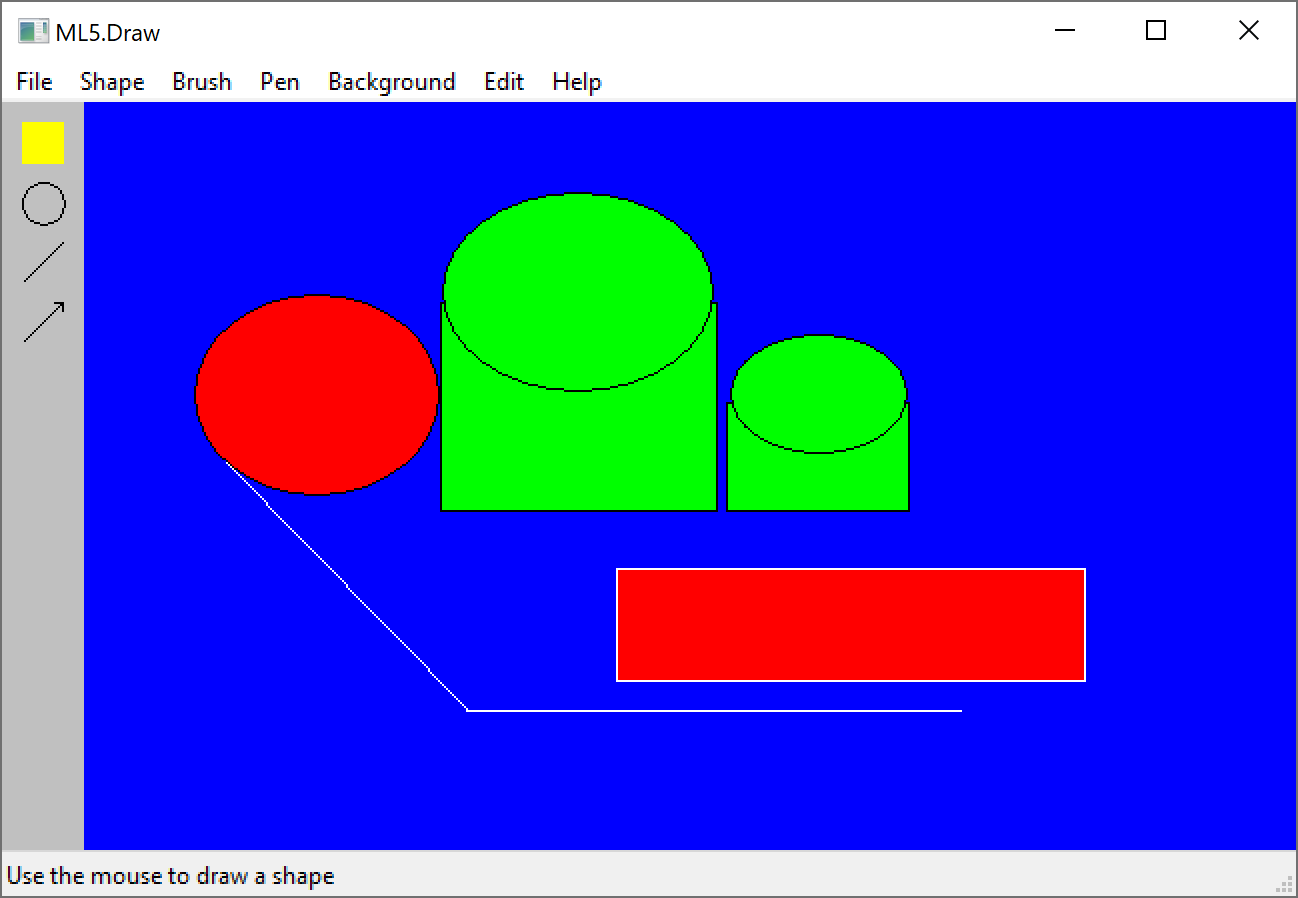
Der folgende Screenshot zeigt, wie eine Farbe geändert werden kann. Hier wurde die Hintergrundfarbe geändert. Die Menüs für das Ändern der Füllung bzw. des Stiftes sehen ident aus.



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

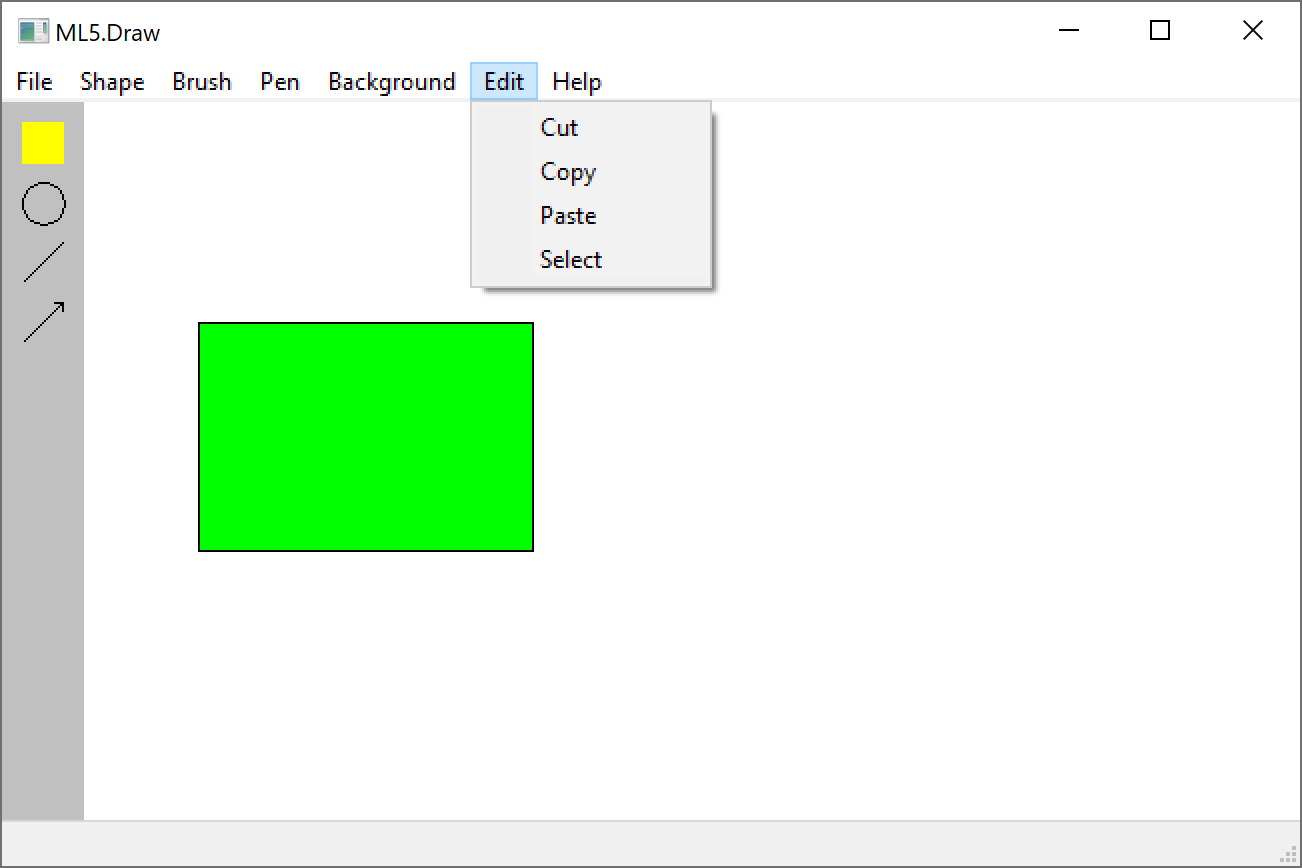


Hier mit geänderter Füllung:

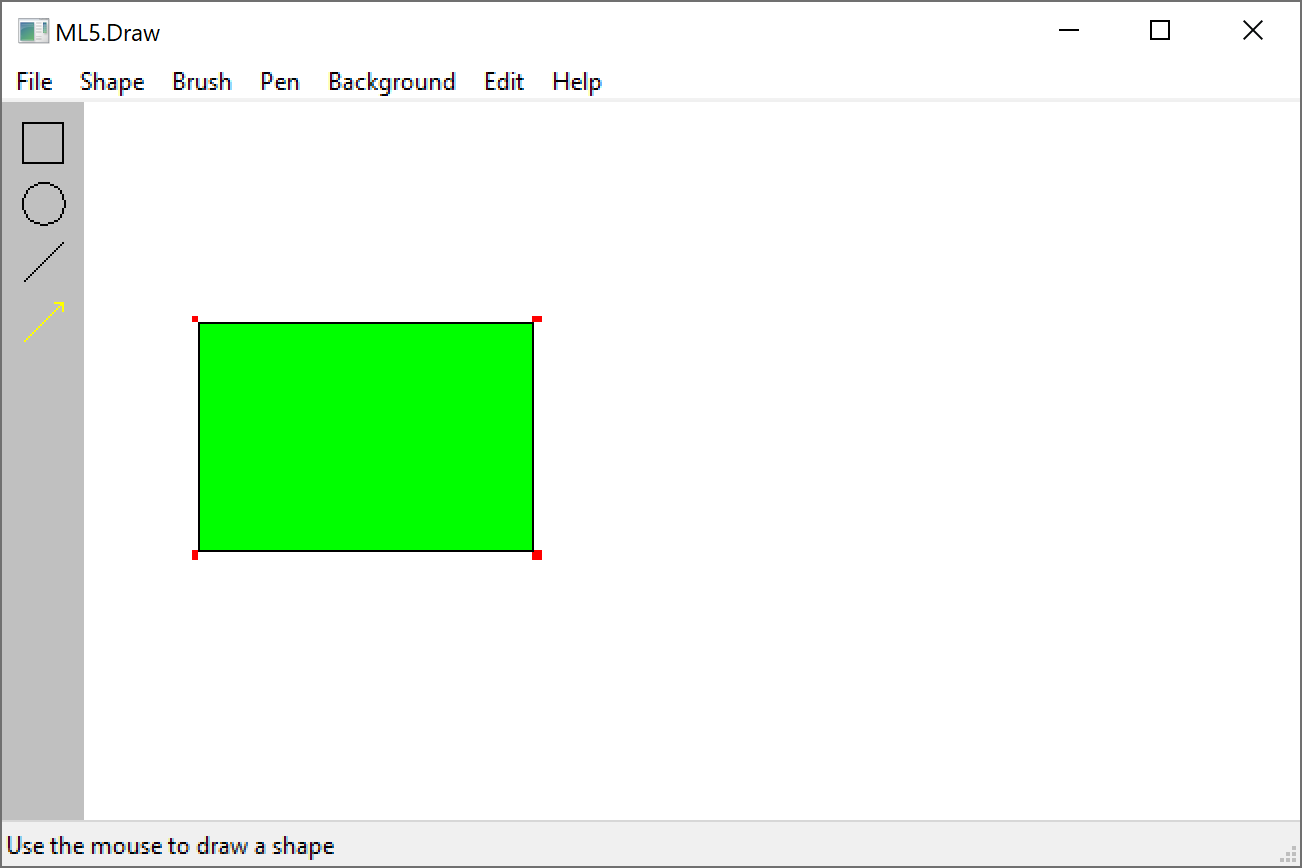


**Selektieren von Shapes**

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.

|  |  |
| --- | --- |
|  |  |
|  |  |

