SWO3	Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3	WS 2018/19, Angabe 6

☒ Gruppe 1 (J. Heinzelreiter)

☐ Gruppe 2 (M. Hava) Name: Papesh Konstantin Aufwand [h]: 10

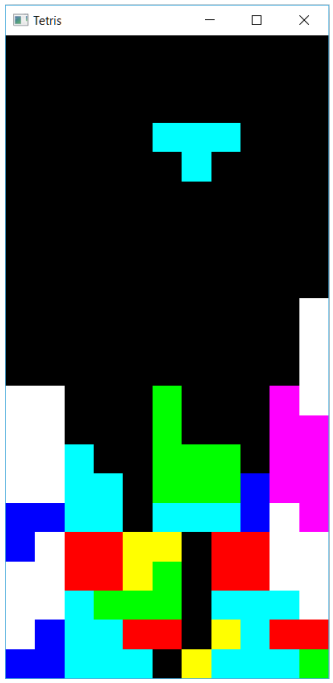
☐ Gruppe 3 (P. Kulczycki) Übungsleiter/Tutor: _____ Punkte: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (100 P)	100	95	90

Beispiel 1: GLFW-Applikation „Tetris“ (`src/tetris/`)

Vervollständigen Sie die in der Übung begonnene GLFW-Applikation „Tetris“. Beachten Sie dabei die folgenden Anforderungen und Hinweise:

1. Es müssen prinzipiell beliebig geformte Spielsteine (Tetriminos) unterstützt werden. Ein Spielstein ist dabei eine Matrix aus eingefärbten „Pixeln“. Natürlich gibt es in Ihrer Applikation einen vordefinierten Satz mit den sieben bekannten Tetriminos I, J, L, O, S, T und Z.
2. Tetriminos müssen die Bewegungen „links“, „rechts“, „drehen“ und „fallen“ durchführen können.
3. Führen Sie eine entsprechende Kollisionsbehandlung durch. Nur die Berücksichtigung eines „minimal umgebenden Rechtecks“ ist nicht ausreichend.
4. Komplett gefüllte Reihen verschwinden vom unteren Rand des Bildschirms.
5. Die Fallgeschwindigkeit der Tetriminos erhöht sich mit Fortgang eines Spiels.
6. Ein Spiel endet, sobald die nicht abgebauten Reihen den Bildschirm füllen.
7. Strukturieren Sie sauber, indem Sie Module und (Hilfs-)Funktionen schreiben. Auch die in der Übungsstunde vorgezeigten und implementierten Funktionen können noch besser strukturiert werden.
8. Siehe die Quelle https://en.wikipedia.org/wiki/Tetris und vor allem die Quelle http://tetris.wikia.com/wiki/Tetris_Guideline.



SWO31 & SWB31 Softwareentwicklung mit klassischen Sprachen und Bibliotheken – WS 2018/19
Übungsabgabe 6

Konstantin Papesh

25. November 2018

6.1 GLFW-Applikation „Tetris“

6.1.1 Lösungsidee

Grundsätzlich soll die Funktion des ursprünglichen Tetris abgebildet werden. Als Vorlage dafür wurde uns im Zuge des Unterrichts die Basisversion von Tetris auf einem Gameboy präsentiert.

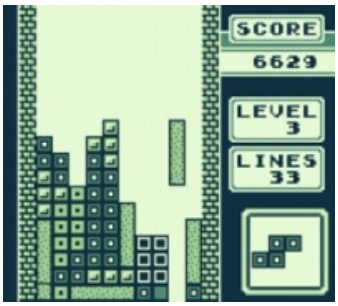


Abbildung 6.1: Tetris auf einem Gameboy. Quelle: de.wikipedia.org

Dabei ist auf eine sorgfältige, in mehrere Module unterteilte Ausarbeitung zu achten. Als zusätzliche Bibliotheken wurden OpenGL und GLFW verwendet, diese sind für die plattformunabhängige Grafikausgabe zuständig. Dabei wurde das Grundgerüst für diese schon vorgegeben, diese wurde also nicht selbstständig ausgearbeitet. Lediglich die Zusammenstellung der Würfel wurde im Zuge dieser Übung ausprogrammiert.

main.c

In diesem File spielt sich der Hauptteil des Programms ab. Hier ist der Code für die Grafikausgabe zu finden. Ausserdem findet hier die Verarbeitung der Tasteneingaben statt, jeweils für *links*, *rechts*, *drehen* und *runter*. Ausserdem wird ein primitiver Timer eingeführt, welcher die Steine in einem bestimmten Interval nach unten fallen lässt.

random.c

enthält Funktionen für die zufällige Auswahl von Spielsteinen als auch Farben.

timer.c

ist für die Verwaltung des implementierten Timer zuständig. Dabei wird im Hauptprogramm immer wieder die Funktion *timer_test* aufgerufen, die vergleicht die verstrichene Zeit mit dem vorher festgesetzten *timer_interval*. Ist die Intervalzeit kleiner als wird der Callback auf die *on_timer* gestartet und der timer zurückgesetzt. Ausserdem wird der *timer_interval* verkleinert im den Faktor *TIMER_MULTIPLICATOR*, dies sorgt für eine verschnellerte Fallzeit der Spielsteine.

try.c

implementiert die *Verschiebfunktionen*. Hier wird jeweils geprüft, ob die gewünschte Aktion mit dem Spielstein überhaupt zulässig ist[1]. Dabei wird ein *virtueller* Spielstein erstellt und um die gewünschte Aktion verschoben bzw. gedreht. Entsteht keine Kollision mit existierenden Spielsteinen oder dem Bildschirmrand, ist die Aktion zulässig und der originale, übergebene Spielstein wird auf die Position bzw. Drehung des virtuellen Spielsteins gesetzt.

types.c

ist für die Initialisierung des *form_types*-Array zuständig. In diesem befinden sich alle möglichen Spielstein-Formen. Ausserdem enthält die Datei eine Hilfsfunktion zum Drehen eines Spielsteins, sowie die Renderfunktionen für einen *Quad*, einen *Block* und des gesamten Spielsteins (auch *form* im Programmcode genannt).
Die Drehfunktion wurde so simpel wie möglich gehalten. Es wird eine Form übergeben, diese wird zunächst in eine weitere Form kopiert. Danach werden bei dieser Form die Achsen gewechselt und die x-Achse negiert[2]. Dies geschied für jeden Block in der Form. Danach wird die kopierte Funktion zurückgegeben.

gameboard.c

enthält die Funktionen zur Verwaltung des Gameboards (oder kurz *gb*). Beispielsweise überprüft die Funktion *gb_valid_pos*, ob an den gegebenen Koordinaten sich bereits ein Spielstein befindet. Dabei wird zuerst überprüft, ob sich die Koordinaten innerhalb des Spielfeldes befinden[3]. Ist dies gegeben, wird über das *blocks*-Array iteriert und alle Positionen verglichen. Ist auch hier kein bereits exisierender Stein im Weg wird *true* zurückgeliefert, in allen anderen Fällen *false*.
Die Funktion *gb_remove_completed_rows* hingegen überprüft nach jedem Hinzufügen von Steinen zum Gameboard, ob eine ganze Reihe gefüllt wurde. Ist dies der Fall, werden alle Steine in dieser Reihe entfernt und die darüberliegenden Reihen nach unten geshiftet. Dies erledigt die Funktion *gb_remove_row*.

[1] Verschieben oder drehen
[2] $x = y$, $y = -x$
[3] Ausser überhalb des sichtbaren Bereiches. Spielsteine können überhalb des Spielfeldes existieren.

6.1.2 Implementierung

Listing 6.1: main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define GLFW_INCLUDE_GLU
5
6  #include <GLFW/glfw3.h>
7  #include "gameboard.h"
8  #include "types.h"
9  #include "timer.h"
10 #include "try.h"
11 #include "random.h"
12
13 #define WIDTH  400
14 #define HEIGHT WIDTH * (GB_ROWS / GB_COLS)
15
16 static form current;
17
18 static bool game_over = false;
19
20 static void on_key(GLFWwindow *window, int key, int scancode, int action, int
       modifiers) {
21     UNUSED(window);
22     UNUSED(scancode);
23     UNUSED(modifiers);
24
25     if(game_over) return;
26     int dx = 0;
27     int dy = 0;
28     switch (key) {
29         case GLFW_KEY_DOWN:
30             dy = -1;
31             break;
32         case GLFW_KEY_LEFT:
33             dx = -1;
34             break;
35         case GLFW_KEY_RIGHT:
36             dx = +1;
37             break;
38         case GLFW_KEY_UP:
39             break;
40         default:
41             return;
42     }
43
44     if (action == GLFW_PRESS || action == GLFW_REPEAT) {
45         if (!try_move(dx, dy, &current)) {
46             if (dy == -1) {
47                 if (!gb_valid_pos(current.pos)) {
48                     game_over = true;
49                     return;
50                 }
51                 gb_add_form(current);
52                 //create new block
```

```
53                  current = random_form();
54              }
55          }
56          if (key == GLFW_KEY_UP)
57              try_rotate(&current);
58      }
59  }
60
61  void on_timer(void) {
62      on_key(NULL, GLFW_KEY_DOWN, 0, GLFW_PRESS, 0);
63  }
64
65  int main() {
66      if (!glfwInit()) {
67          fprintf(stderr, "could not initialize GLFW\n");
68          return EXIT_FAILURE;
69      }
70
71      GLFWwindow *const window = glfwCreateWindow(WIDTH, HEIGHT, "Tetris", NULL, NULL)
        ;
72      if (!window) {
73          glfwTerminate();
74          fprintf(stderr, "could not open window\n");
75          return EXIT_FAILURE;
76      }
77
78      int width, height;
79      glfwGetWindowSize(window, &width, &height);
80      glfwSetWindowAspectRatio(window, width, height);//enforce correct aspect ratio
81      glfwMakeContextCurrent(window);
82      glfwSetKeyCallback(window, on_key);
83
84      double timer_interval = 0.5;
85      timer_init(timer_interval, on_timer);
86      init_form_types();
87      current = random_form();
88
89      while (!glfwWindowShouldClose(window)) {
90          timer_test();
91          glfwGetFramebufferSize(window, &width, &height);
92          glViewport(0, 0, width, height);
93          glClear(GL_COLOR_BUFFER_BIT);//clear frame buffer
94          glMatrixMode(GL_PROJECTION);
95          glLoadIdentity();
96          gluOrtho2D(0, width, 0, height); //orthogonal projection -- origin is in lower-left
        corner
97          glScalef((float) width / (float) GB_COLS, (float) height / (float) GB_ROWS,
98                   1); //scale logical pixel to screen pixels
99
100         gb_render(); //render gameboard
101         render_form(current); //render current block
102
103         const GLenum error = glGetError();
104         if (error != GL_NO_ERROR) fprintf(stderr, "ERROR: %s\n", gluErrorString(
        error));
105
106
```

```
107         glfwSwapBuffers(window);//push image to display
108         glfwWaitEventsTimeout(timer_interval/5); //process all events of the application
109     }
110
111     glfwDestroyWindow(window);
112     glfwTerminate();
113     return EXIT_SUCCESS;
114 }
```

Listing 6.2: random.h

```
1  //
2  // Created by khp on 24.11.18.
3  //
4
5  #ifndef PROJECT_RANDOM_H
6  #define PROJECT_RANDOM_H
7
8  #include "types.h"
9
10 extern color random_color(void);
11 extern form random_form(void);
12
13 #endif //PROJECT_RANDOM_H
```

Listing 6.3: random.c

```
1  //
2  // Created by khp on 24.11.18.
3  //
4  #include <stdlib.h>
5  #include <time.h>
6
7  #include "random.h"
8  #include "types.h"
9  #include "gameboard.h"
10
11 color random_color(void) {
12     static color colors[] = {
13             color_red,
14             color_blue,
15             color_cyan,
16             color_green,
17             color_yellow,
18             color_magenta
19     };
20
21     int n_colors = sizeof(colors)/ sizeof(color);
22     srand(time(NULL));
23     return colors[rand()%n_colors];
24 }
25
26 form random_form(void) {
27     form t_form;
28     t_form.pos.x = GB_COLS / 2;
29     t_form.pos.y = GB_ROWS - 1;
30     t_form.color = random_color();
```

```
31     t_form.form_type = form_types[rand()%AMOUNT_FORMS];
32     return t_form;
33 }
```

Listing 6.4: timer.h

```
1  //
2  // Created by khp on 12.11.18.
3  //
4
5  #ifndef PROJECT_TIMER_H
6  #define PROJECT_TIMER_H
7
8  typedef void (*timer_func)(void);
9  extern void timer_init(double  timer_interval, timer_func on_timer);
10 extern void timer_test(void);
11 extern void timer_reset(void);
12
13 #endif //PROJECT_TIMER_H
```

Listing 6.5: timer.c

```
1  //
2  // Created by khp on 12.11.18.
3  //
4  #include <assert.h>
5  #include <stddef.h>
6  #include <GLFW/glfw3.h>
7  #include "timer.h"
8
9  #define TIMER_MULTIPLICATOR 0.99
10
11 static double timer_interval = 0;
12 static timer_func callback = NULL;
13
14 void timer_init(double  intv, timer_func on_timer){
15     timer_interval = intv;
16     callback = on_timer;
17     timer_reset();
18 }
19
20 void timer_test(void) {
21     assert(callback);
22
23     if(glfwGetTime() >= timer_interval) {
24         callback();
25         timer_interval *= TIMER_MULTIPLICATOR;
26         timer_reset();
27     }
28 }
29
30 void timer_reset(void) {
31     glfwSetTime(0.0);
32 }
```

Listing 6.6: try.h

```
1 //
2 // Created by khp on 24.11.18.
3 //
4
5 #ifndef PROJECT_TRY_H
6 #define PROJECT_TRY_H
7
8 #include "types.h"
9 #include <stdbool.h>
10
11 extern bool try_move(int , int , form*);
12 extern bool try_rotate(form*);
13 #endif //PROJECT_TRY_H
```

Listing 6.7: try.c

```
1 //
2 // Created by khp on 24.11.18.
3 //
4
5 #include "try.h"
6 #include "types.h"
7 #include "gameboard.h"
8
9 extern bool try_move(int dx, int dy, form *current) {
10     position future_pos = current->pos;
11     future_pos.x += dx;
12     future_pos.y += dy;
13     for (int i = 0; i < BLOCKS_IN_FORM; ++i) {
14         position t_pos = future_pos;
15         t_pos.x = future_pos.x + current->form_type.blocks[i].pos.x;
16         t_pos.y = future_pos.y + current->form_type.blocks[i].pos.y;
17         if (!gb_valid_pos(t_pos))
18             return false;
19     }
20     current->pos = future_pos;
21     return true;
22 }
23
24 extern bool try_rotate(form *current) {
25     form t_form = *current;
26     t_form.form_type = rotate_form_type(current->form_type);
27     for (int i = 0; i < BLOCKS_IN_FORM; ++i) {
28         position t_pos = t_form.pos;
29         t_pos.x = t_pos.x + t_form.form_type.blocks[i].pos.x;
30         t_pos.y = t_pos.y + t_form.form_type.blocks[i].pos.y;
31         if (!gb_valid_pos(t_pos))
32             return false;
33     }
34     *current = t_form;
35     return true;
36 }
```

Listing 6.8: types.h

```
1 #ifndef TYPES_H
2 #define TYPES_H
```

```
3
4  #include <assert.h>
5
6  #define BLOCKS_IN_FORM 4
7  #define UNUSED(var) ((void)var)
8  #define AMOUNT_FORMS 7
9
10 typedef enum {
11   color_black,
12   color_red     = 0x0000FFU,
13   color_green   = 0x00FF00U,
14   color_blue    = 0xFF0000U,
15   color_yellow  = color_red   | color_green,
16   color_magenta = color_red   | color_blue,
17   color_cyan    = color_green | color_blue,
18   color_white   = color_red   | color_green | color_blue,
19 } color;
20
21 typedef struct {
22   int x, y;
23 } position;
24
25 typedef struct {
26     position pos;
27     color color;
28 } block;
29
30 typedef struct {
31     block blocks[BLOCKS_IN_FORM];
32 } form_type;
33
34 typedef struct{
35     color color;
36     form_type form_type;
37     position pos;
38 } form;
39
40 form_type form_types[AMOUNT_FORMS];
41
42 extern void init_form_types(void);
43
44 extern form_type rotate_form_type(form_type);
45
46 extern void render_quad(const position pos, const color color);
47
48 extern void render_block(const block block);
49
50 extern void render_form(const form form);
51 #endif
```

Listing 6.9: types.c

```
1 #include <GLFW/glfw3.h>
2 #include "types.h"
3
4
5
```

```
6  void init_form_types() {
7    // I
8    form_types[0].blocks[0].pos.x = 0;
9    form_types[0].blocks[0].pos.y = -1;
10
11   form_types[0].blocks[1].pos.x = 0;
12   form_types[0].blocks[1].pos.y = 0;
13
14   form_types[0].blocks[2].pos.x = 0;
15   form_types[0].blocks[2].pos.y = 1;
16
17   form_types[0].blocks[3].pos.x = 0;
18   form_types[0].blocks[3].pos.y = 2;
19
20   // J
21   form_types[1].blocks[0].pos.x = 0;
22   form_types[1].blocks[0].pos.y = -1;
23
24   form_types[1].blocks[1].pos.x = 0;
25   form_types[1].blocks[1].pos.y = 0;
26
27   form_types[1].blocks[2].pos.x = 0;
28   form_types[1].blocks[2].pos.y = 1;
29
30   form_types[1].blocks[3].pos.x = -1;
31   form_types[1].blocks[3].pos.y = 1;
32
33     // L
34     form_types[2].blocks[0].pos.x = 0;
35     form_types[2].blocks[0].pos.y = -1;
36
37     form_types[2].blocks[1].pos.x = 0;
38     form_types[2].blocks[1].pos.y = 0;
39
40     form_types[2].blocks[2].pos.x = 0;
41     form_types[2].blocks[2].pos.y = 1;
42
43     form_types[2].blocks[3].pos.x = 1;
44     form_types[2].blocks[3].pos.y = 1;
45
46   // Z
47   form_types[3].blocks[0].pos.x = 0;
48   form_types[3].blocks[0].pos.y = -1;
49
50   form_types[3].blocks[1].pos.x = 1;
51   form_types[3].blocks[1].pos.y = -1;
52
53   form_types[3].blocks[2].pos.x = 1;
54   form_types[3].blocks[2].pos.y = 0;
55
56   form_types[3].blocks[3].pos.x = 0;
57   form_types[3].blocks[3].pos.y = -2;
58
59   // S
60   form_types[4].blocks[0].pos.x = 0;
61   form_types[4].blocks[0].pos.y = -1;
62
```

```
63    form_types[4].blocks[1].pos.x = 0;
64    form_types[4].blocks[1].pos.y = 0;
65
66    form_types[4].blocks[2].pos.x = 1;
67    form_types[4].blocks[2].pos.y = -1;
68
69    form_types[4].blocks[3].pos.x = 1;
70    form_types[4].blocks[3].pos.y = -2;
71
72    // T
73    form_types[5].blocks[0].pos.x = -1;
74    form_types[5].blocks[0].pos.y = 0;
75
76    form_types[5].blocks[1].pos.x = 0;
77    form_types[5].blocks[1].pos.y = 0;
78
79    form_types[5].blocks[2].pos.x = 1;
80    form_types[5].blocks[2].pos.y = 0;
81
82    form_types[5].blocks[3].pos.x = 0;
83    form_types[5].blocks[3].pos.y = 1;
84
85    // O
86    form_types[6].blocks[0].pos.x = 0;
87    form_types[6].blocks[0].pos.y = 0;
88
89    form_types[6].blocks[1].pos.x = 1;
90    form_types[6].blocks[1].pos.y = 0;
91
92    form_types[6].blocks[2].pos.x = 0;
93    form_types[6].blocks[2].pos.y = 1;
94
95    form_types[6].blocks[3].pos.x = 1;
96    form_types[6].blocks[3].pos.y = 1;
97  }
98
99  form_type rotate_form_type(form_type original_form) {
100     form_type res_form = original_form;
101     for (int i = 0; i < BLOCKS_IN_FORM; ++i) {
102         res_form.blocks[i].pos.y = -original_form.blocks[i].pos.x;
103         res_form.blocks[i].pos.x = original_form.blocks[i].pos.y;
104     }
105   return res_form;
106 }
107
108 void render_quad(const position pos, const color color) {
109   static_assert(sizeof(color) == 4, "detected unexpected size for colors");
110   glColor3ubv((unsigned char *)&color);
111   glBegin(GL_QUADS); {
112     glVertex2i(pos.x,     pos.y);
113     glVertex2i(pos.x,     pos.y + 1);
114     glVertex2i(pos.x + 1, pos.y + 1);
115     glVertex2i(pos.x + 1, pos.y);
116   } glEnd();
117 }
118
119 void render_block(const block block) {
```

```
120   render_quad(block.pos, block.color);
121 }
122
123 void render_form(const form form) {
124     for (int i = 0; i < BLOCKS_IN_FORM; ++i) {
125     block block;
126     block.color = form.color;
127     block.pos.x = form.pos.x + form.form_type.blocks[i].pos.x;
128     block.pos.y = form.pos.y + form.form_type.blocks[i].pos.y;
129         render_block(block);
130     }
131 }
```

Listing 6.10: gameboard.h

```
1  #ifndef GAMEBOARD_H
2  #define GAMEBOARD_H
3
4  #include <stdbool.h>
5  #include "types.h"
6
7  #define GB_ROWS 22
8  #define GB_COLS 11
9
10 bool gb_valid_pos(const position);
11 void gb_remove_row(int);
12 void gb_remove_completed_rows(void);
13 extern void gb_add_block(const block);
14 extern void gb_add_form(const form);
15 extern void gb_render(void);
16
17 #endif
```

Listing 6.11: gameboard.c

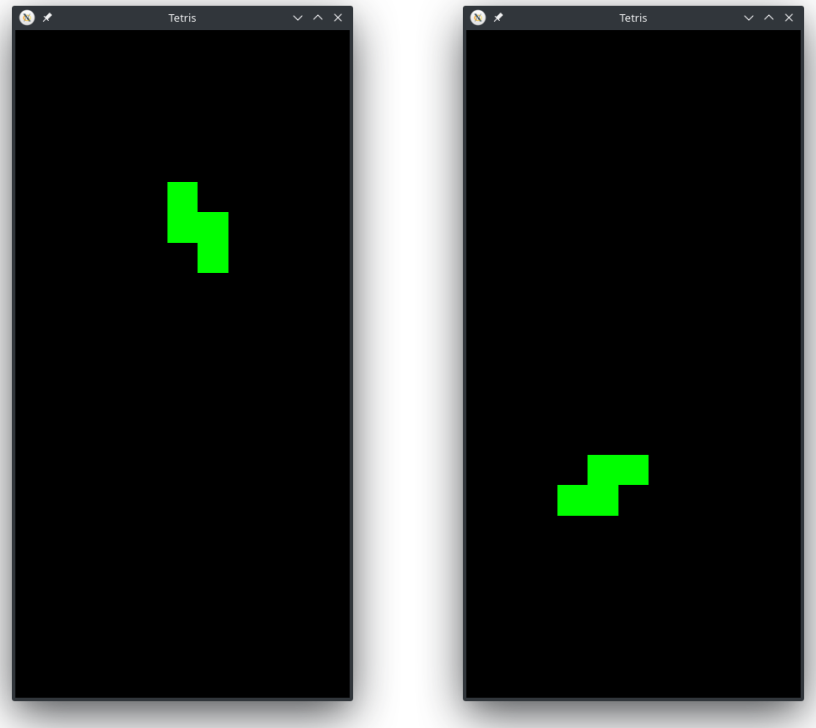
```
1  #include <stddef.h>
2  #include <assert.h>
3  #include <stdlib.h>
4  #include "gameboard.h"
5
6  #define MAX_BLOCK_COUNT 1000
7
8  static size_t block_count = 0;
9  static block blocks[MAX_BLOCK_COUNT];
10
11
12 bool gb_valid_pos(const position pos) {
13     if (!(0 <= pos.x && pos.x < GB_COLS &&
14         0 <= pos.y))
15         return false;
16     for (size_t i = 0; i < block_count; ++i) {
17         position p = blocks[i].pos;
18         if (p.x == pos.x && p.y == pos.y)
19             return false;
20     }
21     return true;
22 }
```

```
23
24 void gb_remove_row(int row) {
25     unsigned long int removedBlocks = 0;
26     for (size_t i = 0; i < block_count; ++i) {
27         if(blocks[i].pos.y == row) {
28             removedBlocks++;
29         } else {
30             if(blocks[i].pos.y > row) {
31                 blocks[i].pos.y--;
32             }
33             blocks[i - removedBlocks] = blocks[i];
34         }
35     }
36     block_count -= removedBlocks;
37 }
38
39 void gb_remove_completed_rows() {
40     int rowArr[GB_ROWS];
41     int removedRows = 0;
42     for (int j = 0; j < GB_ROWS; ++j) {
43         rowArr[j] = 0;
44     }
45     for (size_t i = 0; i < block_count; ++i) {
46         rowArr[blocks[i].pos.y] += 1;
47     }
48     for (int k = 0; k < GB_ROWS; ++k) {
49         if(rowArr[k] == GB_COLS) {
50             gb_remove_row(k-removedRows);
51             removedRows++;
52         }
53     }
54 }
55
56 void gb_add_block(const block block) {
57     assert(block_count < MAX_BLOCK_COUNT);
58     assert(gb_valid_pos(block.pos));
59     blocks[block_count++] = block;
60 }
61
62 void gb_add_form(const form form) {
63     for (int i = 0; i < BLOCKS_IN_FORM; ++i) {
64         block block;
65         block.color = form.color;
66         block.pos.x = form.pos.x + form.form_type.blocks[i].pos.x;
67         block.pos.y = form.pos.y + form.form_type.blocks[i].pos.y;
68         gb_add_block(block);
69     }
70     gb_remove_completed_rows();
71 }
72
73 void gb_render(void) {
74     for (size_t i = 0; i < block_count; ++i) {
75         render_block(blocks[i]);
76     }
77 }
```

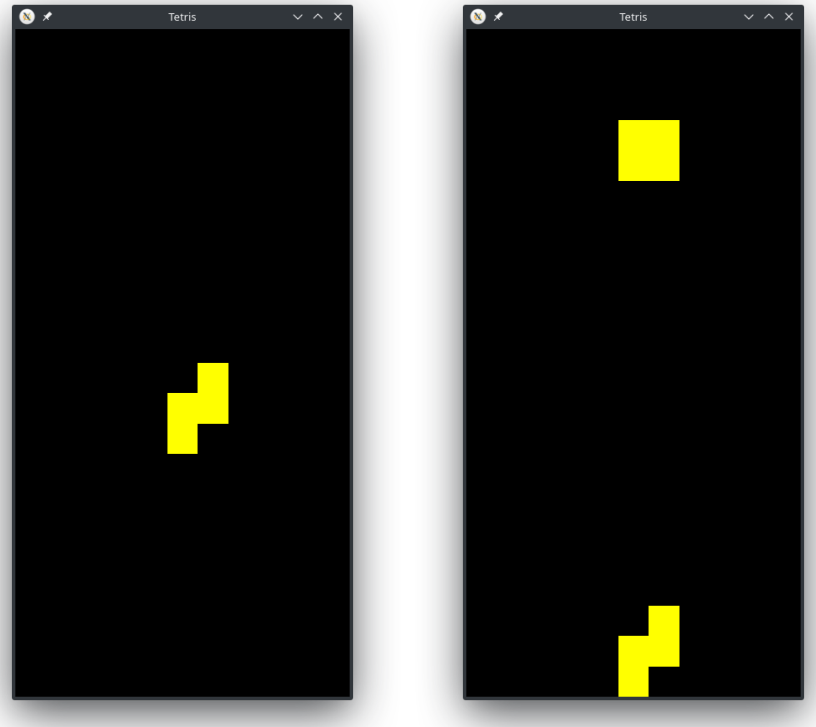
6.1.3 Testen

Durch die Gegebenheit, dass das Programm grafisch als Spiel aufgebaut ist, ist es schwer durch bildliche Darstellung die Funktionsfähigkeit zu demonstrieren. Daher wird empfohlen, selbst das kompilierte Programm zu spielen und so diese zu testen.

Bewegen und Drehen eines Steines im Fall



Setzen und erneute Generierung eines Steins



Löschung einer vollständigen Zeile