Spielekonsole B

Fortgeschrittenenpraktikum SS 2017

Robert Schütz, Daniela Kilian, Stefan Müller

21. September 2017

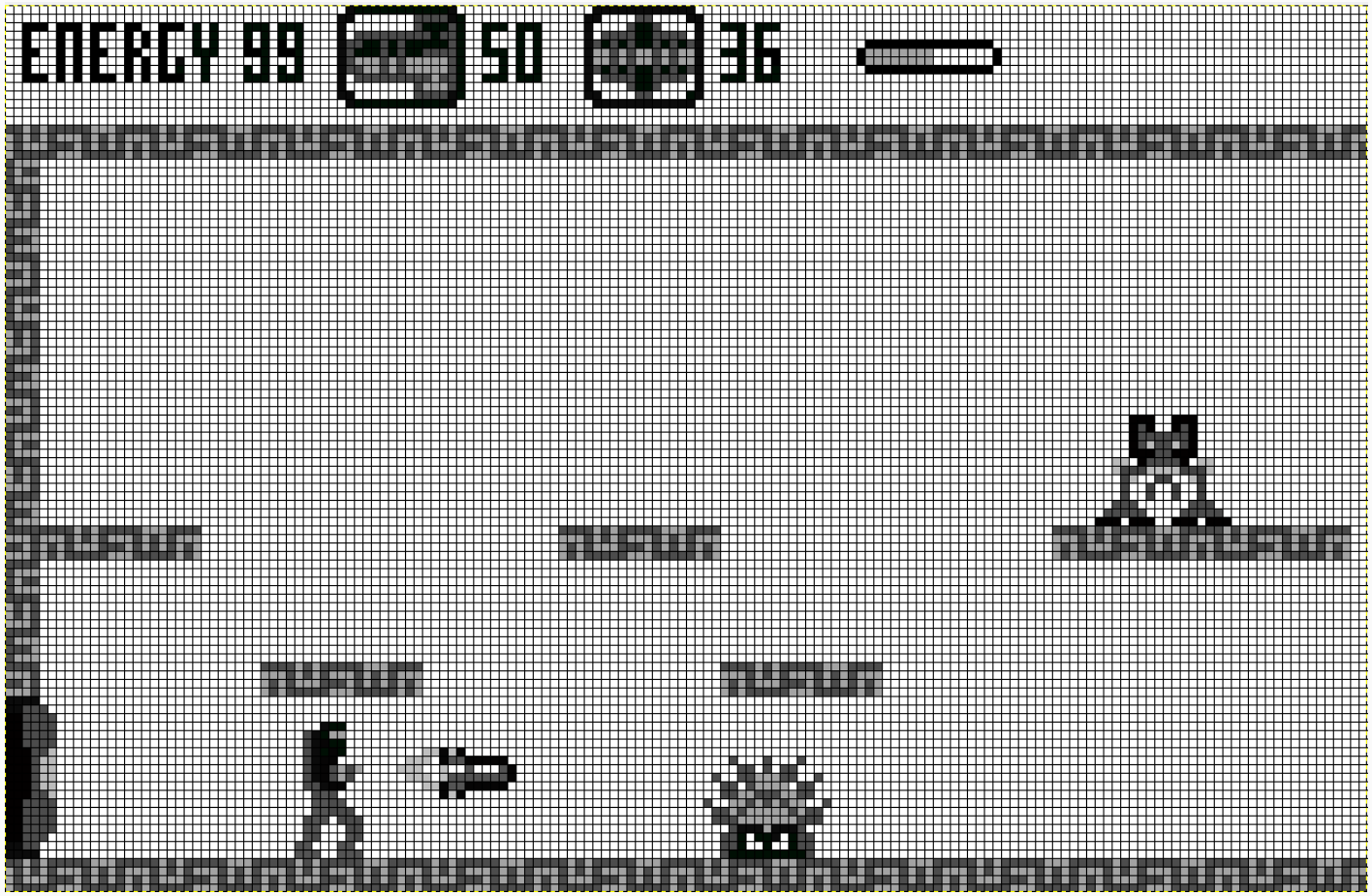
Die Crew

Stefan Müller Daniela Kilian Robert Schütz

Spielidee

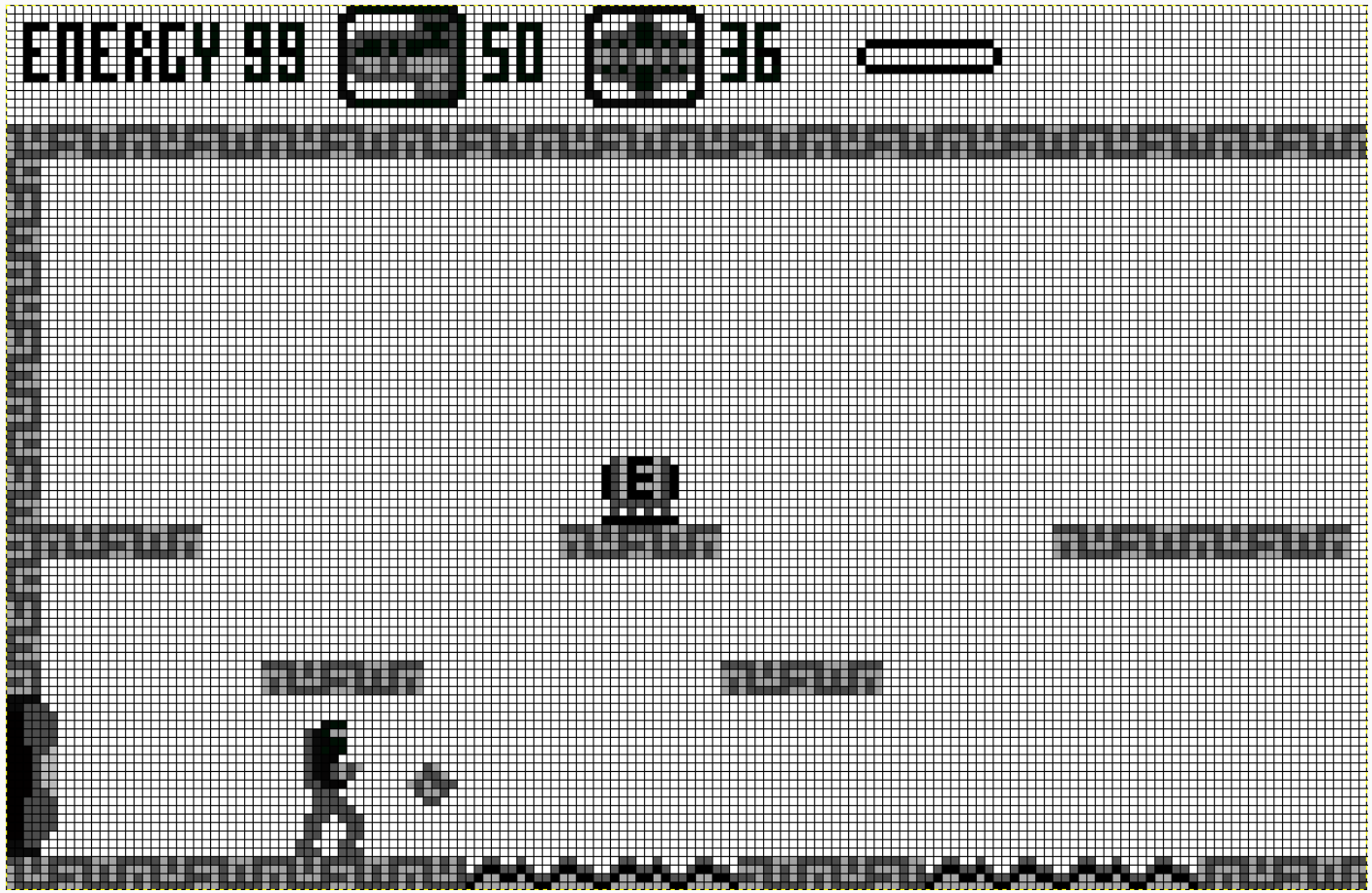
Das Spiel: Metro HD

Spielaufbau: Schritt 1



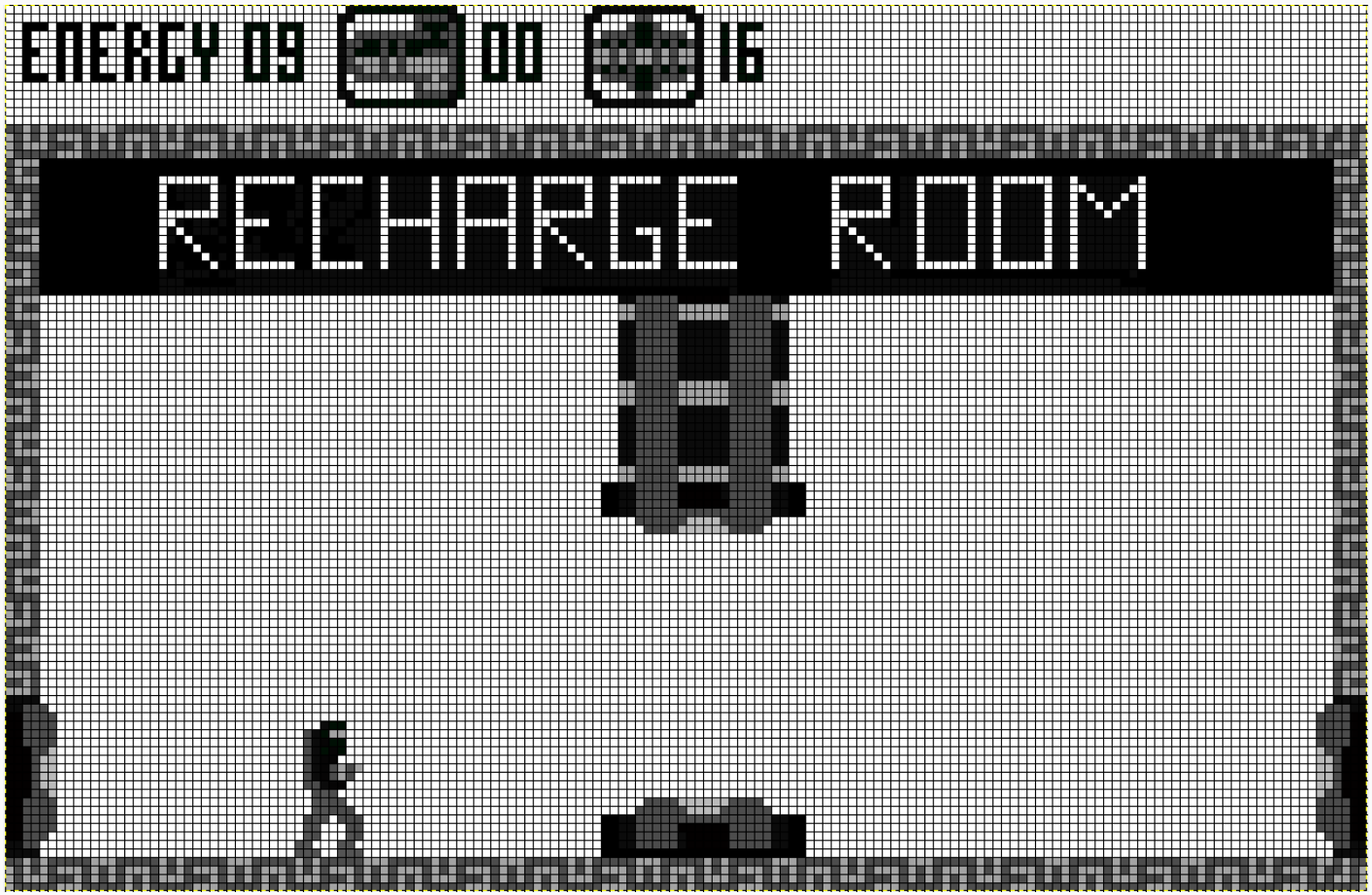
Kämpfe gegen verschiedene Monster!

Spielaufbau: Schritt 2



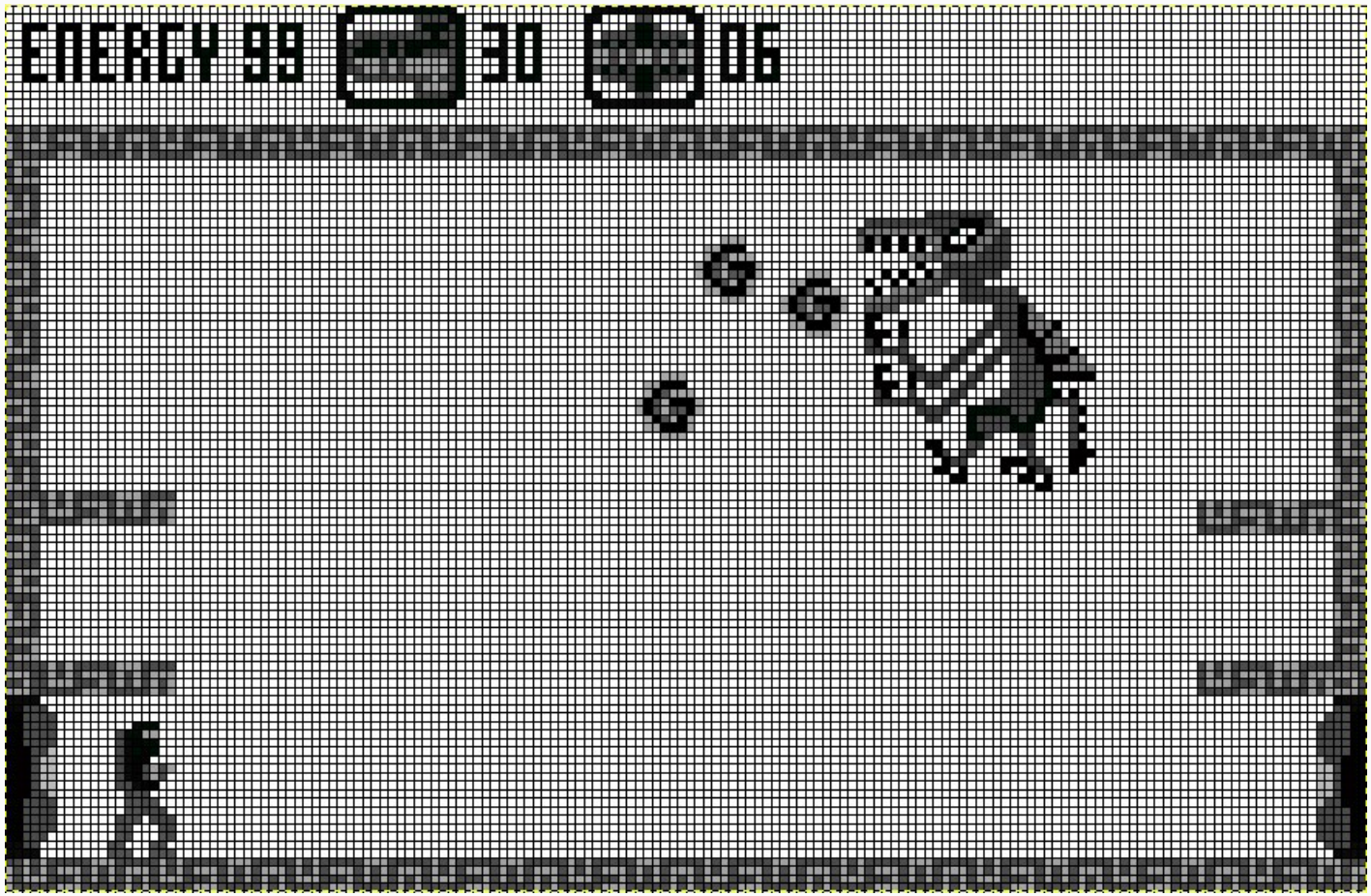
Entdecke aufregende Level!

Spielaufbau: Schritt 3



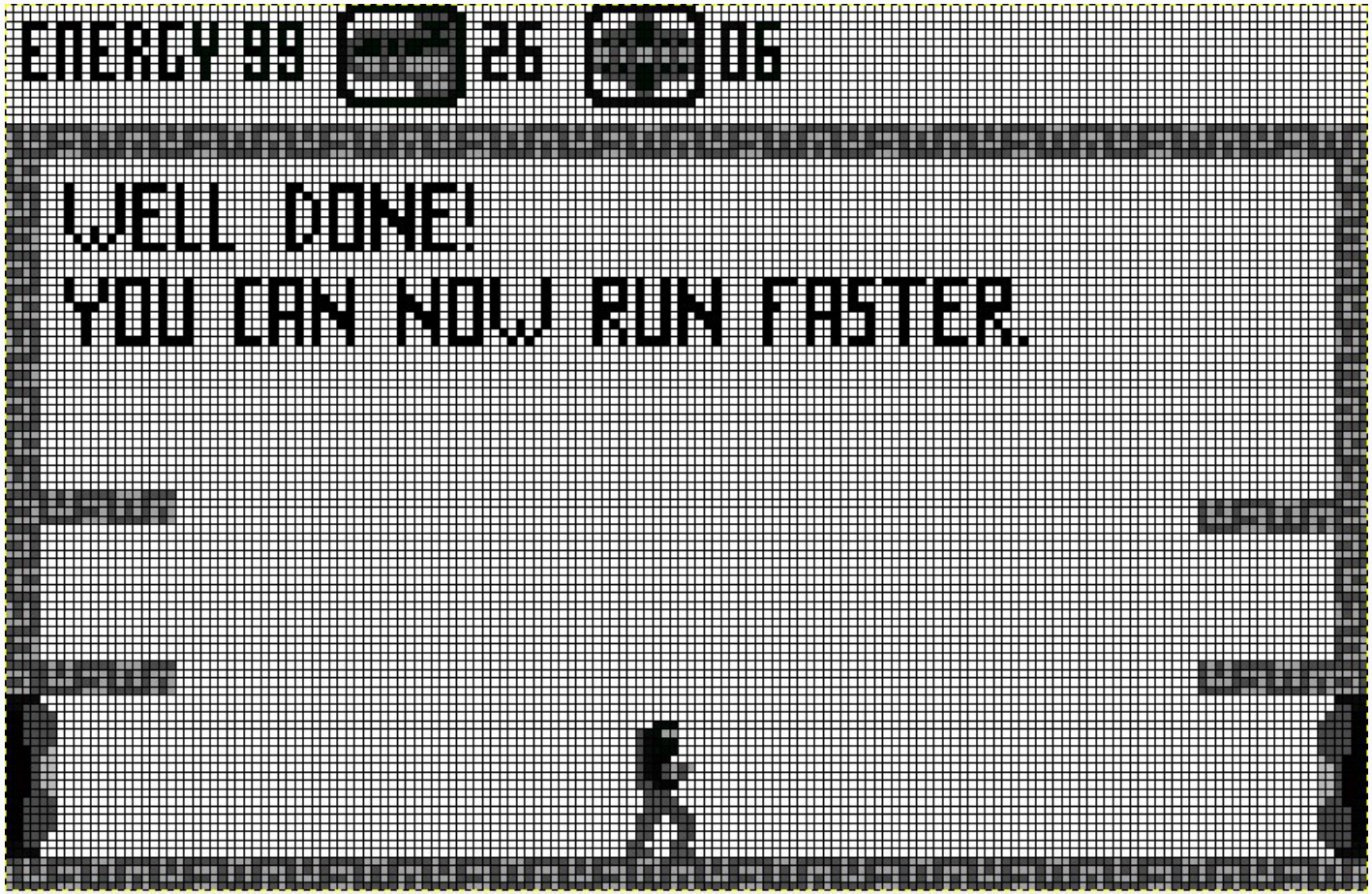
Bereite dich auf einen anstrengenden Kampf vor!

Spielaufbau: Schritt 4



Stelle dich gefährlichen Endbossen!

Spielaufbau: Schritt 5



Verdiene dir mächtige Power-Ups!

Spielumsetzung

Zeichen: Sprites

Das Display zeichnet immer vier Pixel untereinander auf einmal. Ein Python-Skript vereinfacht das Zeichnen:

```
const PROGMEM uint8_t little[16] = {
    0b00000100, 0b00010100,
    0b10010000, 0b10000000,
    0b10000000, 0b10010000,
    0b00010100, 0b00000100,
    0b00000000, 0b00001000,
    0b11111010, 0b01111001,
    0b01111001, 0b11111010,
    0b00001000, 0b00000000
};
```

Zeichnen: Display

Idee: Window-Funktionalität des Displays benutzen

(30)	Set Window Start Column	0	1	1	1	1	0	1	0	0	Set Start Column of Window Function
			WPC0[7..0]							
(31)	Set Window Start Page	0	1	1	1	1	0	1	0	1	Set Start Page of Window Function
			0	0	0	WPP0[4..0]				
(32)	Set Window End Column	0	1	1	1	1	0	1	1	0	Set End Column of Window Function
			WPC1[7..0]							
(33)	Set Window End Page	0	1	1	1	1	0	1	1	1	Set End Page of Window Function
			0	0	0	WPP1[4..0]				
(34)	Set Window Enable	0	1	1	1	1	1	0	0	C4	C4: 0=disable, 1=enable Window Function (disable before changing column and pages)

```
void drawsprite(uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint8_t* sprite)
{
    enable_window(x, y, width, height);
    for (uint16_t i = 0; i < width * height; ++i)
        sendbyte(pgm_read_byte_near(sprite + i), 1);
    disable_window();
}
```

⇒ Schneller als **page( )**

Zeichnen: Pixelweise

```
void drawsprite_px(uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint8_t* sprite)
{
    uint8_t offset = 2 * (y % 4);
    if (offset == 0)
    {
        drawsprite(x, y / 4, width, height / 4, sprite);
    }
    else
    {
        enable_window(x, y / 4, width, height / 4 + 1);
        uint16_t i = 0;
        for (; i < width; ++i)
            sendbyte(pgm_read_byte_near(sprite + i) << offset, 1);
        for (; i < height / 4 * width; ++i)
            sendbyte(pgm_read_byte_near(sprite + i) << offset |
                     pgm_read_byte_near(sprite + i - width) >> (8 - offset), 1);
        for (; i < (height / 4 + 1) * width; ++i)
            sendbyte(pgm_read_byte_near(sprite + i - width) >> (8 - offset), 1);
        disable_window();
    }
}
```

struct Character

Alle beweglichen Objekte werden als `Character` repräsentiert.

```
struct Character
{
    uint8_t x;
    uint8_t y;
    enum {LOOK_MONSTER_MEMU, LOOK_PROTAGONIST, LOOK_FIREBALL, ...} look;
    uint8_t lookstate; // to e.g. store whether the wings are turned upwards or downwards
    uint32_t lastlookstatechg;
    uint8_t width;  // in pixels
    uint8_t height; // in pixels
    enum {DIRECTION_LEFT, DIRECTION_RIGHT} direction;
    enum {DIRECTION_UP, DIRECTION_DOWN} verticaldirection;
    int8_t jumpstate;
    uint8_t initial_health;
    int8_t health;
    uint8_t damage;
    uint8_t jumpheight;
    enum {FOLLOW_PROTAGONIST, BACK_AND_FORTH, ...} movement;
    uint8_t x_pace;
    uint8_t y_pace;
};
```

Spielablauf

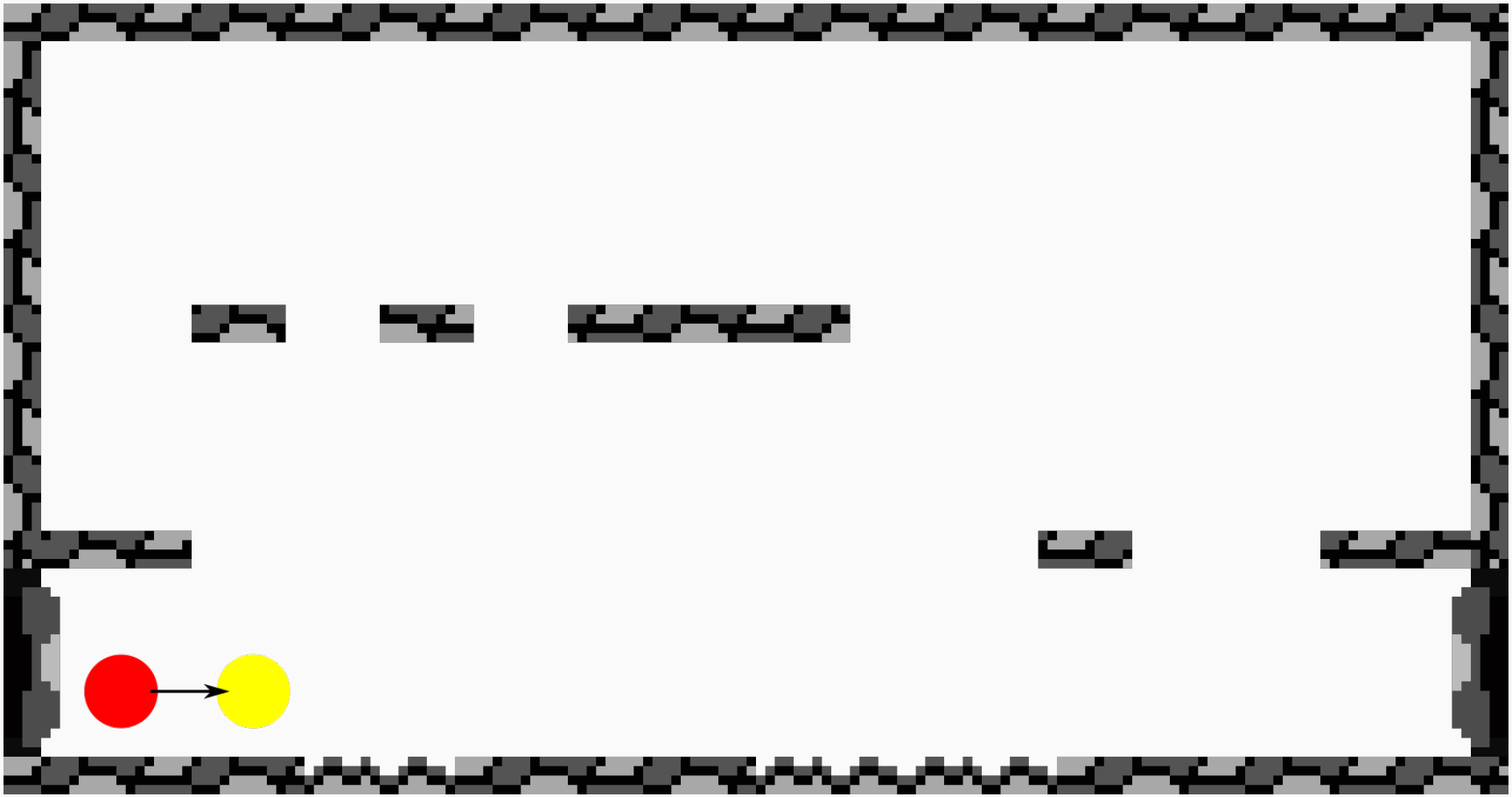
```
while(1)
{
    if (nextmoveevent < getMsTimer())
    {
        if (B_RIGHT)
        {
            moveright(protagonist);
            nextmoveevent = getMsTimer() + 50;
        }
        ...
    }
    if (projectile->movement == HIDDEN
        && num_rockets > 0
        && nextshootevent < getMsTimer()
        && B_A)
    {
        projectile->movement = PROJECTILE;
        draw(projectile);
        num_rockets--;
        eeprom_write_byte(&num_rockets_stored, num_rockets);
        nextshootevent = getMsTimer() + 500;
    }
    if (monster->movement != HIDDEN && collision(protagonist, monster))
    {
        takingdamage(monster->damage);
    }
    ...
}
```

Zufällige Plattformen

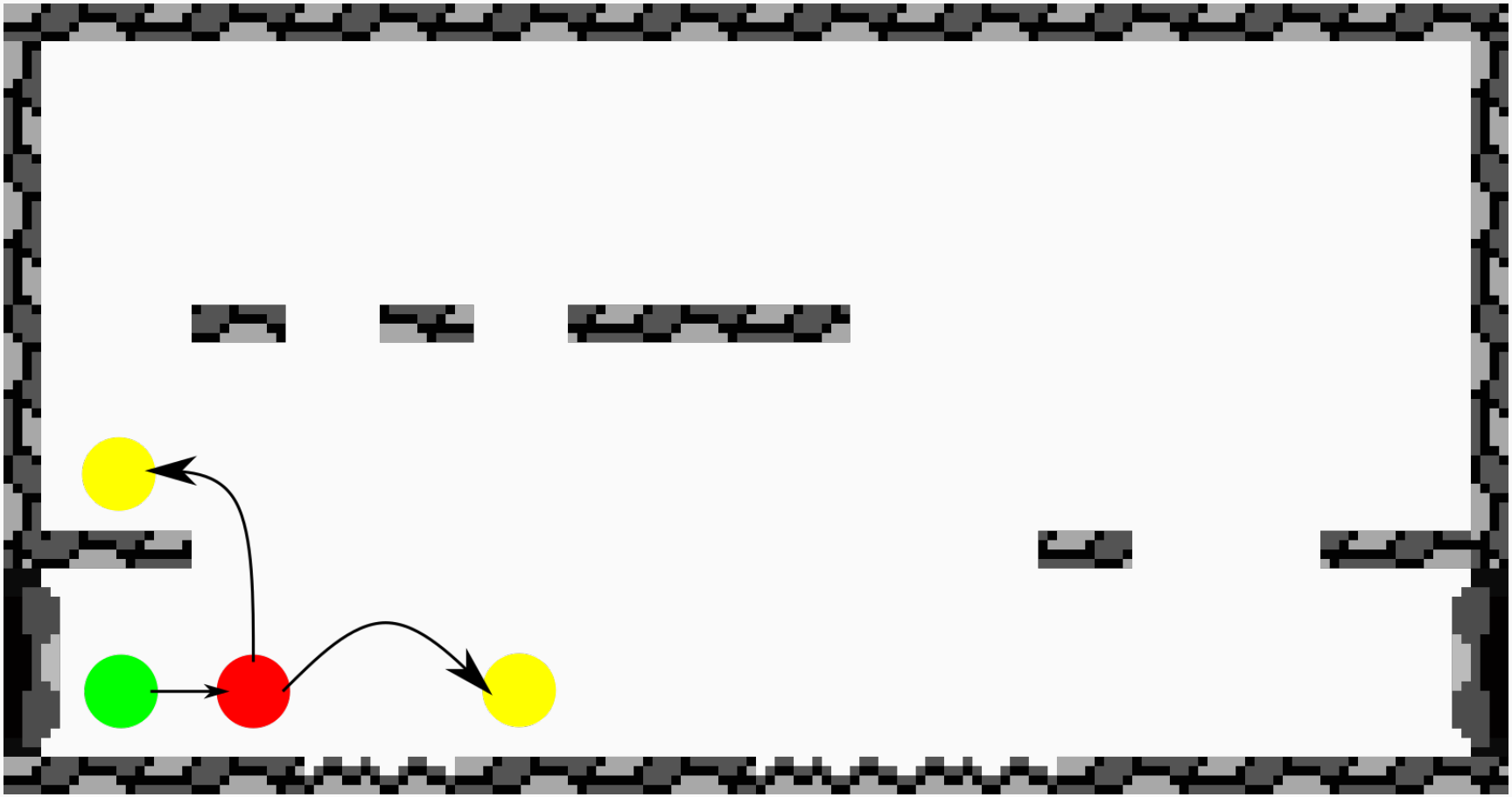
```
srandom(level_seed + level_pos);
platforms_13 = random();
platforms_19 = random();
platforms_24 = random();
nofloor = random();

bool obstacle(uint8_t x, uint8_t y)
{
    if (y >= 19 * 4 && y < 20 * 4)
        return !(platforms_19 & (3l << (x / PLATFORM_WIDTH * 2)));
    else if (y >= 13 * 4 && y < 14 * 4)
        return !(platforms_13 & (3l << (x / PLATFORM_WIDTH * 2)));
    else if (y >= 24 * 4 && y < 25 * 4)
        return !(platforms_24 & (3l << (x / 16 * 2)));
    else if (y >= FLOOR_Y && y < FLOOR_Y + 4)
        return nofloor & (3l << x / 16 * 2);
    else
        return false;
}
```

Die `obstacle()` Funktion dient dazu, herauszufinden, ob an einer gegebenen Stelle eine Plattform ist.

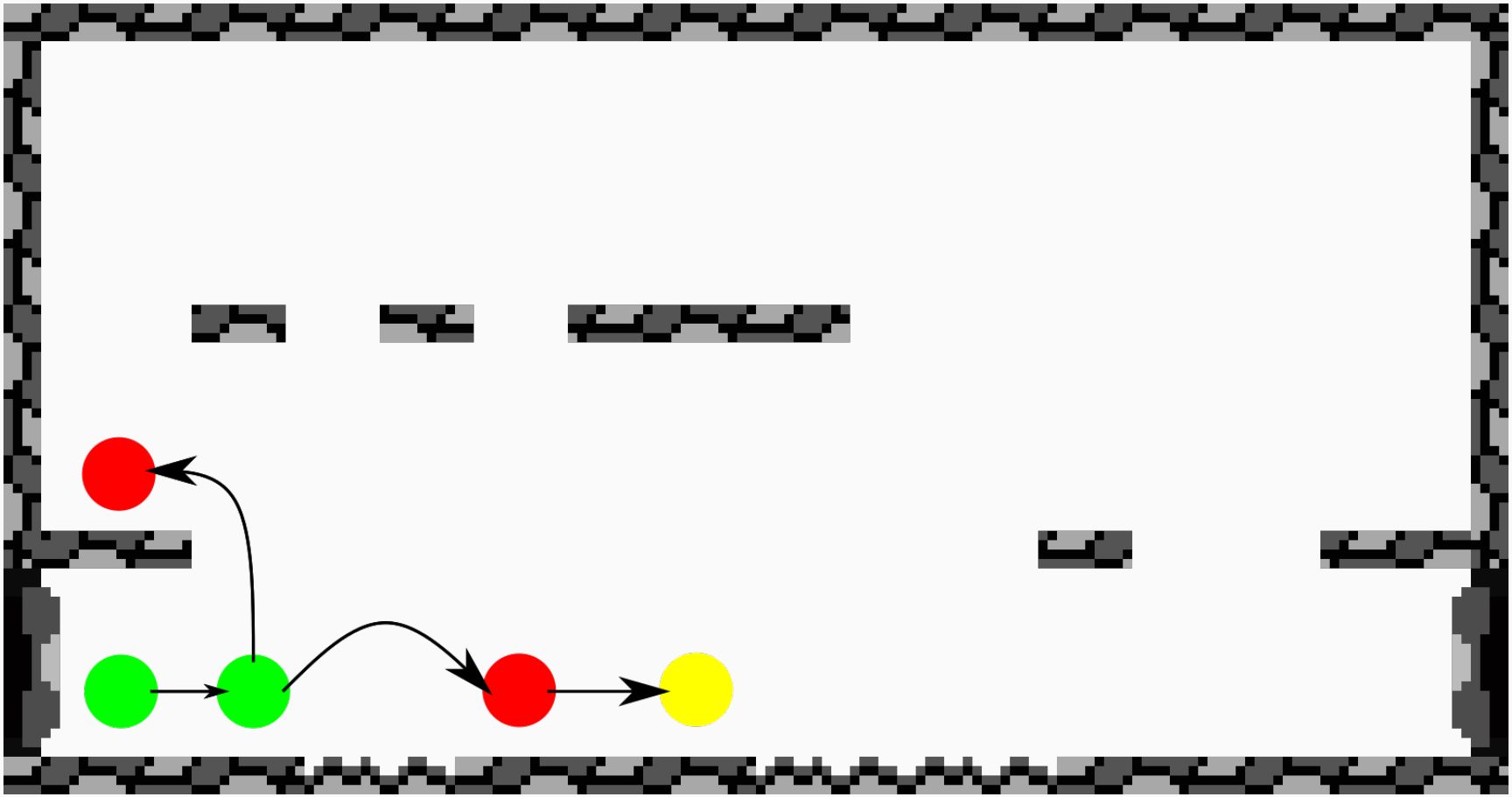
Tiefensuche: Schritt 1



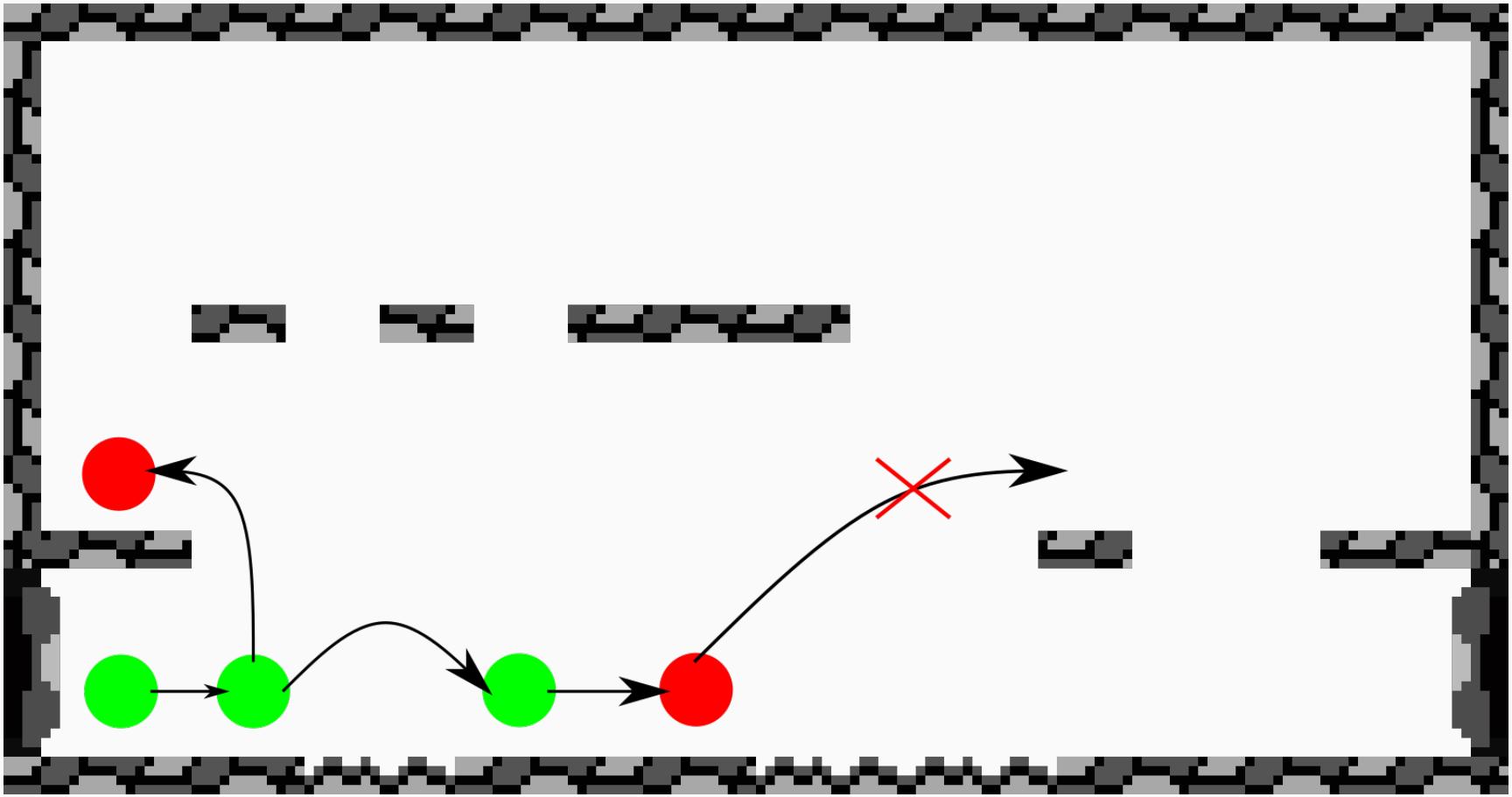
Tiefensuche: Schritt 2



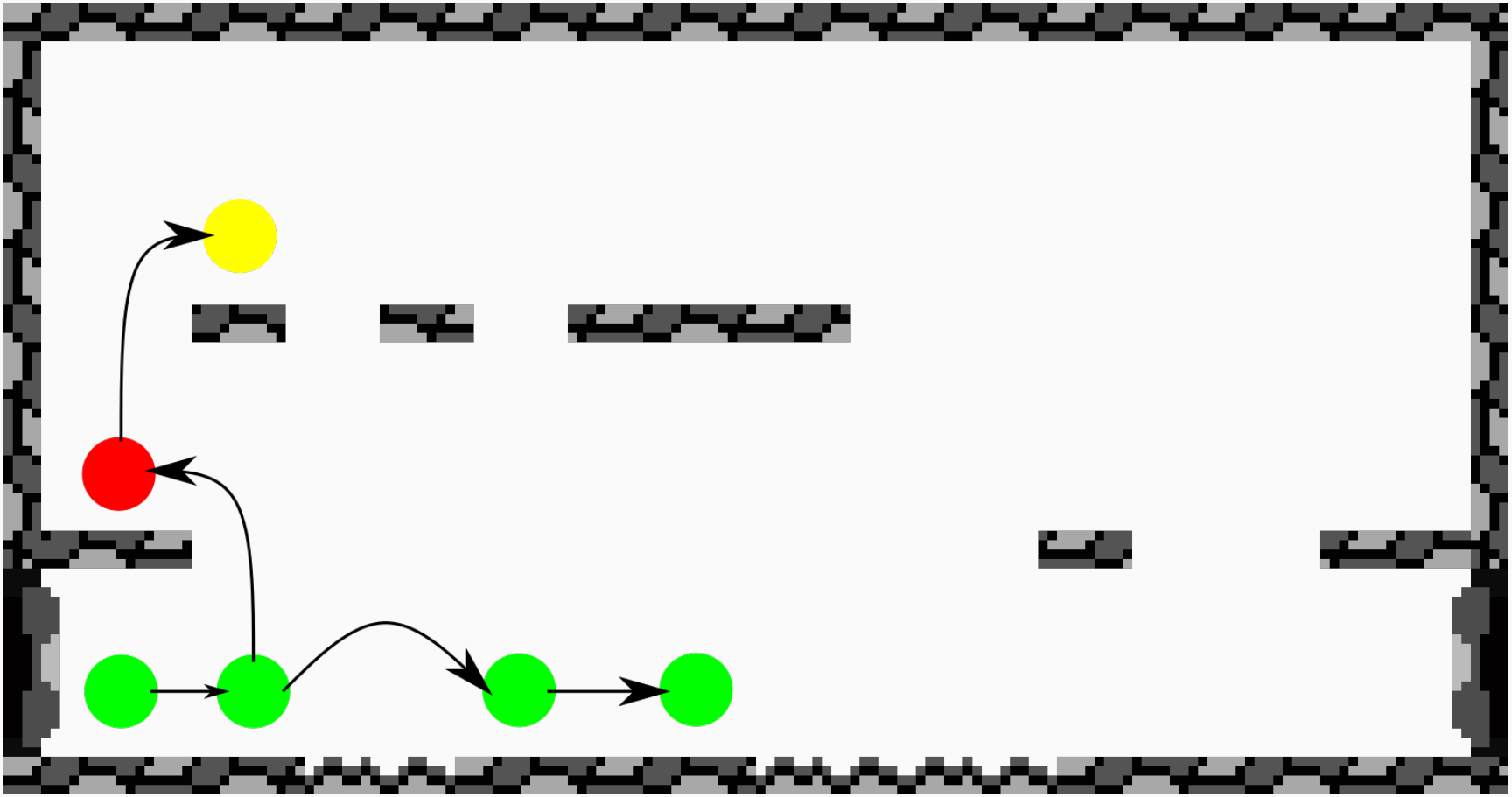
Tiefensuche: Schritt 3



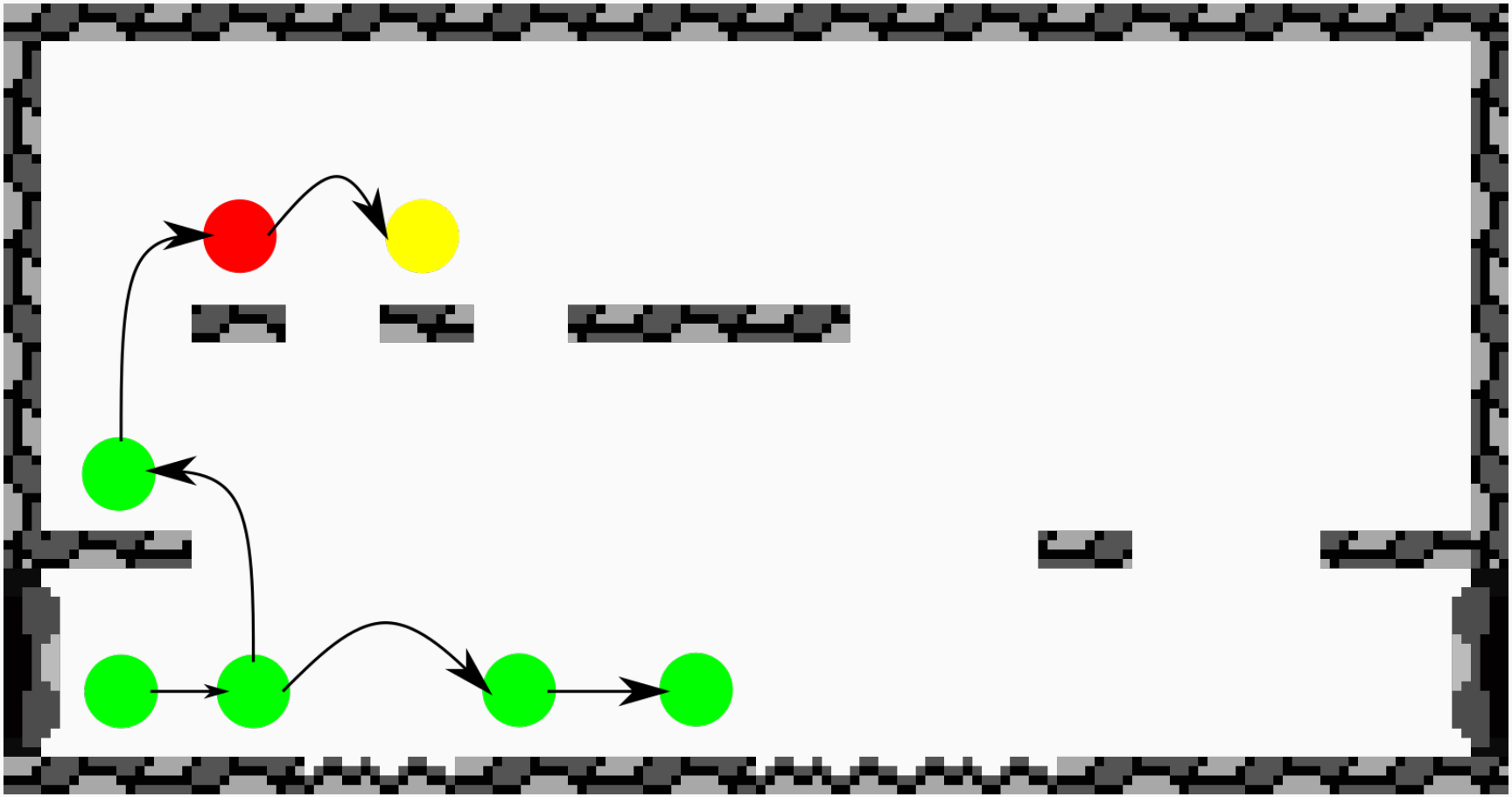
Tiefensuche: Schritt 4



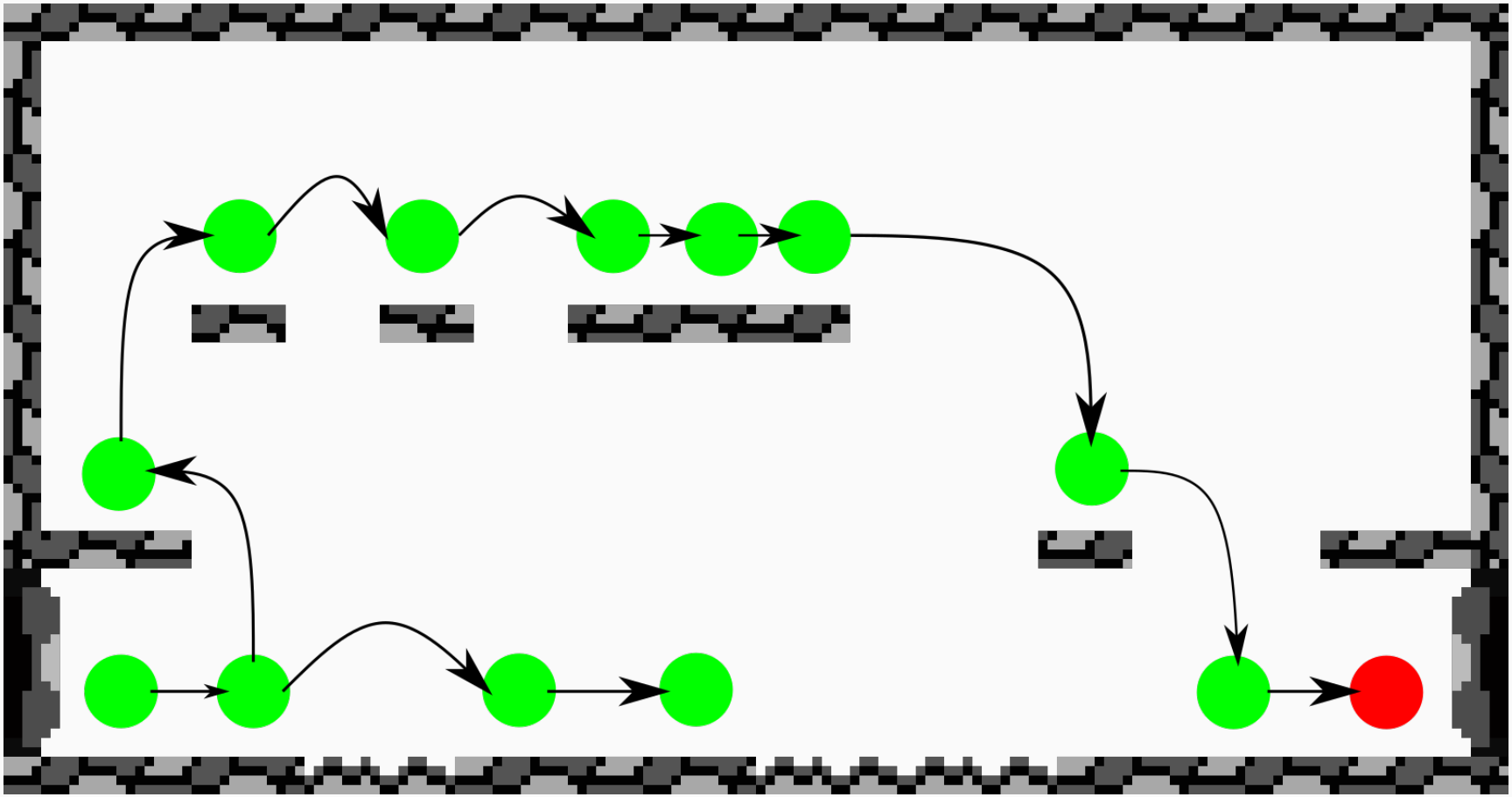
Tiefensuche: Schritt 5



Tiefensuche: Schritt 6



Tiefensuche: Letzter Schritt

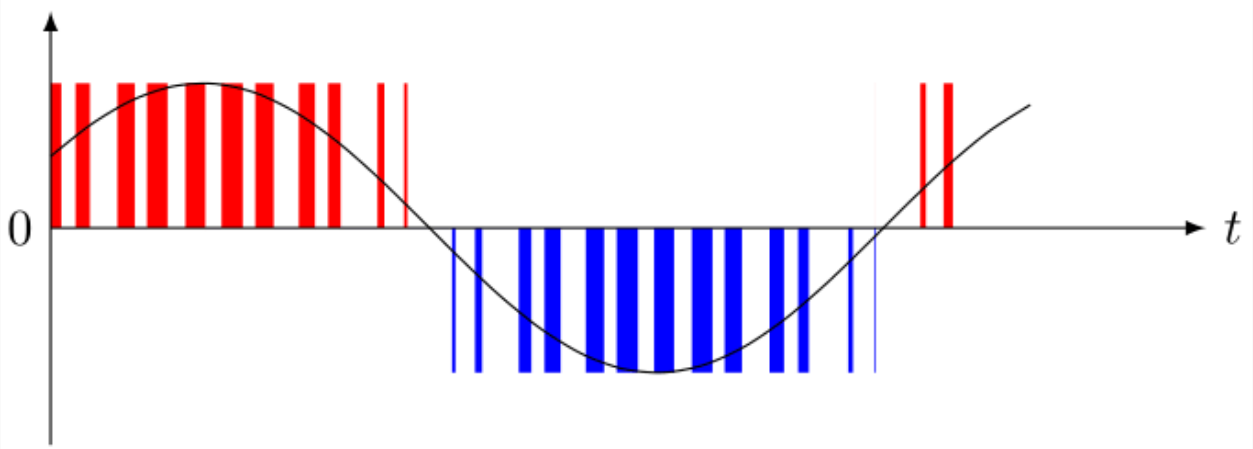


Sound

Timer

Wir verwenden zwei Timer:

- Timer1
  - Frequenz: 62 500 Hz
  - Toggelt Pin B1
  - Pulsweite bestimmt „Ausschlag“ der Welle



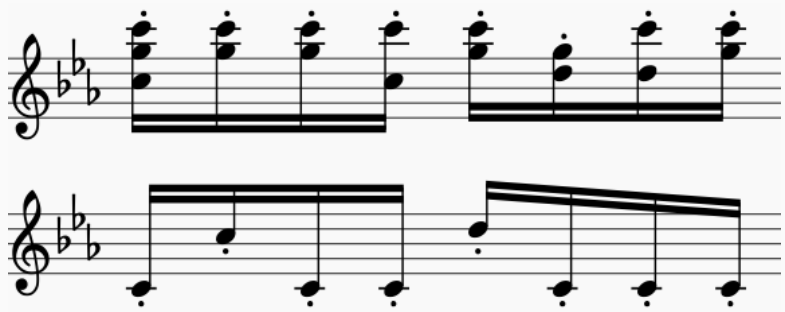
Timer

- Timer2
  - Frequenz des Interrupts: 15 625 Hz
  - Dient zur Zeitmessung
  - Legt den aktuellen Ausschlag fest:
    Für einen Ton mit 440 Hz wird bei jedem Aufruf des Interrupts die Pulsweite (max. 255) um

    $$255/(15625/440) \approx 7,18$$

    erhöht. Für eine höhere Genauigkeit werden **`uint16_t`**s verwendet.

MIDI einlesen



```
const Event boss4[] PROGMEM = {
  { { .track = 0, .increment = 615, .delay = 0 } },
  { { .track = 1, .increment = 307, .delay = 0 } },
  { { .track = 2, .increment = 2463, .delay = 2812 } },
  { { .track = 2, .increment = 1231, .delay = 2812 } },
  { { .track = 0, .increment = 1231, .delay = 0 } },
  ...,
  STOP
};
```

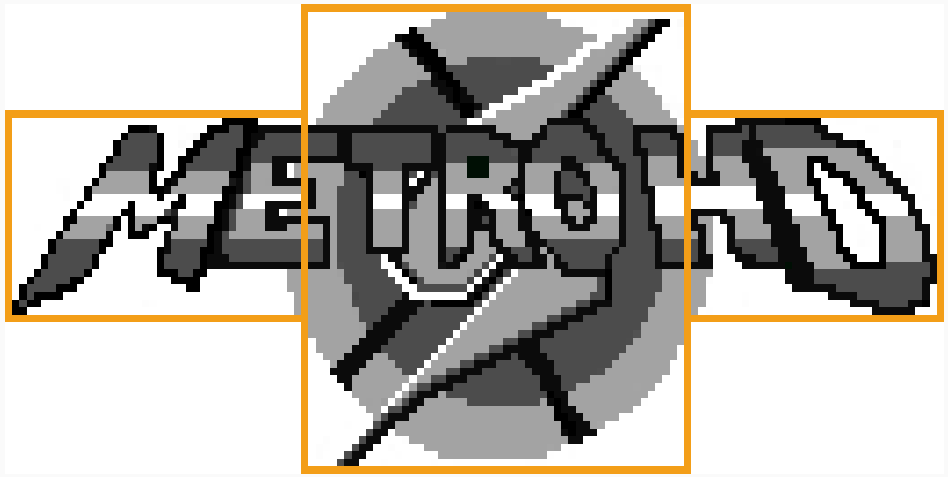
Probleme und Verbesserungen

Aufgetretene Probleme

- Problem: Speicherplatzmangel

```
avrdude: verifying ...
avrdude: 31196 bytes of flash verified

avrdude: safemode: Fuses OK (E:FF, H:D7, L:FF)

avrdude done.  Thank you.
```

- Lösung: Ablegen der Sprites im PROGMEM und effiziente Aufspaltung von großen Bildern



Mögliche Verbesserungen

Hardware

- Tiefpass einbauen
- Farbdisplay

Software

- (Noch) Effizientere Implementierung
- Highscore hinzufügen
- Spiel weiter ausbauen:
  - Neue Monster, Endbosse und Power-Ups
  - Weitere Waffen
  - Geheimwege

DANKE FÜR DIE AUFMERKSAMKEIT