HAND IN MODULE 2

MARIUS HAAKONSEN, OLE K LARSEN

1. Task 1:

Exercise A:
The both of us have already played Tetris, so we'll skip this part.

Exercise B:

Implementing the function void Board::reduce() to remove the lines when completed.

Looping over row number i, from top to bottom.

```
void Board::reduce() {
    for(int i = 3; i < 19; i++) {
```

Defining variables to use while looping over j number of columns.

```
        int count = 0;
        int tilecount = 0;
        for(int j = 1; j < 11; j++) {
            if (tiles[j][i] != sf::Color::Black) {
                count++;
```

If all tiles in row number 'i' is inequal to the color black, the program loops from that row and upwards, setting the current row to be equal to the row above, giving the impression that the rows "falls down".

```
            if(count == 10) {
                tilecount = i;
                for(int k = tilecount; k >= 3; k--) {
                    for(int j = 1; j < 11; j++) {
                        tiles[j][k] = tiles[j][k-1];
                    }
                }
                break;
            }
```

As a little coding-exercise from out part, we also added the functionality of pressing spacebar to make the shapes move all the way down until it intersects with other shapes. We also added a score.

2. Task 2:

Exercise A:

How was this puzzle created?
Puzzling.stackexchange.com was utilized to get the correct specifications of the puzzle and inspiration for the statements to be made by the three people in the encounter.

Knight: Always tells the truth.
Knave: Always tells a lie.
Spy: Tells either the truth or a lie.

The puzzle involves encountering three different people, person A, B and C.

They all have their own statements:

A: "C is the knight. hot dog"
B: "A is the knight."
C: "B is the knave."

Who is the knight, who is the knave, and who is the spy among the three?

To reach the solution of this, the statements are put in a table.

Knight, Knave and Spy			
Combinations A B C	Is it true? A B C	Would it be said? A B C	Solution A B C
0 1 x	0 0 0	1 0 x	
1 0 x	0 1 1	0 0 x	
0 x 1	1 0 0	0 x 0	
1 x 1	0 1 0	0 x 1	
x 0 1	1 0 1	x 1 1	1
x 1 0	0 0 0	x 0 0	

The solution is found by asking the two following questions: 'Is it true?' and 'Would it be said?'.

Based on the combinations of these columns, and based on whether or not the person actually tells the truth or lies, we find the correct combination of knight knave and spy. The next to last row is the only possible solution, as shown by the 'x 1 1' outcome. This means that C is the knight, B is the knace and A is the spy.

Exercise B:
Reformulated statements, giving the same answer:

A: "C may tell the truth."
B: "C is not a knight."
C: "B may tell a lie."

Knight, Knave and Spy			
Combinations A B C	Is it true? A B C	Would it be said? A B C	Solution A B C
0 1 x	1 1 0	0 1 x	
1 0 x	1 1 1	1 0 x	
0 x 1	1 0 1	0 x 1	
1 x 1	0 1 1	0 x 0	
x 0 1	1 0 1	x 1 1	1
x 1 0	0 1 0	x 1 0	

3. **Code Appendix:**

Task 1:

```
class Shape {
public:
    sf::Color tiles[4][4];
    sf::Vector2i pos;
    // times since last downward movement
    float time;

    Shape();
    // reinitialise the shape: move to top and change shape
    void init();
    // move downwards once per second
    void update(float dt);
    // render the shape
    void draw(sf::RenderWindow& w);
    // rotate the shape
    void rotateLeft();
    void rotateRight();
};

void Shape::rotateLeft() {
    sf::Color tmp[4][4];
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            tmp[i][j]=tiles[j][3-i];
        }
    }
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            tiles[i][j]=tmp[i][j];
        }
    }
}

void Shape::rotateRight() {
    rotateLeft();
```

```
    rotateLeft();
    rotateLeft();
}

Shape::Shape() {
    init();
}

void Shape::draw(sf::RenderWindow& w) {
    sf::CircleShape s;
    s.setRadius(8);
    s.setOrigin(8,8);
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            if(tiles[i][j] != sf::Color::Black) {
                s.setFillColor(tiles[i][j]);
                s.setPosition(sf::Vector2f(pos.x * 16 + 16 * i + 100, pos.y * 16 + 16 * j + 100
                w.draw(s);
            }
        }
    }
}

void Shape::update(float dt) {
    time += dt;
    if(time > 1) {
        time = 0;
        pos.y += 1;
    }
}

void Shape::init() {
    // move to top and reset timer
    pos.y = 0;
    pos.x = 4;
    time = 0.0f;

    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            tiles[i][j] = sf::Color::Black;
        }
    }
    switch(rand() % 8) {
        case 0:
            tiles[1][1] = sf::Color::Yellow;
            tiles[1][2] = sf::Color::Yellow;
            tiles[2][1] = sf::Color::Yellow;
            tiles[2][2] = sf::Color::Yellow;
            break;
        case 1:
```

```
            tiles[2][0] = sf::Color(255,160,0);
            tiles[2][1] = sf::Color(255,160,0);
            tiles[2][2] = sf::Color(255,160,0);
            tiles[2][3] = sf::Color(255,160,0);
            break;
        case 2:
            tiles[0][2] = sf::Color::Blue;
            tiles[1][2] = sf::Color::Blue;
            tiles[1][1] = sf::Color::Blue;
            tiles[2][1] = sf::Color::Blue;
            break;
        case 3:
            tiles[0][2] = sf::Color::Green;
            tiles[1][2] = sf::Color::Green;
            tiles[2][2] = sf::Color::Green;
            tiles[1][1] = sf::Color::Green;
            break;
        case 4:
            tiles[2][3] = sf::Color::White;
            tiles[2][2] = sf::Color::White;
            tiles[1][2] = sf::Color::White;
            tiles[0][2] = sf::Color::White;
            break;
        case 5:
            tiles[2][3] = sf::Color(4,235,250);
            tiles[2][2] = sf::Color(4,235,250);
            tiles[1][2] = sf::Color(4,235,250);
            tiles[0][2] = sf::Color(4,235,250);
            break;
        case 6:
            tiles[2][1] = sf::Color(255,0,255);
            tiles[2][2] = sf::Color(255,0,255);
            tiles[1][2] = sf::Color(255,0,255);
            tiles[0][2] = sf::Color(255,0,255);
            break;
        case 7: {
            int x=rand() % 4; int y = rand() % 4;
            tiles[x][y] = sf::Color(75,0,150);
            x=rand() % 4; y = rand() % 4;
            tiles[x][y] = sf::Color(75,0,150);
            x=rand() % 4; y = rand() % 4;
            tiles[x][y] = sf::Color(75,0,150);
            x=rand() % 4; y = rand() % 4;
            tiles[x][y] = sf::Color(75,0,150);
            break;
        }
    }
}
```

‘hallo‘
test