Abgabe 01: Asteroids

Table of Contents

Lösungsidee

Code

Listing 1. flying_object.h

```
#pragma once
#include <ml5/ml5.h>

namespace asteroids {

    enum class rotate_direction {
        right, left
    };

    constexpr int turn_factor = 10;
    constexpr int full_degree = 360;


    class flying_object {
    public:

        using context_t = ml5::paint_event::context_t;

        explicit flying_object(wxRealPoint pos) : position_{ pos } {}

        /**
         * Rotate the object
         */
        virtual void rotate(rotate_direction const dir) {
            this->direction_ += dir == rotate_direction::right ? turn_factor : -
turn_factor;
            if (this->direction_ < 0) {
                this->direction_ = full_degree + this->direction_;
            } else if (this->direction_ >= full_degree) {
                this->direction_ = this->direction_ - full_degree;
            }
        }

        /**
         * Custom function for every child object
         */
```

```
        virtual void draw(context_t& ctx) = 0;

        /*
         * Move the object in a linear way, should be enough for most objects.
         */
        virtual void move() {
            this->position_.x += cos(to_radiant()) * this->speed_;
            this->position_.y += sin(to_radiant()) * this->speed_;
        }

        /**
         * Check if this object had an collision with another flying object
         */
        [[nodiscard]] bool has_collision(const flying_object &other) const {
            auto own_shape = this->create_transformed_shape_with_offset();
            wxRegion own_region(own_shape.size(), &own_shape[0]);

            auto other_shape = other.create_transformed_shape_with_offset();
            wxRegion const other_region(other_shape.size(), &other_shape[0]);

            own_region.Intersect(other_region);
            return !own_region.IsEmpty();
        }

    protected:
        wxRealPoint position_;
        int direction_ = 0; // The angle of the object in degree
        double speed_ = 0; // Default flies with zero percent

        /**
         * Convert the direction to a radiant from degree
         */
        [[nodiscard]] double to_radiant() const {
            return to_radiant(this->direction_);
        }

        static double to_radiant(int const dir) {
            return dir * ml5::util::PI / (full_degree / 2);
        }

        /*
         * Rotate points according to the transformation matrix
         */

        [[nodiscard]] std::vector<wxPoint> transform_points(std::vector<wxPoint>
points) const {
            for (auto &point: points) {
                auto const old = point;
                point.x = old.x * cos(to_radiant()) - old.y * sin(to_radiant());
                point.y = old.x * sin(to_radiant()) + old.y * cos(to_radiant());
            }
```

```
            return points;
        }

        /**
        * if an object moves out of sight it loops back
        * at the other side of the window
        **/
        virtual void stay_in_window(context_t& ctx) {
            auto const size = ctx.GetSize();
            if (this->position_.x + this->length() < 0) {
                this->position_.x = size.x - 1.0;
            }
            if (this->position_.y + this->length() < 0) {
                this->position_.y = size.y - 1.0;
            }
            if (this->position_.x > size.x) {
                this->position_.x = this->position_.x - size.x;
            }
            if (this->position_.y > size.y) {
                this->position_.y = this->position_.y - size.y;
            }
        }

        /**
         * Utility function to draw polygons
         */
        virtual void do_draw(context_t &ctx) const {
            auto points_vec = this->create_shape();
            ctx.DrawPolygon(points_vec.size(), &points_vec[0], this->position_.x, this
->position_.y);
        }

        [[nodiscard]] virtual int length() const = 0;

        /**
         * Create a polygon with its starting point in (0/0)
         */
        [[nodiscard]] virtual std::vector<wxPoint> create_shape() const = 0;

    private:

        /**
         * Create a polygon for this object with its offset and transformation already
calculated
         */
        [[nodiscard]] std::vector<wxPoint> create_transformed_shape_with_offset()
const {
            auto shape = this->create_shape();
            shape = this->transform_points(shape);
            for (auto& point : shape) {
                point.x += this->position_.x;
```

```
                point.y += this->position_.y;
            }
            return shape;
        }

    };
}
```

Listing 2. spaceship.h

```
#pragma once
#include "flying_object.h"

constexpr int spaceship_size = 30;
constexpr double acceleration_factor = 0.5;
constexpr double deacceleration_factor = 0.01;
constexpr int max_speed = 3;

namespace asteroids {

    class spaceship final : public flying_object {
    public:
        using context_t = ml5::paint_event::context_t;

        spaceship() :
            spaceship{ wxRealPoint() } {}

        explicit spaceship(wxRealPoint const pos) : flying_object{ pos} {
        }

        spaceship(int const x, int const y) : flying_object{ wxRealPoint(x, y) } {
        }

        void draw(context_t& ctx) override {
            this->stay_in_window(ctx);
            ctx.SetBrush(*wxCYAN_BRUSH);
            ctx.SetPen(*wxCYAN_PEN);
            do_draw(ctx);
        }

        void accelerate() {
            if (this->speed_ < max_speed) {
                this->speed_ += acceleration_factor;
            } else {
                this->speed_ = 1;
            }
        }

        void deaccelerate() {
            if (this->speed_ > 0) {
                this->speed_ -= deacceleration_factor;
```

```
            } else {
                this->speed_ = 0;
            }
        }

        [[nodiscard]] wxRealPoint position() const {
            return this->position_;
        }

        [[nodiscard]] double speed() const {
            return this->speed_;
        }

        [[nodiscard]] int direction() const {
            return this->direction_;
        }

    protected:

        [[nodiscard]] int length() const override {
            return spaceship_size;
        }

        [[nodiscard]] std::vector<wxPoint> create_shape() const override    {
            const std::vector<wxPoint> vec{
                wxPoint(this->length(), 0),
                wxPoint(0, -this->length() / 3),
                wxPoint(0, this->length() / 3)
            };
            return this->transform_points(vec);
        }
    };
}
```

Listing 3. asteroid.h

```
#pragma once
#include "flying_object.h"

namespace asteroids {

    constexpr int asteroid_min_size = 10;
    constexpr int crack_width = 30;
    constexpr double standard_speed = 1.5;
    constexpr int points = 1;

    constexpr int asteroid_size_count = 3;
    enum class asteroid_size {
        tiny = 1,
        medium = 2,
        big = 3
```

```
    };

    class asteroid final : public flying_object {
    public:
        using context_t = ml5::paint_event::context_t;

        explicit asteroid(wxRealPoint const pos) :
            flying_object{pos} {
            // Calculate a random asteroid size
            this->size_ = static_cast<asteroid_size>(rand() % asteroid_size_count +
1);
            // Random direction
            this->direction_ = rand() % full_degree;
            // Different sized asteroids have different speed
            this->speed_ = standard_speed / static_cast<double>(this->size_);
            // Where the Crack of the asteroid starts and ends, to give it some kind
of pacman shape
            this->crack_start_ = rand() % full_degree;
            this->crack_end_ = crack_start_ + crack_width;
        }

        void draw(context_t& ctx) override {
            this->stay_in_window(ctx);
            ctx.SetBrush(*wxBLACK_BRUSH);
            ctx.SetPen(*wxWHITE_PEN);
            do_draw(ctx);
        }

        /**
         * When an asteroid is hit it will split up in smaller asteroids, unless it
was already tiny
         */
        [[nodiscard]] std::vector<asteroid> split() const {
            std::vector<asteroid> parts;
            switch (this->size_) {
            case asteroid_size::big: {
                if (rand() % 2 == 0) {
                    parts.push_back(asteroid{ asteroid_size::medium, this->position_
});
                    parts.push_back(asteroid{ asteroid_size::tiny, this->position_ });
                } else {
                    for (auto i = 0; i < 3; i++) {
                        parts.push_back(asteroid{ asteroid_size::tiny, this->position_
});
                    }
                }
                break;
            }
            case asteroid_size::medium: {
                parts.push_back(asteroid{ asteroid_size::tiny, this->position_ });
                parts.push_back(asteroid{ asteroid_size::tiny, this->position_ });
```

```
                break;
            }
            case asteroid_size::tiny:
            default: break;
            }
            return parts;
        }

        /**
         * How much this asteroid is worth to the player
         */
        [[nodiscard]] static int score() {
            return points;
        }

        friend bool operator==(asteroid const& left, asteroid const& right) {
            return left.length() == right.length() && left.position_ == right
.position_;
        }


    protected:
        [[nodiscard]] int length() const override {
            return static_cast<int>(this->size_) * asteroid_min_size;
        }

        [[nodiscard]] std::vector<wxPoint> create_shape() const override {
            std::vector<wxPoint> vec;
            for (auto i = 0; i < full_degree / 2; i++) {
                wxPoint p;
                auto const point = i * 2;
                if (point > crack_start_ && point < crack_end_) {
                    p.x = cos(to_radiant(point)) * this->length() / 4;
                    p.y = sin(to_radiant(point)) * this->length() / 4;
                } else {
                    p.x = cos(to_radiant(point)) * this->length();
                    p.y = sin(to_radiant(point)) * this->length();
                }

                vec.push_back(p);
            }
            return this->transform_points(vec);
        }


    private:
        asteroid_size size_;
        int crack_start_;
        int crack_end_;

        asteroid(const asteroid_size size, wxRealPoint const pos) :
```

```
            asteroid{ pos } {
            this->size_ = size;
        }
    };
}
```

Listing 4. projectile.h

```
#pragma once
#include "flying_object.h"

namespace asteroids {
    constexpr int projectile_speed = 10;
    constexpr int projectile_size = 5;


    class projectile : public flying_object {
    public:
        projectile(wxRealPoint const& position,
            int const direction, bool is_enemy = false)
            : flying_object{position} {
            this->direction_ = direction;
            this->speed_ = projectile_speed;
            this->is_enemy_ = is_enemy;
        }

        void draw(context_t& ctx) override {
            if(this->is_enemy_) {
                ctx.SetBrush(*wxRED_BRUSH);
                ctx.SetPen(*wxRED_PEN);
            } else {
                ctx.SetBrush(*wxGREEN_BRUSH);
                ctx.SetPen(*wxGREEN_PEN);
            }
            ctx.DrawRectangle(this->position_,
                              wxSize{projectile_size, projectile_size});
        }

        /**
         * Check if projectile is still in window
         */
        [[nodiscard]] bool is_in_window(const int width, const int height) const {
            return !(this->position_.x < 0 || this->position_.x > width || this-
>position_.y < 0 || this->position_.y > height);
        }

    protected:
        [[nodiscard]] int length() const override {
            return projectile_size;
        }
```

```
        [[nodiscard]] std::vector<wxPoint> create_shape() const override {
            return std::vector<wxPoint>{
                wxPoint{0, 0},
                wxPoint{this->length(), 0},
                wxPoint{this->length(), this->length()},
                wxPoint{0, this->length()},
            };
        }

    private:
        bool is_enemy_;
    };
}
```

Listing 5. saucer.h

```
#pragma once
#include "flying_object.h"

namespace asteroids {
    constexpr int saucer_length = 30;
    constexpr int saucer_speed_limit = 2;
    constexpr int min_curve_height = 50;
    constexpr int min_curve_width = 200;

    class saucer : public flying_object {
    public:

        explicit saucer(const wxPoint& position)
            : flying_object{position} {
            this->speed_ = (static_cast<double>(rand() % saucer_speed_limit) + 1) *
((rand() % 2 == 0) ? -1 : 1) ;
        }

        saucer(const wxPoint& position,
                       const int max_height,
                       const int max_width)
            : saucer{position} {
            this->y_offset_ = position_.y;
            this->curve_height_ = rand() % (max_height ) ;
            this->curve_width_ = rand() % max_width;
        }

        void draw(context_t& ctx) override {
            this->stay_in_window(ctx);
            ctx.SetPen(*wxYELLOW_PEN);
            ctx.SetBrush(*wxBLACK_BRUSH);
            do_draw(ctx);
        }

        void move() override {
```

```
            this->position_.x += this->speed_;
            this->position_.y = this->curve_height_ * sin(to_radiant(static_cast
<double>(this->curve_width_) * this->position_.x)) + this->y_offset_;
        }

        static int score() {
            return 5;
        }

        [[nodiscard]] wxRealPoint position() const {
            return this->position_;
        }

    protected:
        [[nodiscard]] int length() const override {
            return saucer_length;
        }

        [[nodiscard]] std::vector<wxPoint> create_shape() const override {
            return std::vector<wxPoint>{
                wxPoint(0, 0),
                wxPoint(this->length() / 5, this->length() / 3),
                wxPoint((this->length() / 5) * 4, this->length() / 3),
                wxPoint(this->length(), 0),
                wxPoint((this->length() / 5) * 4, -this->length() / 3),
                wxPoint(this->length() / 5, -this->length() / 3),
            };
        }

    private:
        int curve_height_;
        int curve_width_;
        int y_offset_;
    };
}
```

Listing 6. asteroids_window.h

```
#pragma once
#include <ml5/ml5.h>
#include "spaceship.h"
#include "asteroid.h"
#include "projectile.h"
#include "saucer.h"

namespace asteroids {
    constexpr int tick_interval = 10;
    constexpr int window_width = 800;
    constexpr int window_height = 600;
    constexpr int asteroid_limit = 20;
    constexpr int saucer_limit = 4;
```

```
    constexpr int ticks_between_shots = 20;
    constexpr int asteroid_spawn_chance = 200;
    constexpr int saucer_spawn_chance = 500;
    constexpr int enemy_projectile_spawn_chance = 300;

    class asteroids_window final : public ml5::window {
    public:
        using context_t = ml5::paint_event::context_t;

        asteroids_window() : window{"A really cool Game! :("} {
            set_prop_allow_resize(false);
            set_prop_initial_size({window_width, window_height});
        }

        void on_init() override {
            set_prop_background_brush(*wxBLACK_BRUSH);
            start_timer(std::chrono::milliseconds{tick_interval});
            this->ship_ = spaceship{this->get_width() / 2, this->get_height() / 2};

            add_menu("Game", {
                        {"Restart", "Restart the game"},
                    });
        }

        void on_paint(ml5::paint_event const& event) override {
            set_status_text("Score: " + std::to_string(score_) +
                "; Speed: " + std::to_string(ship_.speed()));

            auto& ctx = event.get_context();
            if (game_over_) {
                ctx.SetPen(*wxRED_PEN);
                ctx.SetBrush(*wxRED_BRUSH);
                ctx.SetTextForeground(*wxRED);
                const wxFont font(50, wxFONTFAMILY_TELETYPE, wxFONTSTYLE_NORMAL,
wxFONTWEIGHT_BOLD);
                ctx.SetFont(font);
                ctx.DrawText("GAME OVER", 225, this->get_height() / 4);
                ctx.DrawText("Your Score:", 200, (this->get_height() / 2));
                ctx.DrawText(std::to_string(this->score_), 225, 400);
                return;
            }

            spawn_asteroid(ctx);
            spawn_saucer();
            spawn_enemy_projectile();

            auto s = ctx.GetSize();
            ship_.draw(ctx);
            for (auto& asteroid : this->asteroids_) {
                asteroid.draw(ctx);
            }
```

```
            for (auto& projectile : this->projectiles_) {
                projectile.draw(ctx);
            }
            for (auto& sauce : this->saucers_) {
                sauce.draw(ctx);
            }
            for (auto& projectile : this->enemy_projectiles_) {
                projectile.draw(ctx);
            }
        }

        void on_timer(ml5::timer_event const& event) override {
            std::vector<projectile> new_projectiles;
            //Check if a projectile had a collision
            for (const auto& proj : this->projectiles_) {
                bool had_collision = false;
                auto asteroid = this->asteroids_.begin();
                while (!had_collision && asteroid < this->asteroids_.end()) {
                    if (proj.has_collision(*asteroid)) {
                        this->score_ += asteroid->score();
                        had_collision = true;
                    } else {
                        ++asteroid;
                    }
                }

                if (had_collision) {
                    auto split_asts = asteroid->split();
                    this->asteroids_.erase(asteroid);
                    this->asteroids_.insert(this->asteroids_.end(), split_asts.
begin(), split_asts.end());
                    continue;
                }

                auto sauce = this->saucers_.begin();
                while (!had_collision && sauce < this->saucers_.end()) {
                    if (proj.has_collision(*sauce)) {
                        this->score_ += sauce->score();
                        had_collision = true;
                    } else {
                        ++sauce;
                    }
                }

                if (!had_collision) {
                    if (proj.is_in_window(this->get_width(), this->get_height())) {
                        new_projectiles.push_back(proj);
                    }
                } else {
                    this->saucers_.erase(sauce);
```

```
            }
        }

        this->projectiles_ = new_projectiles;

        ship_.move();

        // Check if the ship had a collision
        for (auto& asteroid : this->asteroids_) {
            asteroid.move();
            if (this->ship_.has_collision(asteroid)) {
                game_over();
            }
        }
        for (auto& saucer : this->saucers_) {
            if (this->ship_.has_collision(saucer)) {
                game_over();
            }
            saucer.move();
        }

        for (auto& projectile : this->enemy_projectiles_) {
            if (this->ship_.has_collision(projectile)) {
                game_over();
            }
            projectile.move();
        }

        for (auto& projectile : this->projectiles_) {
            projectile.move();
        }

        count_down_for_projectiles_--;

        /* Ugly hack since there is no key up or down event.*/
        if (not_accelerated_count_ > 10000) {
            ship_.deaccelerate();
        } else {
            not_accelerated_count_ += 1;
        }
        refresh();
    }

    void on_key(ml5::key_event const& event) override {
        switch (event.get_key_code()) {
        case 'w': {
            ship_.accelerate();
            not_accelerated_count_ = 0;
            break;
        }
        case 'a': {
```

```
                ship_.rotate(rotate_direction::left);
                break;
            }
            case 'd': {
                ship_.rotate(rotate_direction::right);
                break;
            }
            case WXK_SPACE: {
                if (count_down_for_projectiles_ <= 0) {
                    count_down_for_projectiles_ = ticks_between_shots;
                    projectile pro{ship_.position(), ship_.direction()};
                    projectiles_.emplace_back(pro);
                }
                break;
            }
            default: break;
            }
        }

        void on_menu(ml5::menu_event const& event) override {
            const auto& item{event.get_item()};

            if (item == "Restart") {
                this->ship_ = spaceship{this->get_width() / 2, this->get_height() /
2};
                this->asteroids_.clear();
                this->projectiles_.clear();
                this->enemy_projectiles_.clear();
                this->saucers_.clear();
                this->score_ = 0;
                this->not_accelerated_count_ = 0;
                this->count_down_for_projectiles_ = 0;
                this->game_over_ = false;
                start_timer(std::chrono::milliseconds{tick_interval});
            }
        }

    private:
        spaceship ship_;
        std::vector<asteroid> asteroids_;
        std::vector<projectile> projectiles_;
        std::vector<projectile> enemy_projectiles_;
        std::vector<saucer> saucers_;
        int not_accelerated_count_ = 0;
        int count_down_for_projectiles_ = 0;
        int score_ = 0;
        bool game_over_ = false;

        void spawn_asteroid(const context_t& ctx) {
            if (rand() % asteroid_spawn_chance != 0
                || this->asteroids_.size() > asteroid_limit) {
```

```
            return;
        }
        auto const size = ctx.GetSize();
        double x = 0;
        double y = 0;
        switch (rand() % 4) {
        case 2: {
            // Spawn bottom
            y = size.GetHeight();
        }
        case 0: {
            // Spawn top
            x = rand() % size.GetWidth();
            break;
        }
        case 1: {
            // Spawn right
            x = size.GetWidth();
        }
        case 3: {
            // Spawn left
            y = rand() % size.GetHeight();
            break;
        }
        default: break;
        }
        const asteroid ast{wxRealPoint{x, y}};
        this->asteroids_.push_back(ast);
    }

    void spawn_saucer() {
        if (rand() % saucer_spawn_chance != 0
            || this->saucers_.size() > saucer_limit) {
            return;
        }
        this->saucers_.push_back(saucer{
            wxPoint{0, this->get_height() / 2},
            this->get_height() / 2, 3
        });
    }

    void spawn_enemy_projectile() {
        if (this->saucers_.empty() || rand() % enemy_projectile_spawn_chance != 0)
{
            return;
        }
        auto const start = this->saucers_
                               .at(rand() % this->saucers_.size())
                               .position();
        auto const vec = this->ship_.position() - start;
        int const direction = atan(vec.y / vec.x) * 180 / ml5::util::PI;
```

```
            this->enemy_projectiles_.emplace_back(
                start,
                direction,
                true
            );
        }

        /**
         * Stop the game
         */
        void game_over() {
            this->game_over_ = true;
            this->refresh();
            this->stop_timer();
        }
    };
}
```

Listing 7. asteroids_app.h

```
#pragma once
#include <ml5/ml5.h>
#include "asteroids_window.h"

namespace asteroids {

    class  asteroids_app final : public ml5::application {
    protected:
        [[nodiscard]] std::unique_ptr <ml5::window> make_window() const override {
            return std::make_unique<asteroids_window>();
        }
    };
}
```

Listing 8. main.cpp

```
#include "./asteroids_app.h"


int main() {
    srand(time(nullptr));
    asteroids::asteroids_app{}.run();
}
```

Testfälle