SWO3	Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3	WS 2019/20, ÜZ 6

☐ Gruppe M. Hava

☒ Gruppe J. Heinzelreiter Name: Neuhold Michael Aufwand [h]: 5

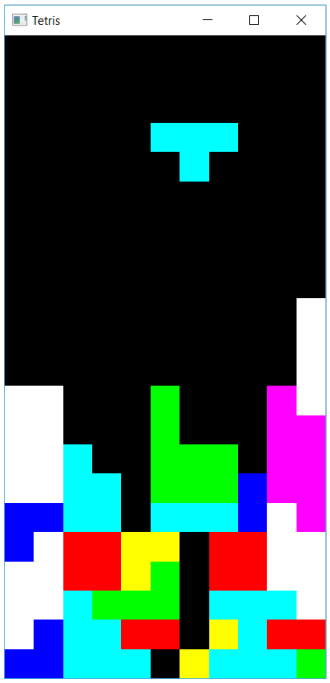
☐ Gruppe P. Kulczycki Feedback von: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (100 P)	100	85	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.



Inhaltsverzeichnis

1. GLFW-Applikation „Tetris“

1.1. Lösungsidee

Basierend auf dem vorbereiteten Projekt wurden diverse Funktionen zusätzlich implementiert, sodass man schlussendlich Tetris spielen kann. Für die sogenannten Tetrominos wurde eine neue Datenstruktur angelegt. Eine solche Datenstruktur beinhaltet ein Positionen-Array und eine Variable, in der die Farbe des Steines gespeichert wird. Im Feld werden die Positionen der 4 Blöcke gespeichert, aus denen ein Tetromino besteht. Vor dem Start der Schleife, im Main, wird ein Feld, welches alle möglichen Tetrominos (T,L,S,Z,O,I,J) beinhaltet initialisiert. Immer wenn ein Stein am Spielfeld fixiert wurde, so wird ein neuer Tetromino zufällig aus dem zuvor initialisierten Feld ausgewählt. Die fixierten Blöcke werden mit Hilfe eines 2-Dimensionalen Feldes abgebildet. Dabei stehen die Indizes für eine Koordinate eines Blockes.

Wichtig beim Anlegen eines Tetrominos ist es, dass der erste Eintrag im Positionen-Feld immer der Block ist, der den Mittelpunkt des Steines angibt.

Das Drehen der Tetrominos wurde mit einer Rotations-Matrix durchgeführt. Um eine solche Matrix auf einen Block anwenden zu können, muss dieser jedoch in den Ursprung verschoben werden. Im Anschluss kann die Rotationsmatrix angewandt werden. Zu guter Letzt muss der Tetromino wieder an die Ursprüngliche Position geschoben werden.

Immer wenn ein Tetromino fixiert wird, so wird überprüft ob eine Zeile vollständig mit Blöcken befüllt ist. Ist dies der Fall, so wird diese Gelöscht indem sie mit der Nachfolgenden überschrieben wird. Das Spiel ist beendet, wenn die erste Position, die ein Stein einnehmen würde bereits ungültig ist.

Um die Geschwindigkeit während des Spieles zu erhöhen, wird das Timer-Intervall veringert.

Hinweis: Aufgrund der Tatsache, dass ich auf einem MAC entwickle, wurde die GLFW Bibliothek Global auf meinem System mit dem Package Manager *brew* installiert.

1.2. Code

Types.h

```h
#ifndef TYPES_H
#define TYPES_H

#define NUMBER_OF_TETROMINOS 7

typedef enum {
   color_black,
   color_red     = 0x0000FFU,
   color_green   = 0x00FF00U,
   color_blue    = 0xFF0000U,
   color_orange  = 0x0095FFU,
   color_yellow  = color_red   | color_green,
   color_magenta = color_red   | color_blue,
   color_cyan    = color_green | color_blue,
   color_white   = color_red   | color_green | color_blue,
} color;

typedef struct {
   int x, y;
} position;

typedef struct {
    color color;
    position positions[4];
} tetromino;

tetromino tetromino_blocks[NUMBER_OF_TETROMINOS];

/*-------------------------------------------------------*/

void init_tetromino();
void render_tetromino(tetromino *curr_tetro);
void render_quad(const position * pos, const color * color);

/*-------------------------------------------------------*/

#endif
```

Types.c

```c
#include <assert.h>
#include <GLFW/glfw3.h>

#include "types.h"
#include "./gameboard.h"

/*-----------------------------------------------------------------------*/

void render_quad(const position * pos, const color * color) {
   assert(pos);
   assert(color);
   static_assert(sizeof(*color) == 4, "detected unexpected size for colors");
   glColor3ubv((unsigned char *)color);
   glBegin(GL_QUADS); {
      glVertex2f(pos->x + 0.025,     pos->y+ 0.025);
      glVertex2f(pos->x + 0.025,     pos->y + 0.975);
      glVertex2f(pos->x + + 0.975, pos->y + + 0.975);
      glVertex2f(pos->x + + 0.975, pos->y + 0.025);
   } glEnd();
}

/*-----------------------------------------------------------------------*/
```

```
void init_tetromino() {

    /* J */
    tetromino tetro1 = {
            color_blue,
            .positions[0] = { 2 , 2 },
            .positions[1] = { 1 , 1 },
            .positions[2] = { 2 , 1 },
            .positions[3] = { 2 , 3 }
    };

    /* O */
    tetromino tetro2 = {
            color_yellow,
            .positions[0] = { 2 , 2 },
            .positions[1] = { 1 , 1 },
            .positions[2] = { 1 , 2 },
            .positions[3] = { 2 , 1 }
    };

    /* S */
    tetromino tetro3 = {
            color_green,
            .positions[0] = { 1 , 0 },
            .positions[1] = { 0 , 0 },
            .positions[2] = { 1 , 1 },
            .positions[3] = { 2 , 1 }
    };

    /* Z */
    tetromino tetro4 = {
            color_red,
            .positions[0] = { 1 , 0 },
            .positions[1] = { 0 , 1 },
            .positions[2] = { 1 , 1 },
            .positions[3] = { 2 , 0 }
    };

    /* I */
    tetromino tetro5 = {
            color_cyan,
            .positions[0] = { 2 , 1 },
            .positions[1] = { 1 , 1 },
            .positions[2] = { 3 , 1 },
            .positions[3] = { 4 , 1 }
    };

    /* T */
    tetromino tetro6 = {
            color_magenta,
            .positions[0] = { 2 , 1 },
            .positions[1] = { 1 , 1 },
            .positions[2] = { 3 , 1 },
            .positions[3] = { 2 , 2 }
    };

    /* L */
    tetromino tetro7 = {
            color_orange,
            .positions[0] = { 5 , 11 },
            .positions[1] = { 5 , 10 },
            .positions[2] = { 5 , 12 },
            .positions[3] = { 6 , 10 }
    };

    tetromino_blocks[0] = tetro1;
    tetromino_blocks[1] = tetro2;
```

```
    tetromino_blocks[2] = tetro3;
    tetromino_blocks[3] = tetro4;
    tetromino_blocks[4] = tetro5;
    tetromino_blocks[5] = tetro6;
    tetromino_blocks[6] = tetro7;
}

/*------------------------------------------------------------------------*/

void render_tetromino(tetromino *curr_tetro) {
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        render_quad(&(curr_tetro -> positions[i]), &(curr_tetro -> color));
    }
}
```

Timer.h

```
#ifndef TIMER_H
#define TIMER_H

typedef void(*timer_callback)(void);

extern double timer_interval;

void timer_init(timer_callback func);
void timer_test(void);
void timer_reset(void);

#endif
```

Timer.c

```
#include <assert.h>
#include <GLFW/glfw3.h>
#include "timer.h"

double timer_interval = 1;

static
timer_callback callback;

void timer_init(timer_callback func) {
   assert(func);
   assert(!callback);
   callback = func;
   timer_reset();
}

void timer_test(void) {
   assert(callback);
   if (glfwGetTime() >= timer_interval) {
      callback();
      timer_reset();
   }
}

void timer_reset(void) {
   glfwSetTime(0);
}
```

Main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define GLFW_INCLUDE_GLU
#include <GLFW/glfw3.h>
#include "gameboard.h"
#include "timer.h"

#define WIDTH  400
#define HEIGHT WIDTH * (GB_ROWS / GB_COLS)
#define SPEED_STEP 0.02

#define BLOCK_SELECT rand() % 7

// select specific tetromino for testing
// #define BLOCK_SELECT 0

static bool game_over;
tetromino current;

/*-------------------------------------------------------------------------*/

static bool try_move(int dx, int dy) {
    bool valid = true;
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        position pos = current.positions[i];
        pos.x += dx;
        pos.y += dy;
        if (!gb_is_valid_position(&pos)) valid = false;
    }
    if(valid) {
        for(int j = 0; j < NUMBER_OF_BLOCKS; j++) {
            current.positions[j].x += dx;
            current.positions[j].y += dy;
        }
    }
   return valid;
}

/*-------------------------------------------------------------------------*/

static int current_max_y() {
    int max = 0;
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        if(current.positions[i].y > max) max = current.positions[i].y;
    }
    return max;
}

/*-------------------------------------------------------------------------*/

static int current_max_x() {
    int max = 0;
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        if(current.positions[i].x > max) max = current.positions[i].x;
    }
    return max;
}

/*-------------------------------------------------------------------------*/

static void set_current_to_top() {
    int offset_y = GB_ROWS - current_max_y();
    int offset_x = (GB_COLS / 2) - current_max_x();
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        current.positions[i].y += offset_y - 1;
```

```
        current.positions[i].x += offset_x + 1;
    }
}

/*----------------------------------------------------------------------*/

static int line_is_full() {
    for (int i = 0; i < GB_ROWS; i++) {
        bool full = true;
        for(int j = 0; j < GB_COLS; j++) {
            if(!fixed_blocks[j][i].is_set) {
                full = false;
            }
        }
        if(full) {
            printf("line is full\n");
            return i;
        }
    }
    return -1;
}

/*----------------------------------------------------------------------*/

static void clear_line(int line_nr) {
    for (int i = line_nr + 1; i < GB_ROWS; i++) {
        for(int j = 0; j < GB_COLS; j++) {
            fixed_blocks[j][i-1].is_set = fixed_blocks[j][i].is_set;
            fixed_blocks[j][i-1].block_color = fixed_blocks[j][i].block_color;
        }
    }
    for(int i = 0; i < GB_COLS; i++) {
        fixed_blocks[i][GB_ROWS - 1].is_set = false;
    }
}

/*----------------------------------------------------------------------*/

static void rotate_tetro() {

    tetromino tmp_tetromino = current;
    bool valid = true;

    // move to origin
    for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {

        // move to origin
        tmp_tetromino.positions[i].x -= current.positions[0].x;
        tmp_tetromino.positions[i].y -= current.positions[0].y;

        // rotation matrix
        int new_x = 0 * tmp_tetromino.positions[i].x + 1 *
tmp_tetromino.positions[i].y;
        int new_y = -1 * tmp_tetromino.positions[i].x + 0 *
tmp_tetromino.positions[i].y;
        tmp_tetromino.positions[i].x = new_x;
        tmp_tetromino.positions[i].y = new_y;

        // move to correct position
        tmp_tetromino.positions[i].x += current.positions[0].x;
        tmp_tetromino.positions[i].y += current.positions[0].y;
        if (!gb_is_valid_position(&tmp_tetromino.positions[i])) valid = false;
    }

    if(valid) current = tmp_tetromino;
}

/*----------------------------------------------------------------------*/
```

```
static void on_key(GLFWwindow* window, int key, int scancode, int action, int
modifiers) {
   bool rotate = false;
   if (game_over) return;
   int dx = 0, dy = 0;
   switch (key) {
      case GLFW_KEY_DOWN:     dy = -1;    break;
      case GLFW_KEY_UP:
          if (action == GLFW_PRESS || action == GLFW_REPEAT) { rotate_tetro();
rotate = true; }
          break;
      case GLFW_KEY_LEFT:     dx = -1;    break;
      case GLFW_KEY_RIGHT:    dx = +1;    break;
      default: return; //ignore other keys
   }

   if ((action == GLFW_PRESS || action == GLFW_REPEAT) && !rotate) {
         if (!try_move(dx, dy)) { //why couldnt we move
            if (dy == -1) { //tried to move down => hit bottom!
                gb_update(current);

                /*******************************************/

                // check if line is full
                int line_nr = line_is_full();
                while(line_nr != -1) {
                    clear_line(line_nr);
                    line_nr = line_is_full();
                }
                // update current with new random block
                current = tetromino_blocks[BLOCK_SELECT];
                set_current_to_top();
                // increment speed
                timer_interval -= SPEED_STEP;
                printf("current speed: %f\n", timer_interval);

                /*******************************************/

                if (!gb_is_valid_position(current.positions)) game_over =
true;
                timer_reset();
            }
        }
   }
}

/*------------------------------------------------------------------------*/

// simulate keypress
static void on_timer(void) {
    on_key(NULL, GLFW_KEY_DOWN, 0, GLFW_PRESS, 0);
}

int main() {
   if(!glfwInit()) {
      fprintf(stderr, "could not initialize GLFW\n");
      return EXIT_FAILURE;
   }

    /*******************************************/
   // init different tetromino blocks
    init_tetromino();
   // init fixed blocks
   init_fixed_blocks();
    /*******************************************/

   GLFWwindow * const window = glfwCreateWindow(WIDTH, HEIGHT, "Tetris", NULL,
```

```
NULL);
   if(!window) {
      glfwTerminate();
      fprintf(stderr, "could not open window\n");
      return EXIT_FAILURE;
   }

   int width, height;
   glfwGetWindowSize(window, &width, &height);
   glfwSetWindowAspectRatio(window, width, height);//enforce correct aspect
ratio
   glfwMakeContextCurrent(window);
   glfwSetKeyCallback(window, &on_key);
   timer_init(&on_timer);

   /*******************************************/
   // set first random tetromino
    current = tetromino_blocks[BLOCK_SELECT];
    // set current to top
    set_current_to_top();
    /*******************************************/

   while(!glfwWindowShouldClose(window)) {
      timer_test();

      glfwGetFramebufferSize(window, &width, &height);
      glViewport(0, 0, width, height);
      glClear(GL_COLOR_BUFFER_BIT);//clear frame buffer
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0, width, 0, height);//orthogonal projection – origin is in
lower-left corner
      glScalef((float)width / (float)GB_COLS, (float)height / (float)GB_ROWS,
1);//scale logical pixel to screen pixels

        /*******************************************/
        // render current tetromino
        render_tetromino(&current);
      // render fixed blocks at the bottom
      gb_render();
        /*******************************************/

      const GLenum error = glGetError();
      if(error != GL_NO_ERROR) fprintf(stderr, "ERROR: %s\n",
gluErrorString(error));
      glfwSwapBuffers(window);//push image to display
      glfwWaitEventsTimeout(timer_interval); // sleep
   }

   glfwDestroyWindow(window);
   glfwTerminate();
   return EXIT_SUCCESS;
}
```

Gameboard.h

```
#ifndef GAMEBOARD_H
#define GAMEBOARD_H

#include <stdbool.h>
#include "types.h"

#define GB_ROWS 22
#define GB_COLS 11
#define NUMBER_OF_BLOCKS 4
```

```
typedef struct {
    bool is_set;
    color block_color;
} fixed;

/*-----------------------------------------------------*/

fixed fixed_blocks[GB_COLS][GB_ROWS];
bool gb_is_valid_position(const position* pos);
void gb_update(tetromino tetro);
void gb_render(void);
void init_fixed_blocks();

/*-----------------------------------------------------*/

#endif
```

Gameboard.c

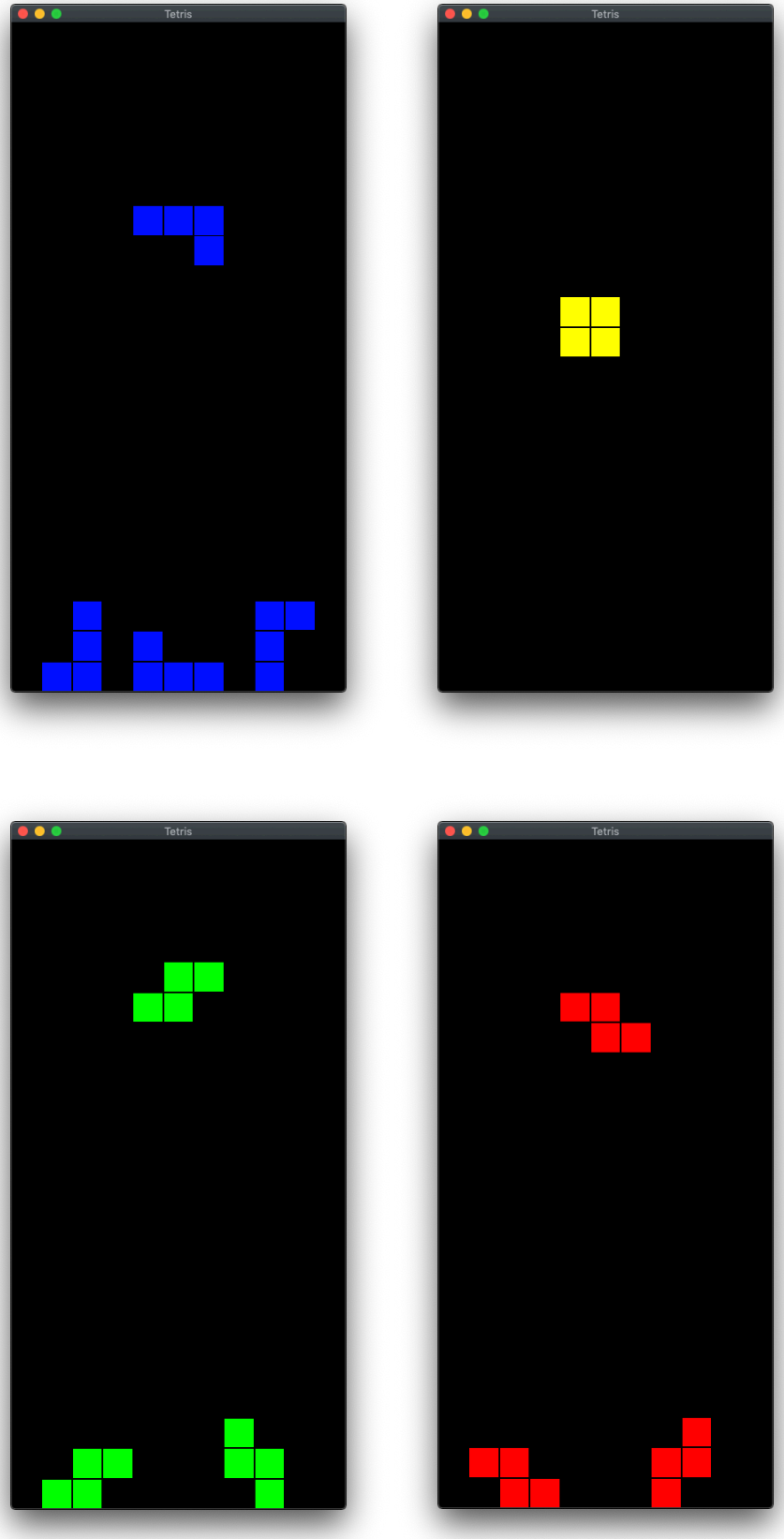
```
#include <assert.h>
#include <stdbool.h>
#include "gameboard.h"
#include <stdio.h>

/*-------------------------------------------------------------------------*/

void init_fixed_blocks() {
    for (int i = 0; i < GB_ROWS; i++) {
        for(int j = 0; j < GB_COLS; j++) {
            fixed_blocks[j][i].is_set = false;
        }
    }
}

/*-------------------------------------------------------------------------*/

bool gb_is_valid_position(const position* pos) {
   assert(pos);
   if (pos-> x < 0 || pos-> y < 0 || pos-> x >= GB_COLS || pos -> y >=
GB_ROWS) return false;
    for (int i = 0; i < GB_ROWS; i++) {
        for(int j = 0; j < GB_COLS; j++) {
            if(fixed_blocks[j][i].is_set && pos->x == j && pos->y == i) {
                return false;
            }
        }
    }
   return true;
}

/*-------------------------------------------------------------------------*/

void gb_update(tetromino tetro) {
   for(int i = 0; i < NUMBER_OF_BLOCKS; i++) {
        fixed_blocks[tetro.positions[i].x][tetro.positions[i].y].is_set =
true;
       fixed_blocks[tetro.positions[i].x][tetro.positions[i].y].block_color =
tetro.color;
   }
}
```

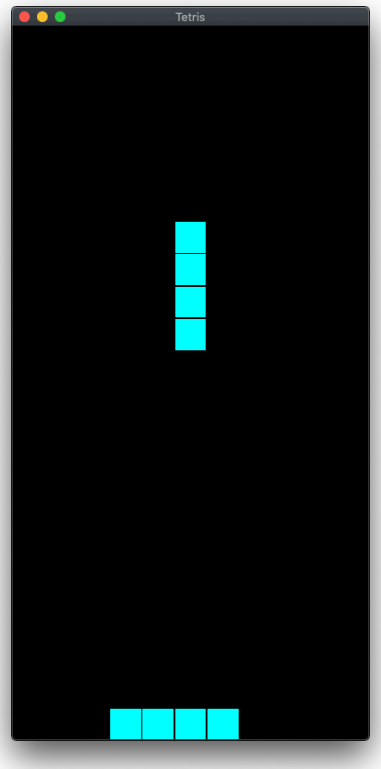
```
/*------------------------------------------------------------------------*/

void gb_render(void) {
   for (int i = 0; i < GB_ROWS; i++) {
       for(int j = 0; j < GB_COLS; j++) {
           position pos;
           pos.x = j;
           pos.y = i;
           if(fixed_blocks[j][i].is_set) {
               render_quad(&pos, &fixed_blocks[j][i].block_color);
           }
       }
   }
}

/*------------------------------------------------------------------------*/
```

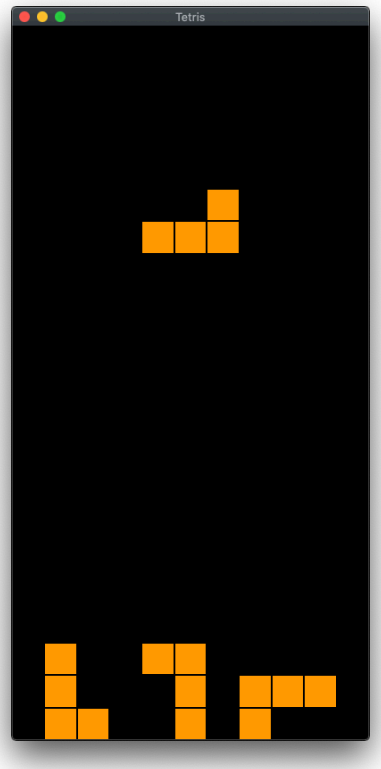
1.3. Testfälle

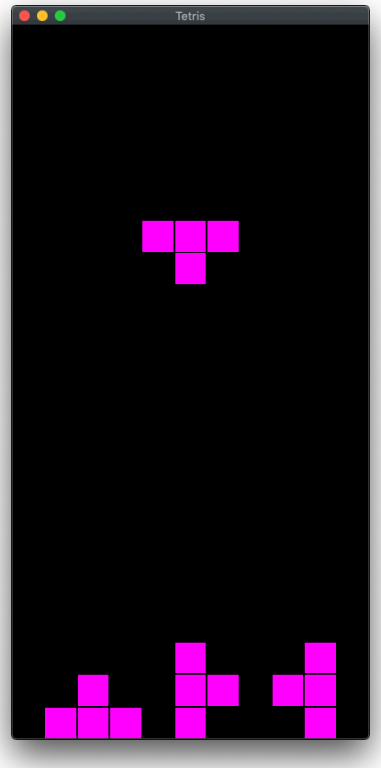
1.3.1. Tetromino Rotations

Die folgenden Screenshots zeigen alle Tetrominos rotiert in alle Richtungen.

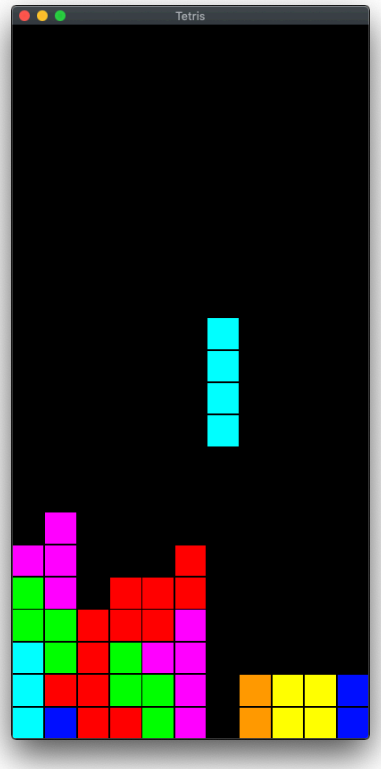



Tetris


Tetris

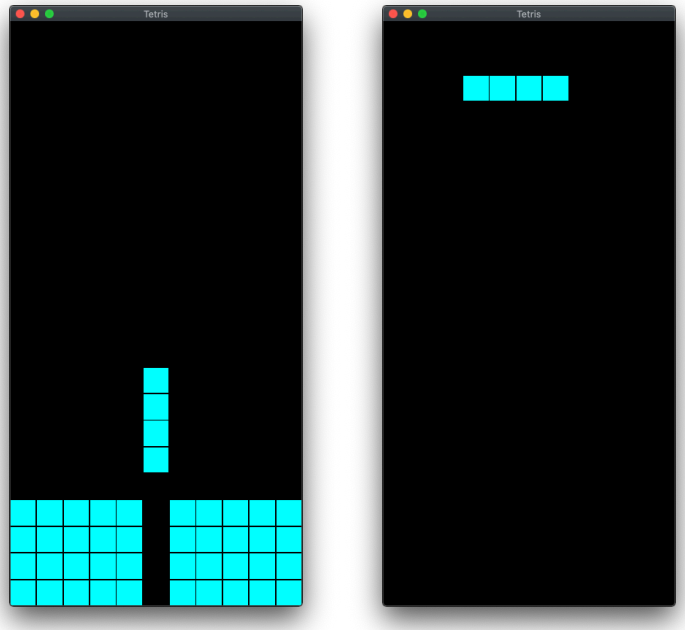

Tetris

1.3.2. Stapeln von Tetrominos



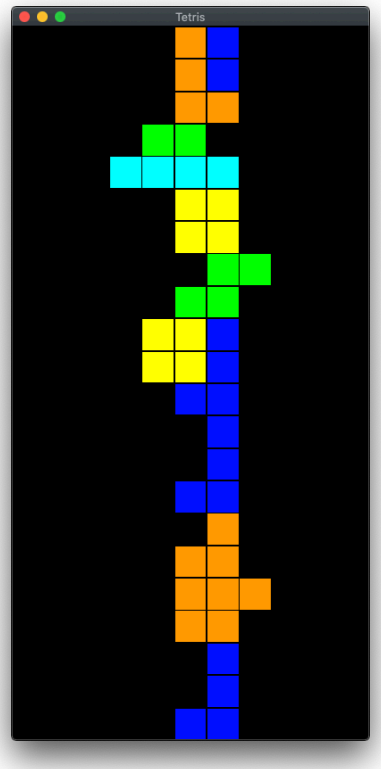
1.3.3. Löschen von Zeilen

Dieser Testfall wurde nur generiert um zu zeigen, dass mehrere Zeilen gleichzeitig gelöscht werden können.



1.3.4. Game Over

Wenn die erste Position eines neuen Steins bereits nicht valid ist, so ist das Spiel beendet und es werden keine neuen Steine mehr generiert.



1.3.5. Speed

Der Speed wird nach jedem Block um 0.02 erhöht bzw. das Timer-Interval um 0.02 gesenkt.

```
current speed: 0.980000
current speed: 0.960000
current speed: 0.940000
current speed: 0.920000
current speed: 0.900000
current speed: 0.880000
current speed: 0.860000
current speed: 0.840000
current speed: 0.820000
current speed: 0.800000
current speed: 0.780000
```