Inhaltsverzeichnis

[1. GLFW-Applikation „Tetris“ 2](#_Toc25527990)

[1.1. Lösungsidee 2](#_Toc25527991)

[1.2. Code 3](#_Toc25527992)

[1.3. Testfälle 12](#_Toc25527993)

[1.3.1. Tetromino Rotations 12](#_Toc25527994)

[1.3.2. Stapeln von Tetrominos 14](#_Toc25527995)

[1.3.3. Löschen von Zeilen 14](#_Toc25527996)

[1.3.4. Game Over 15](#_Toc25527997)

[1.3.5. Speed 15](#_Toc25527998)

# GLFW-Applikation „Tetris“

## 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.

## Code

|  |
| --- |
| Types.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 |
| #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 |
| #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**; } |

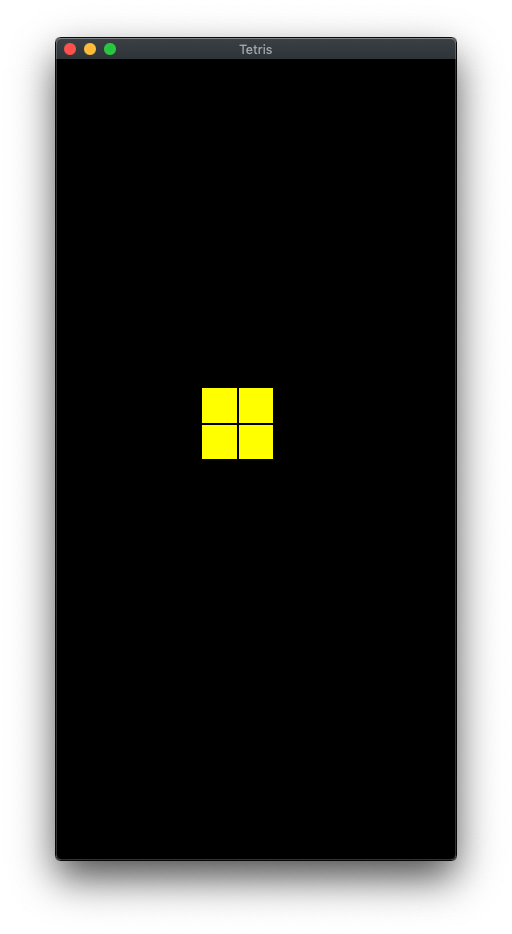
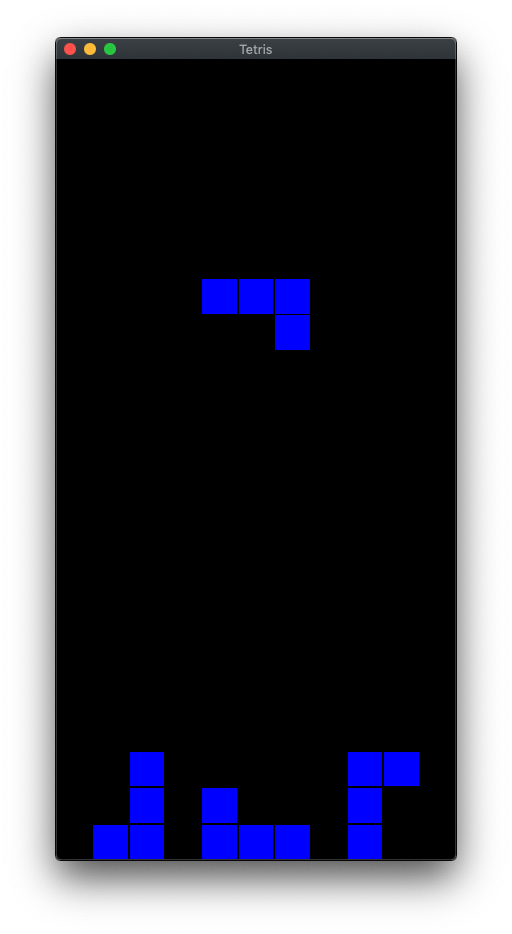
|  |
| --- |
| Gameboard.h |
| #ifndef **GAMEBOARD\_H** #define **GAMEBOARD\_H** #include **<stdbool.h>** #include **"types.h"** #define **GB\_ROWS** 22 #define **GB\_COLS** 11 #define **NUMBER\_OF\_BLOCKS** 4  **typedef struct** {  **bool** is\_set;  color block\_color; } fixed;  */\*----------------------------------------------------\*/* 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 |

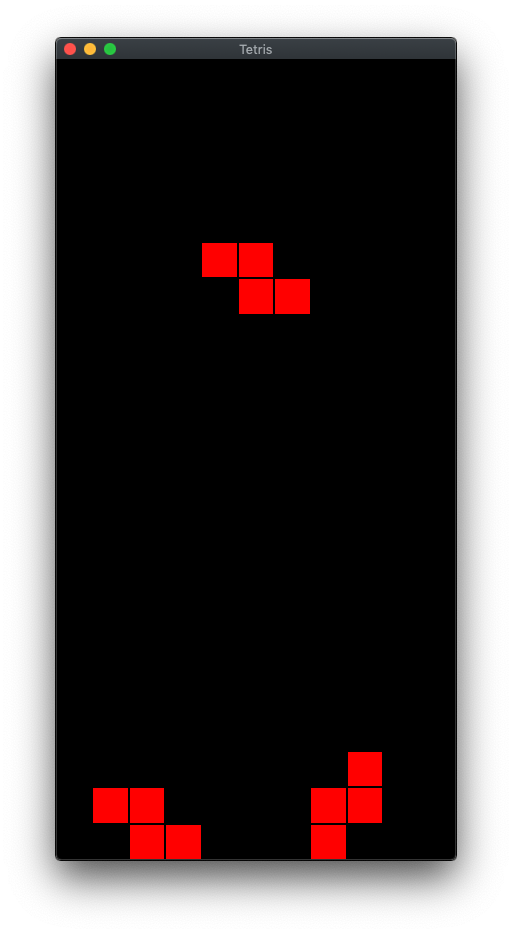
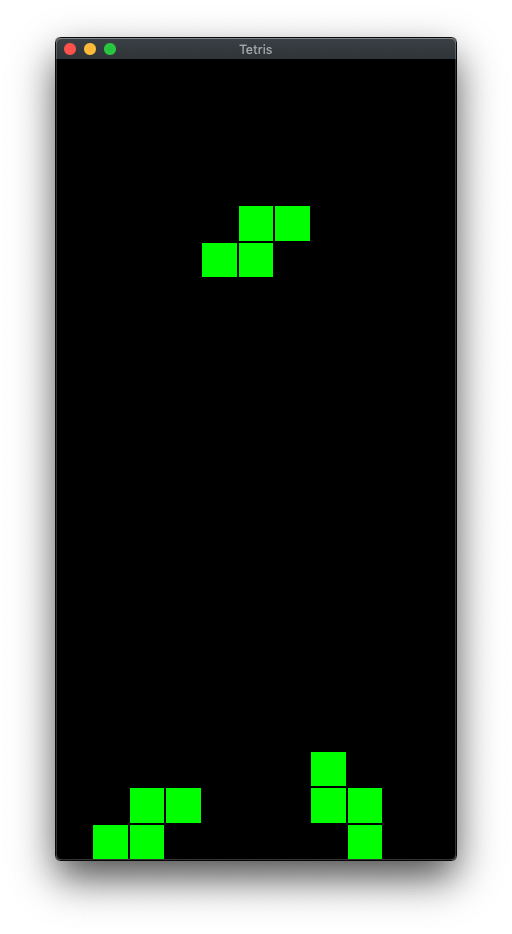
|  |
| --- |
| Gameboard.c |
| #include **<assert.h>** #include **<stdbool.h>** #include **"gameboard.h"** #include **<stdio.h>** */\*---------------------------------------------------------------------\*/* **void** init\_fixed\_blocks() {  **for** (**int** i = 0; i < **GB\_ROWS**; i++) {  **for**(**int** 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);  }  }  } }  */\*---------------------------------------------------------------------\*/* |

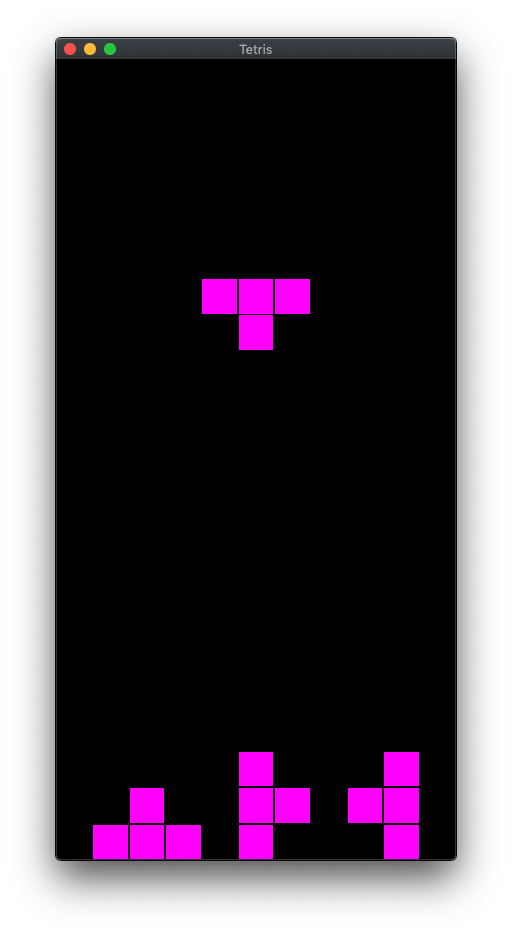
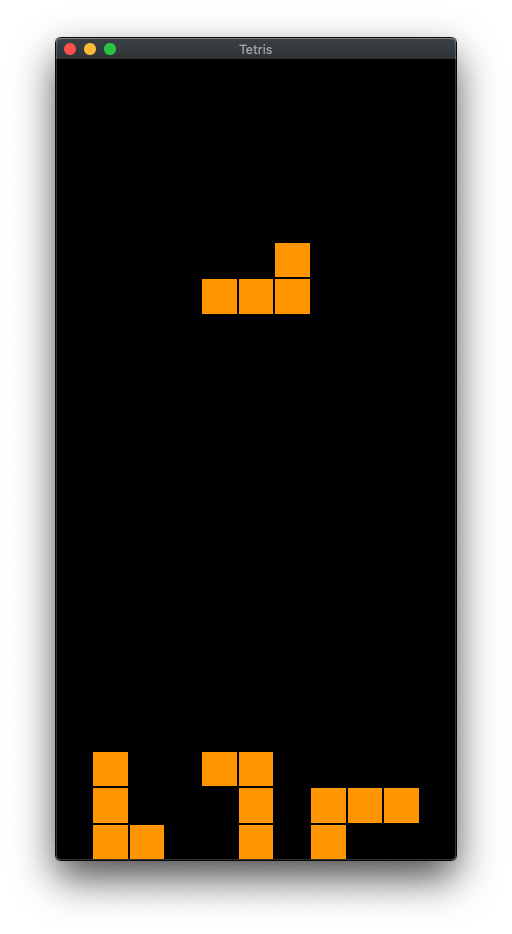
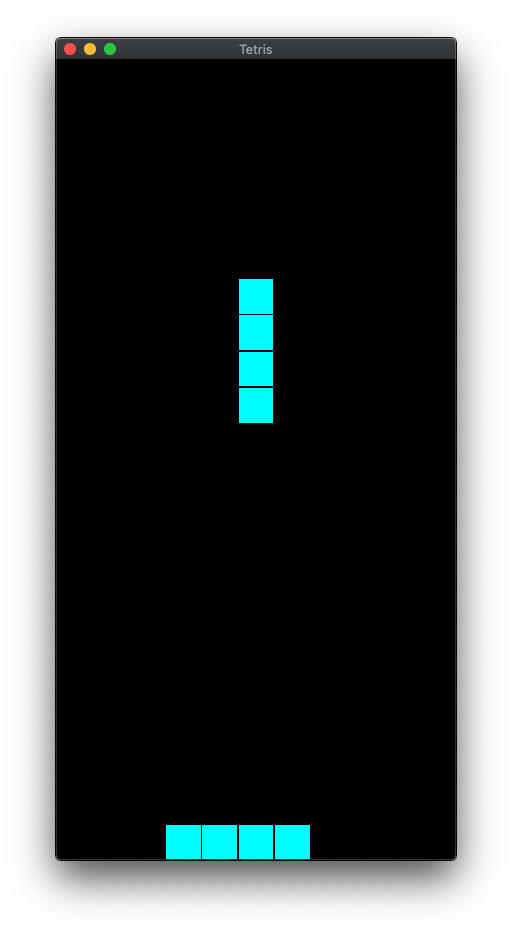
## Testfälle

### Tetromino Rotations

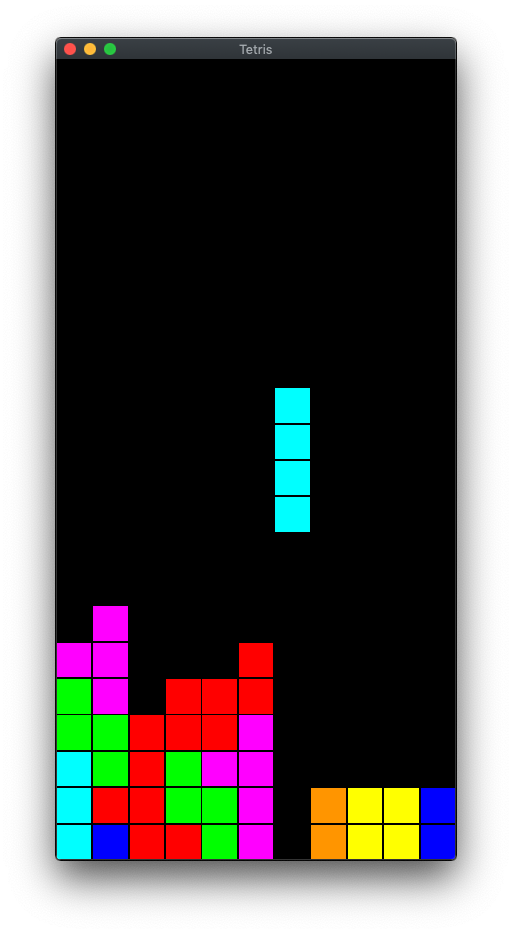
Die folgenden Screenshots zeigen alle Tetrominos rotiert in alle Richtungen.



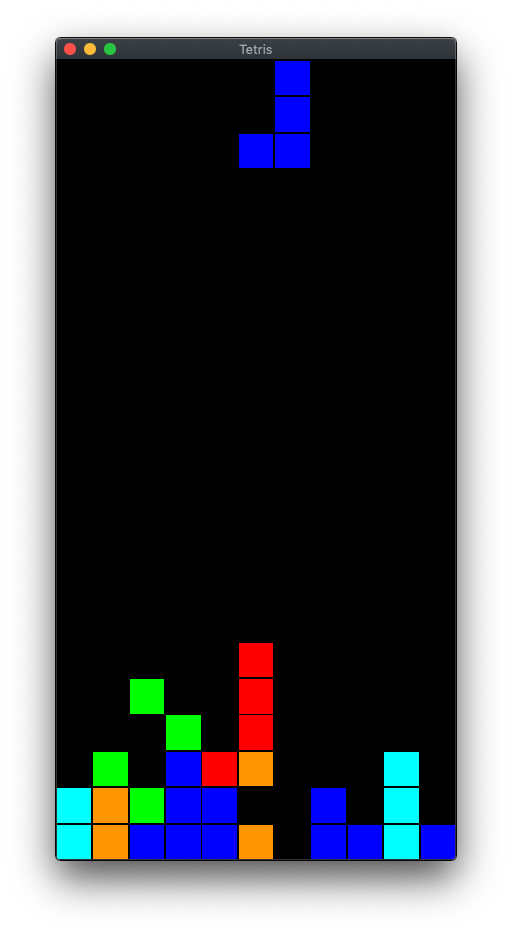




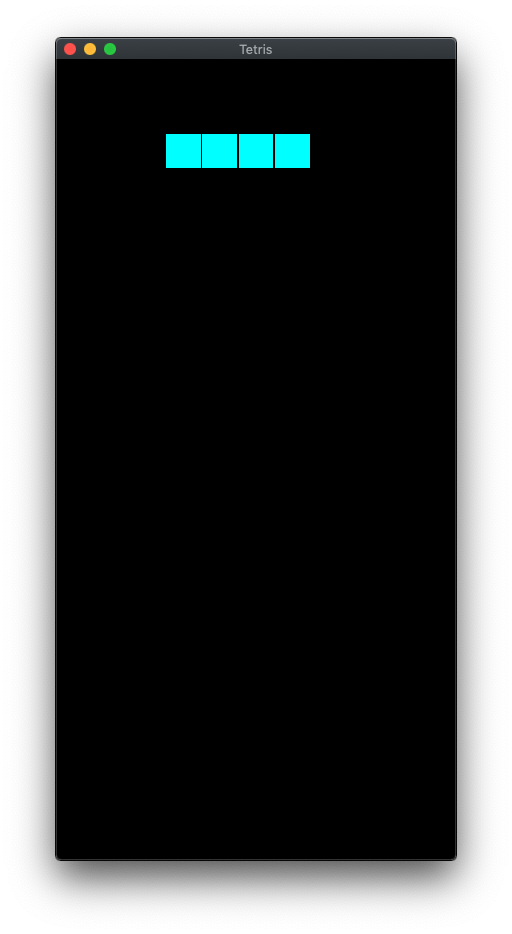
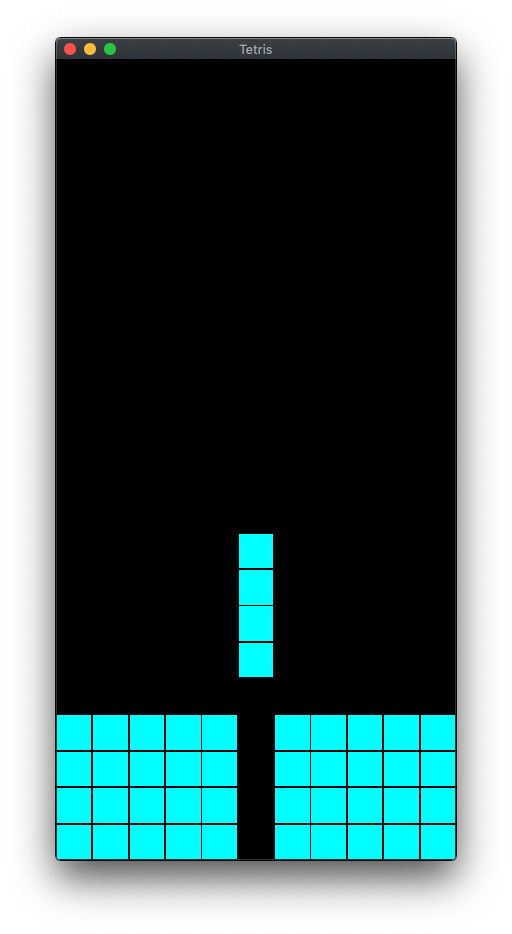
### Stapeln von Tetrominos



### Löschen von Zeilen

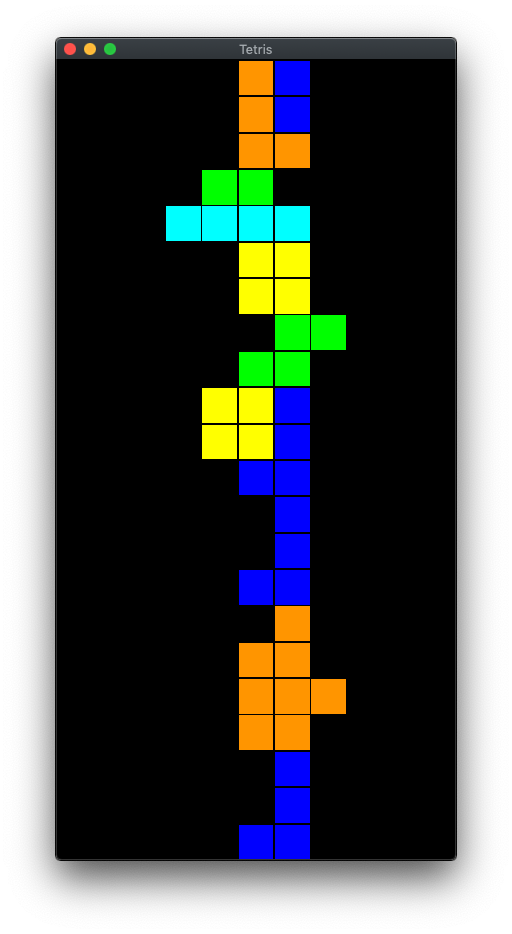


Dieser Testfall wurde nur generiert um zu zeigen, dass mehrere Zeilen gleichzeitig gelöscht werden können.



### 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.



### Speed

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

