MCIGraph Grafikbibliothek

Installation und Verwendung

Sebastian Stabinger, Thomas Hausberger

SS2021

Installation

Installation von MCIGraph — Visual Studio, Code::Blocks

Visual Studio

- Auf Sakai finden Sie die Datei `MCIGraphTemplate_visualstudio.zip`
- Entpacken Sie diese
- Öffnen Sie die Datei `MCIGraphTemplate.sln` in Visual Studio
- Nach Ausführung sollten Sie ein Beispielfenster sehen

Code::Blocks

- Auf Sakai finden Sie die Datei `MCIGraphTemplate_codeblocks.zip`
- Entpacken Sie diese
- Öffnen Sie die Datei `MCIGraph.cbp` als Projekt in Code::Blocks
- Nach Ausführung sollten Sie ein Beispielfenster sehen

Linux

- Installieren sie die SDL2 Developer Library mit dem Paketmanager ihrer Distribution
- Laden sie die Datei `mcigraph.hpp` von Sakai und speichern sie diese im selben Verzeichnis wie ihre Quellcodedatei.
- Compilieren Sie ihr Programm mittels:

```
g++ -std=c++11 main.cpp $(sdl2-config --cflags --libs) -lpthread
```

Installation von MCIGraph — MacOS X

- Installieren Sie Homebrew indem sie folgendes am Terminal eingeben (alles in einer Zeile und mit Leerzeichen zwischen `-fsSL` und `https...`):

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

- Anschließend installieren Sie SDL2 mit dem Befehl

```
brew install sdl2
```

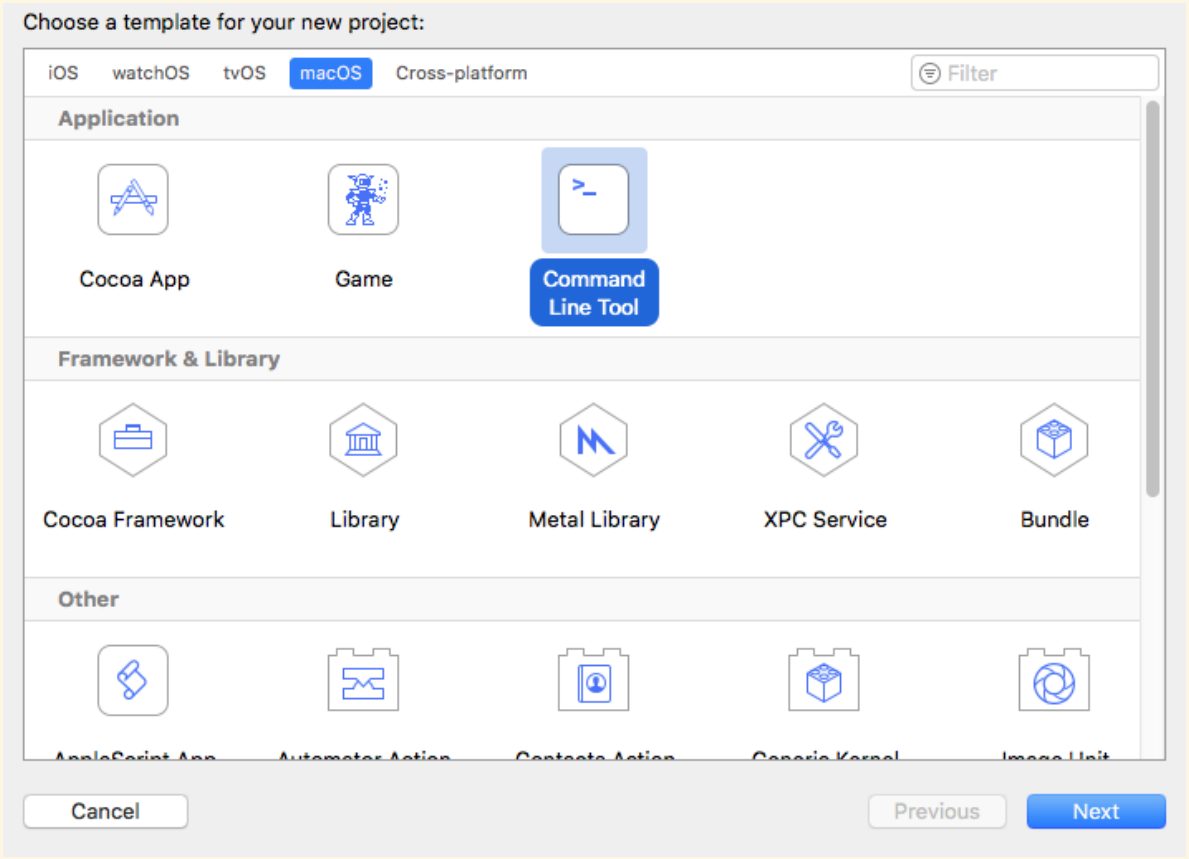
- Laden sie die Datei `mcigraph.hpp` von Sakai und speichern Sie diese im selben Verzeichnis wie ihre Quellcodedatei.
- Falls ihre Quellcodedatei `test.cpp` heißt, compilieren Sie ihr Programm mittels

```
g++ -std=c++11 test.cpp $(sdl2-config --cflags --libs) -lpthread
```

- Sie können dann ihr Programm mittels `./a.out` starten

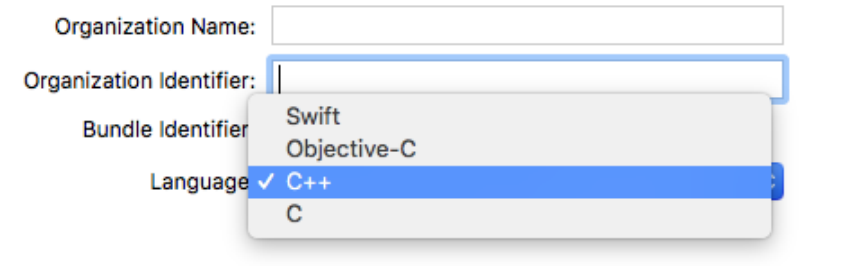
Installation von MCIGraph — XCode auf MacOS X

- Installieren Sie `SDL2` mit Homebrew (siehe vorherige Folie)
- Öffnen Sie XCode und erzeugen Sie ein neues Projekt
- Wählen Sie als Template eine macOS Kommandozeilenanwendung aus:

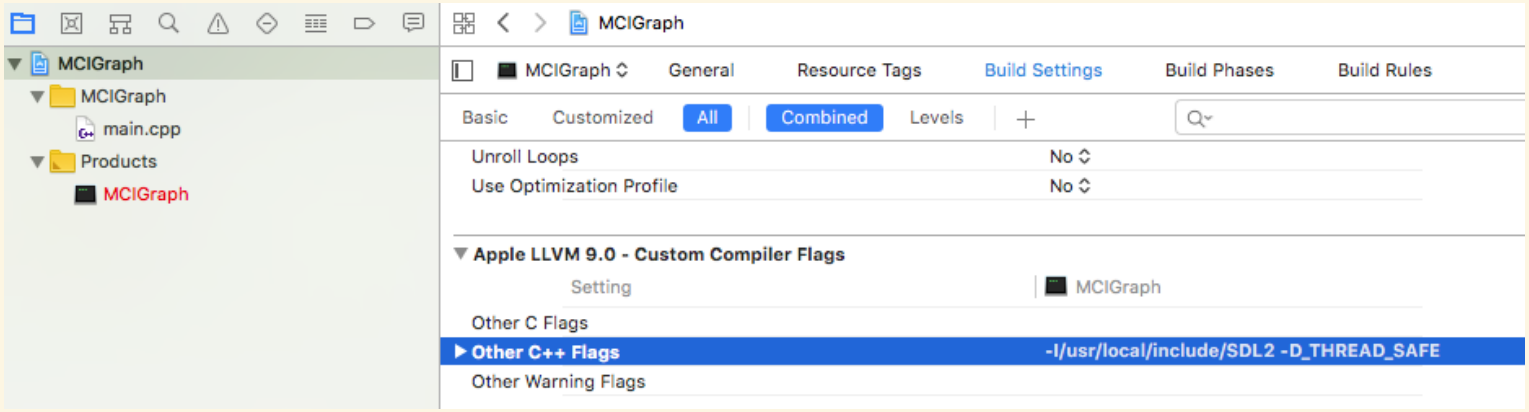


Installation von MCIGraph — XCode auf MacOS X

- Stellen Sie als Sprache C++ ein:

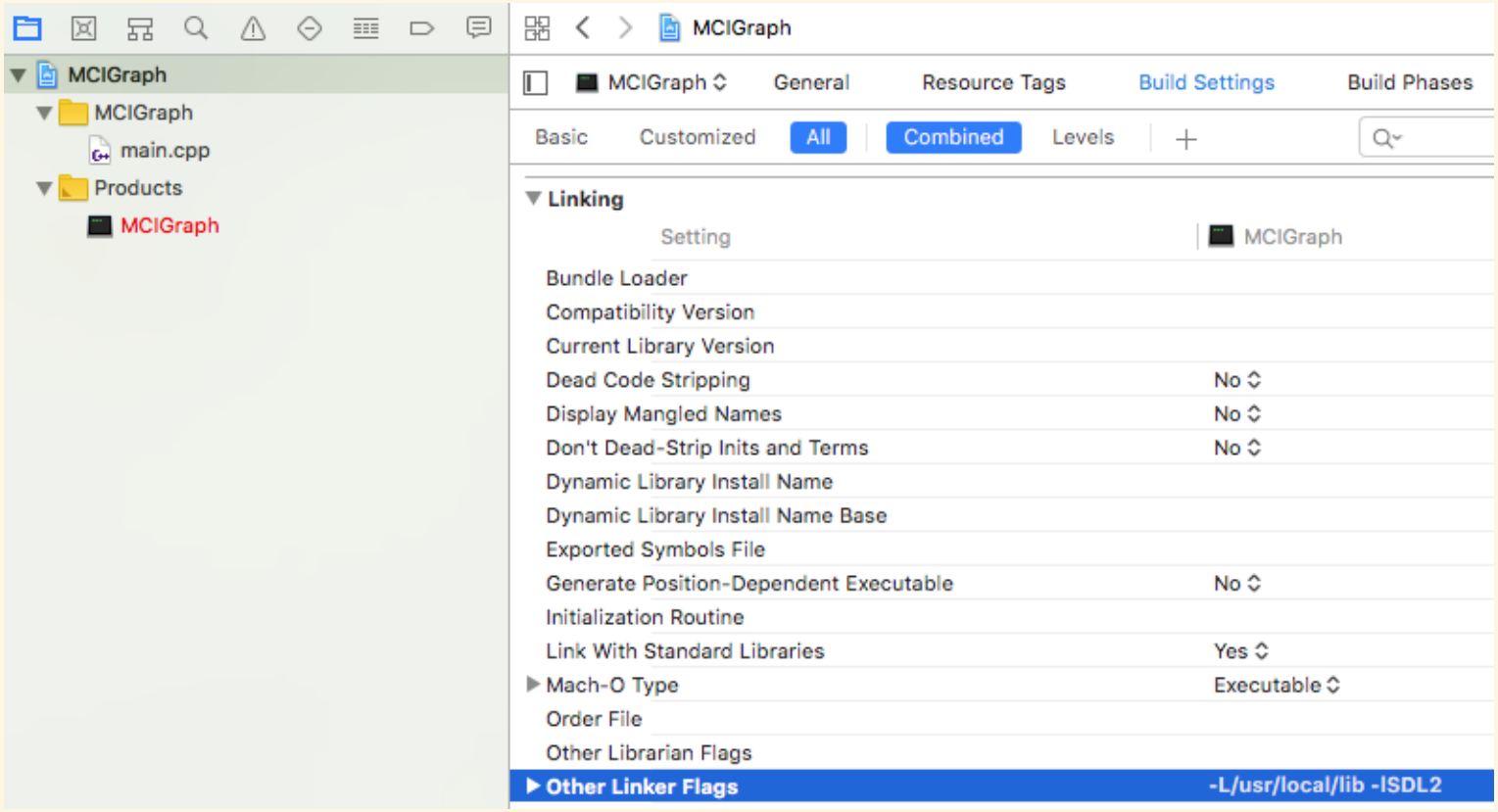


- Führen Sie auf dem Terminal den Befehl `sdl2-config --cflags` aus und kopieren Sie das Ergebnis in die C++ Flags der Build Optionen



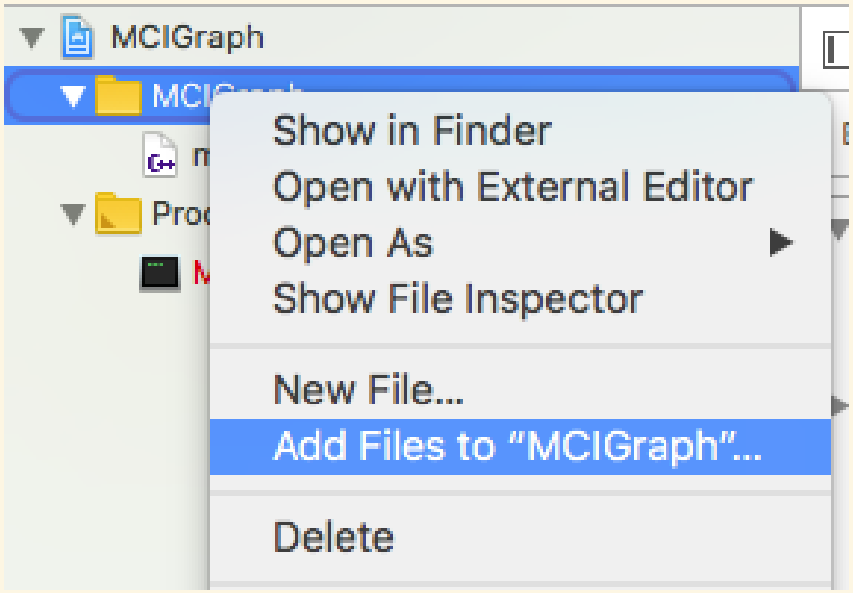
Installation von MCIGraph — XCode auf MacOS X

- Führen Sie auf dem Terminal den Befehl `sdl2-config --libs` aus und kopieren Sie das Ergebnis in die Linker Flags der Build Optionen



Installation von MCIGraph — XCode auf MacOS X

- Fügen Sie die Datei `mcigraph.hpp` aus Sakai ihrem Projekt hinzu:



Installation von MCIGraph — Generell

- Laden Sie die für ihr System passenden Bibliotheken von `https://www.libsdl.org/download-2.0.php`
- Suchen Sie im Internet wie sie ihre Entwicklungsumgebung für SDL2 einrichten können.
- Laden sie die Datei `mcigraph.hpp` von Sakai und speichern sie diese im selben Verzeichnis wie ihre Quellcodedatei.

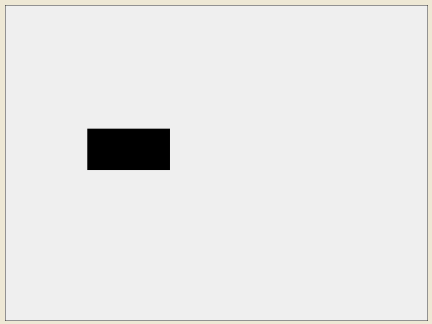
Verwendung

Hello World

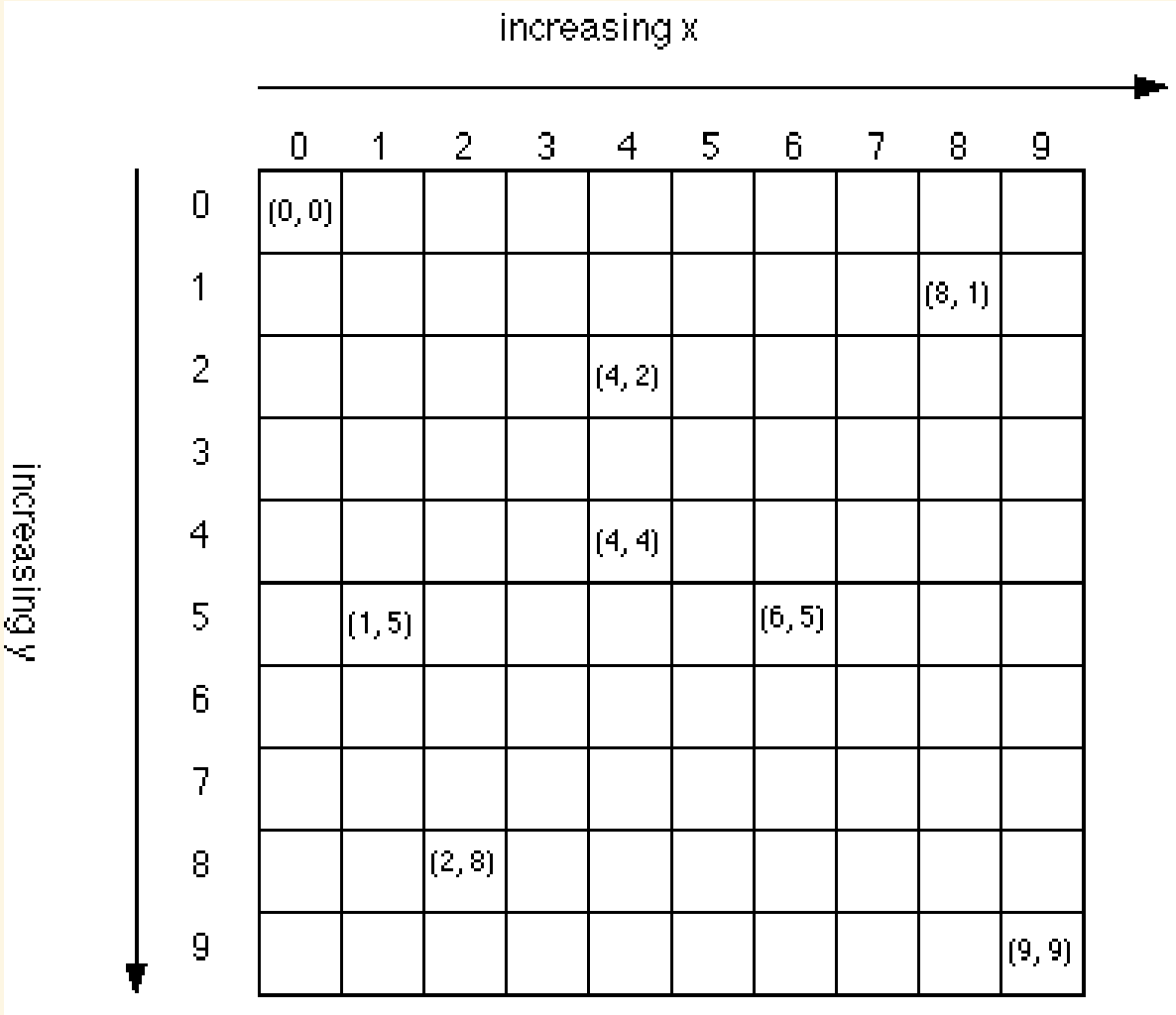
Source Code

```
#include "mcigraph.hpp"

int main(int argc, char *argv[]) {
  while (running()) {
    draw_rect(200, 300, 200, 100);
    present();
  }

  return 0;
}
```

Resultat



Das verwendete Koordinatensystem



Wir haben Koordinaten von 0 bis 1023 auf der x-Achse und von 0 bis 767 in der y-Achse!

Zeichne Linie

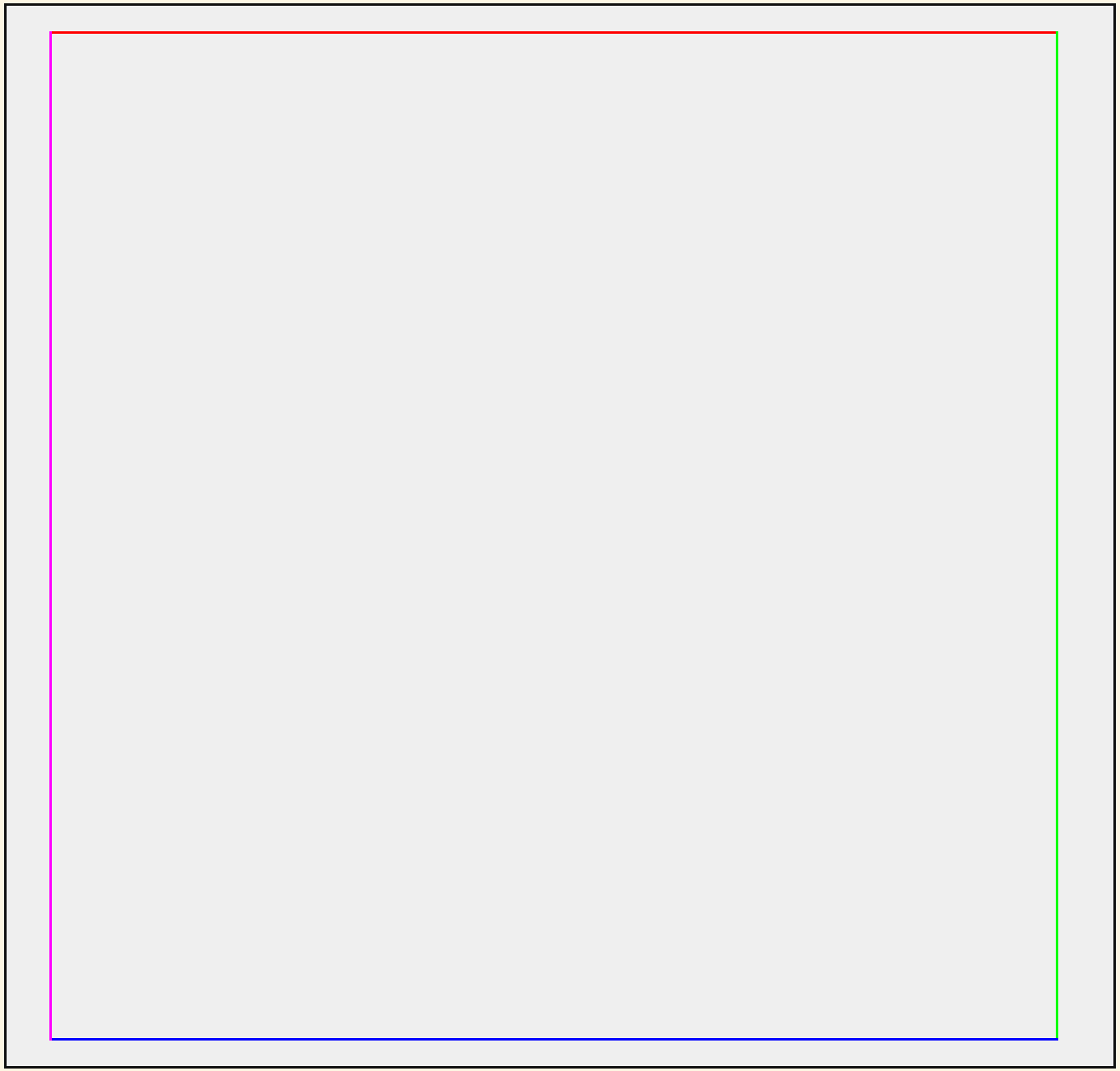
```
draw_line(int x1, int y1, int x2, int y2, [int red, int green, int blue])
```

- `x1` und `y1` : Startkoordinate der Linie
- `x2` und `y2` : Endkoordinate der Linie
- `red`, `green` und `blue` : Optionale Farbe der Linie. Jeder Farbparameter kann Werte zwischen `0` und `255` annehmen.

Zeichne Linie — Beispiel

```
#include "mcigraph.hpp"

int main(int argc, char *argv[]) {
  while (running()) {
    draw_line(100, 100, 500, 100, 255, 0, 0);
    draw_line(500, 100, 500, 500, 0, 255, 0);
    draw_line(500, 500, 100, 500, 0, 0, 255);
    draw_line(100, 500, 100, 100, 255, 0, 255);
    present();
  }
  return 0;
}
```

Zeichne Linie — Resultat



Zeichne Linie — Beispiel 2

```
#include "mcigraph.hpp"
#include <stdlib.h>

int main(int argc, char *argv[]) {
  while (running()) {

    // Zeichne 100 zufällige Linien mit zufälliger Farbe
    for (int i = 0; i < 100; i++) {
      int x1 = rand() % 1024;
      int y1 = rand() % 768;
      int x2 = rand() % 1024;
      int y2 = rand() % 768;
      int red = rand() % 256;
      int green = rand() % 256;
      int blue = rand() % 256;

      draw_line(x1, y1, x2, y2, red, green, blue);
    }
    present();
  }

  return 0;
}
```

Zeichne Linie — Resultat 2

Zeichne Punkt

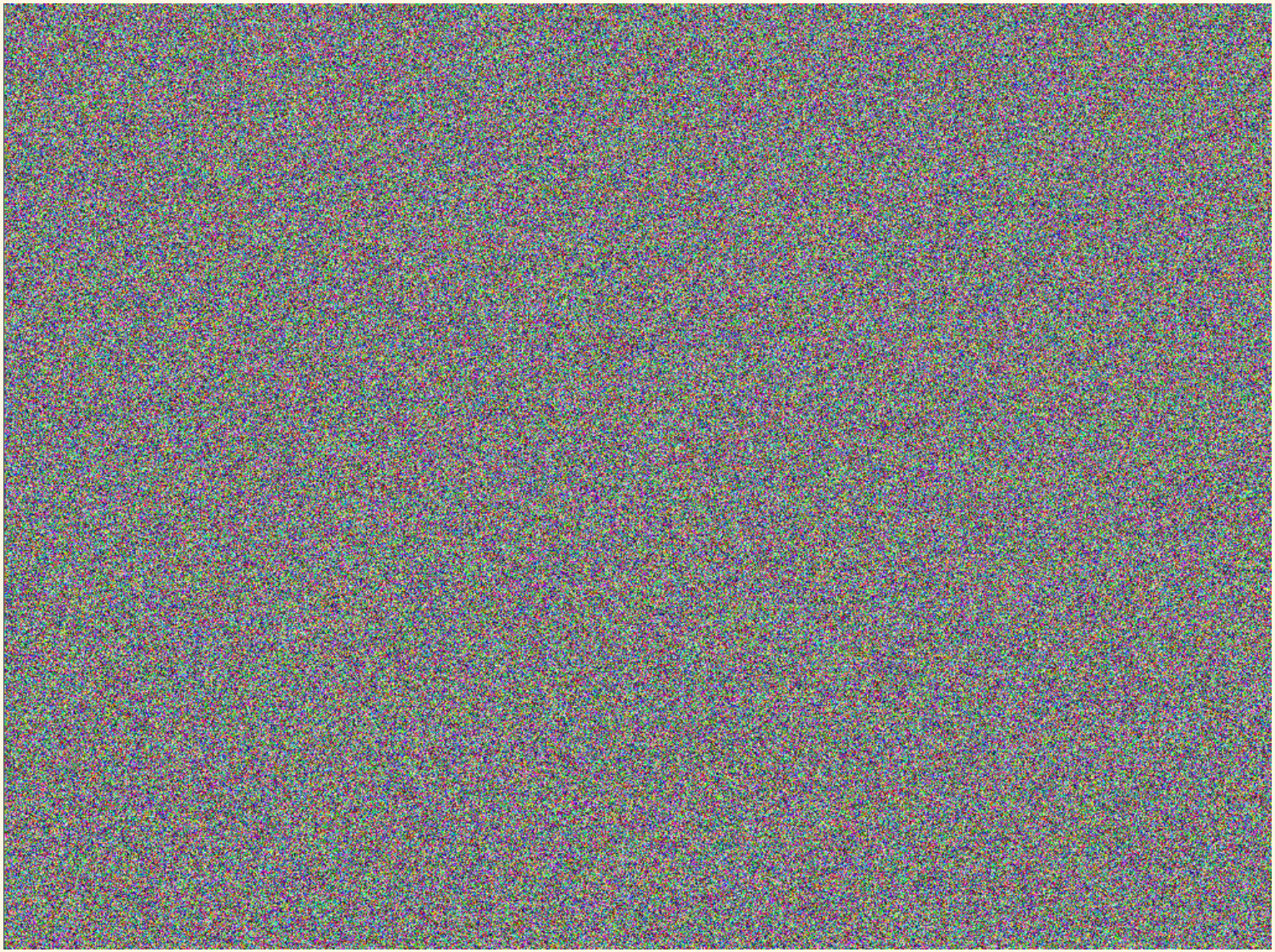
```
draw_point(int x, int y, [int red, int green, int blue])
```

- `x` und `y` : Koordinaten des Punkts
- `red`, `green` und `blue` : Optionale Farbe des Punkts. Jeder Farbparameter kann Werte zwischen `0` und `255` annehmen.

Zeichne Punkt — Beispiel

```
#include "mcigraph.hpp"
#include <stdlib.h>

int main(int argc, char *argv[]) {
  while (running()) {

    // Zeichne an jeder Koordinate einen Punkt mit zufälliger Farbe
    for (int x = 0; x < 1024; x++) {
      for (int y = 0; y < 768; y++) {
        int red = rand() % 255;
        int green = rand() % 255;
        int blue = rand() % 255;

        draw_point(x, y, red, green, blue);
      }
    }
    present();
  }

  return 0;
}
```

Zeichne Punkt — Resultat



Zeichne Rechteck

```
draw_rect(int x, int y, int width, int height,
          [bool outline, int red, int green, int blue])
```

- `x` und `y` : Koordinate des linken oberen Punkts
- `width` und `height` : Breite und Höhe des Rechtecks in Pixeln
- `outline` : Wenn `false` wird ein gefülltes Rechteck gezeichnet
- `red`, `green` und `blue` : Optionale Farbe des Punkts. Jeder Farbparameter kann Werte zwischen `0` und `255` annehmen.

Zeichne Rechteck — Beispiel

```
#include "mcigraph.hpp"
#include <stdlib.h>

int main(int argc, char *argv[]) {
  while (running()) {
    // Zeichne 100 zufällige, gefüllte Rechtecke mit zufälliger Farbe
    for (int i = 0; i < 100; i++) {
      int x = rand() % 1024;
      int y = rand() % 768;
      int width = rand() % 1024 - x;
      int height = rand() % 768 - y;
      int red = rand() % 256;
      int green = rand() % 256;
      int blue = rand() % 256;

      draw_rect(x, y, width, height, false, red, green, blue);
    }
    present();
  }

  return 0;
}
```

Zeichne Rechteck — Resultat

Zeige Bild

```
draw_image(std::string filename, int x, int y)
```

- `filename` : Dateiname des Bilds
- `x` und `y` : Koordinate des linken oberen Punkts

Mitgelieferte Bilder

Auf Sakai finden Sie die Datei `tiles.zip` welche viele Bilder im Format 16×16 Pixel enthält. Hier eine Auswahl:

Die Bereiche in Pink werden von MCIGraph transparent dargestellt

Zeige Bild — Beispiel

```
#include "mcigraph.hpp"
#include <stdlib.h>

int main(int argc, char *argv[]) {
  while (running()) {

    // Fülle das ganze Fenster mit Gras
    for (int x = 0; x < 1024; x += 16) {
      for (int y = 0; y < 768; y += 16) {
        draw_image("grass.bmp", x, y);
      }
    }
    // Setze einen Charakter in die Mitte
    draw_image("character.bmp", 32 * 16, 24 * 16);
    present();
  }

  return 0;
}
```

Zeige Bild — Resultat

Frage Tastendrücke ab

```
bool is_pressed(int keycode)
```

- `keycode` : Gibt an welche Taste man abfragen will
- Die Funktion gibt `true` zurück falls die Taste mit `keycode` aktuell gedrückt ist

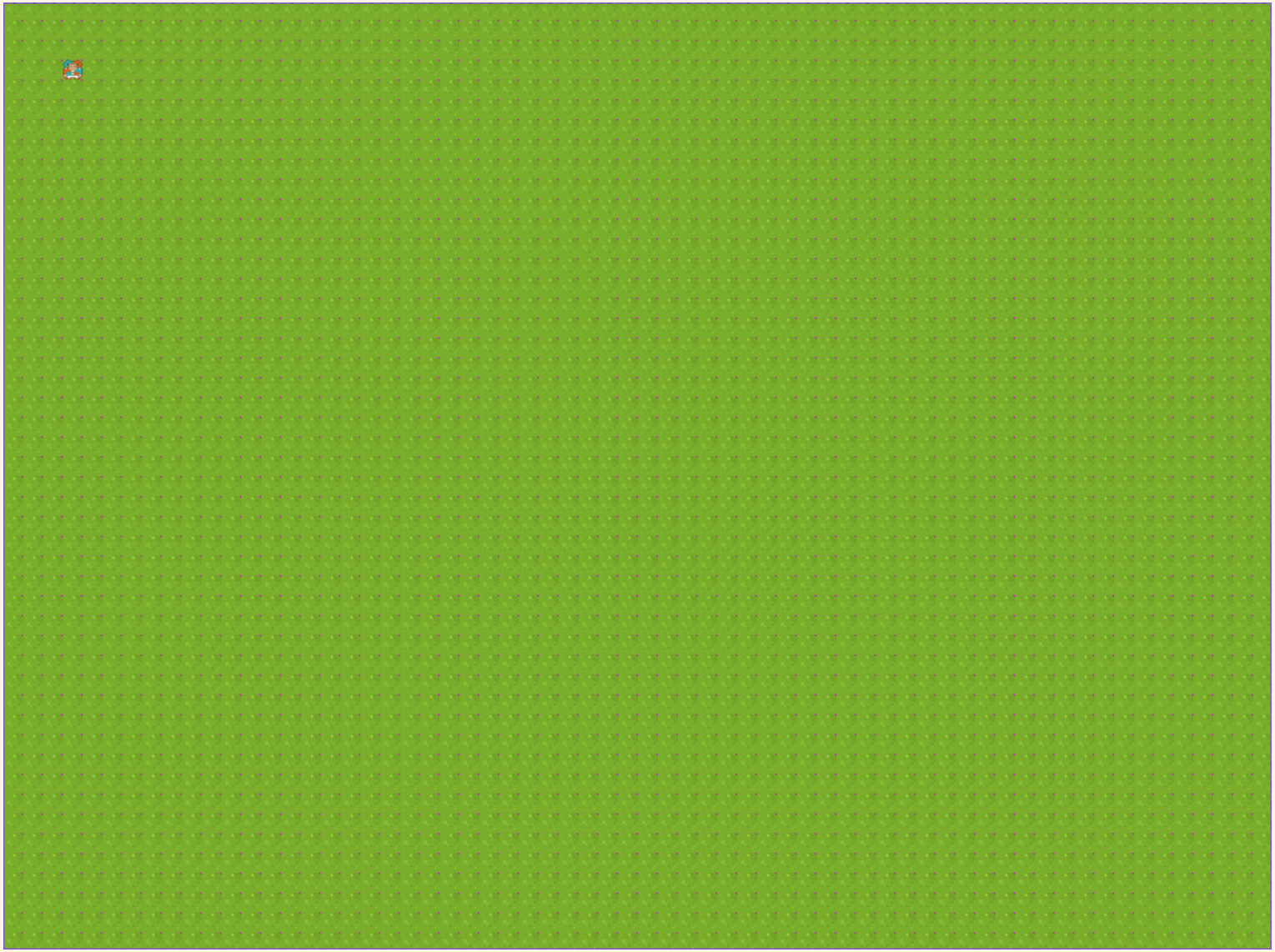
Keycodes

- `KEY_LEFT`, `KEY_RIGHT`, `KEY_UP`, `KEY_DOWN`
- `KEY_SPACE`
- `KEY_W`, `KEY_S`, `KEY_A`, `KEY_D`
- `KEY_0` ... `KEY_9`

Frage Tastendrücke ab — Beispiel

```
#include "mcigraph.hpp"
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int cx = 32, cy = 24; // aktuelle Position der Spielfigur

  while (running()) {
    for (int x = 0; x < 1024; x += 16) // Fülle das ganze Fenster mit Gras
      for (int y = 0; y < 768; y += 16)
        draw_image("grass.bmp", x, y);

    // Bewege Figur je nach Tastendrücken
    if (is_pressed(KEY_LEFT)) cx--;
    if (is_pressed(KEY_RIGHT)) cx++;
    if (is_pressed(KEY_UP)) cy--;
    if (is_pressed(KEY_DOWN)) cy++;

    draw_image("character.bmp", cx * 16, cy * 16); // Setze die Spielfigur
    present();
  }
  return 0;
}
```

Frage Tastendrücke ab — Resultat



Frage Tastendrücke ab 2

Alternativ zu `is_pressed` gibt es auch die Funktion `was_pressed(int keycode)`. Im Gegensatz zu `is_pressed` liefert diese Funktion nur `true` zurück wenn die Taste seit dem letzten Aufruf erneut gedrückt wurde. Damit verhält sich diese Funktion eher so wie man es von einer normalen Tastatureingabe gewöhnt ist.

Versuch

Ersetzen Sie alle `is_pressed`-Aufrufe im letzten Programm durch `was_pressed`- Aufrufe und beobachten Sie das geänderte Verhalten.

Zusätzliche Funktionalität

- Mit `set_delay(int x)` kann eingestellt werden wie schnell das Programm läuft (wie häufig der Code ausgeführt wird). Ein höherer Wert führt zu einer langsameren Ausführung.

Übung

1. Erweitern Sie das vorherige Programm so, dass zwei Figuren angezeigt werden
2. Sorgen sie dafür, dass die zweite Figur mit den Tasten WSAD steuerbar ist
3. Verhindern Sie für beide Figuren, dass sie aus dem Spielfeld laufen können
4. Verhindern Sie, dass beide Figuren auf dem selben Feld stehen können