SWE4	Übung zu Softwareentwicklung mit modernen Plattformen 4	SS 2022, Übung 1

☐ Gruppe 1 (J. Heinzelreiter)

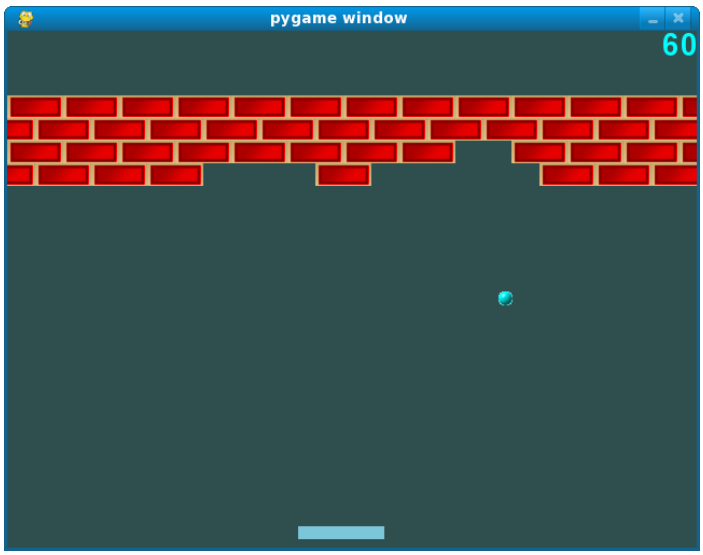
☒ Gruppe 2 (P. Kulczycki) Name: Angelos Angelis Aufwand [h]: 27

☐ Gruppe 3 (M. Hava) Peer-Review von: _____

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

MiniBreakout (`src/breakout`)

Breakout ist ein absoluter Klassiker unter den Videospielen, der Mitte der Siebzigerjahre von Atari entwickelt wurde. Das zugrundeliegende Spielkonzept ist ganz einfach. Der Spielende hat die Kontrolle über einen Schläger, mit dem er versuchen muss, einen Ball so lange wie möglich im Spiel zu halten. Mit dem Ball kann er Ziegel aus einer Mauer schießen. Für jeden abgeschossenen Ball bekommt er Punkte gutgeschrieben. Die Wertigkeit der Ziegel kann unterschiedlich sein. Mit Fortdauer des Spiels steigert sich der Schwierigkeitsgrad, da die Geschwindigkeit des Balls zunimmt.



Ihre Aufgabe ist es nun, unter möglichst weitreichendem Einsatz der MiniLib (Grafikklassen, Behälterbibliothek) einen einfachen Breakout-Klon zu entwickeln, der zumindest folgende Anforderungen erfüllt:

- Der Spielende kann den Schläger mit Maus und Tastatur bewegen.
- Der Austrittswinkel des Balls hängt von seinem Eintrittswinkel und der Position, an dem der Ball den Schläger berührt hat ab.
- Die Mauer besteht aus mehreren Reihen von Ziegeln. Die Wertigkeit der Ziegel (wie viele Punkte der Spielende für das Abschießen des Ziegels bekommt) wird dargestellt (farbcodiert oder in Form einer Beschriftung).
- Der aktuelle Spielstand wird angezeigt und laufend aktualisiert.
- Über ein Menü können verschieden Konfigurationsparameter eingestellt werden (initiale Ballgeschwindigkeit, Größe des Schlägers, Parameter, die den Aufbau der Mauer beschreiben etc.).

Sie dürfen bei der Realisierung des Spiels Ihrer Fantasie aber freien Lauf lassen und können es optional mit zusätzlicher Funktionalität und Spezialeffekten anreichern. Hier nur einige Anregungen dazu.

Die MiniLib baut auf dem Framework *wxWidgets* auf, welches eine breite Palette an Grafikoperationen zur Verfügung stellt. So ist es relativ einfach möglich, Bilder, die in gängigen Bildformaten vorliegen, in ein Fester zu zeichnen:

```
// bei der Initialisierung der Anwendung
wxImage::AddHandler(new wxPNGHandler);
wxImage brick_image;
brick_image.LoadFile(wxT(".\\images\\brick.png"), wxBITMAP_TYPE_PNG);
brick_image.Rescale(width, height);

// in on_paint
auto &context = event.get_context();
context.DrawBitmap(brick_image, position);

// beim Beenden der Anwendung
wxImage::CleanUpHandlers();
```

wxWidgets enthält auch Funktionalität zum Abspielen von Sounds (wxSound). Mit Bildern und Sounds lässt sich die Qualität Ihres Spiels natürlich wesentlich steigern.

Sie können aber auch an der Spiellogik feilen. Beispielsweise können Sie mehrere Spielebenen mit unterschiedlichen Mauern realisieren. Der Spielende kann mehrere Leben haben. Ziegel können mit Spezialeffekten ausgestattet werden: Joker mit Bonuspunkten, Verdoppelung der Punkte über eine gewisse Zeitspanne hinweg, Erzeugung zusätzlicher Bälle, die zusätzlich ins Spiel gebracht werden etc.

Lösungsidee:

Der Schläger und die Blöcke sind rectangles wobei der Ball eine Elypse ist.

Shape

Ist die Basisklasse für alle Objekte im Spiel. Die Klasse gibt einpaar Basis Methoden vor wie die getPos() Methode eine scale methode und auch eine move Methode die Hauptsächlich vom Schläger benutzt wird

Racket

Ist an sich ein Rectangle. Ist von dem User Steuebar entweder mit der Maus oder auch mit a&d. Der Schläger wird immer Rot gerendert. Die Move methode von shape stellt klar dass der Schläger nicht außerhalb des Bildschirms bewegt.

Blocks

Sind die Blöcke die der Spieler zerstören muss. Hierbei ist die unterste Reihe an Blöcken immer am wenigsten Punkte Wert und die oberste Reihe ist immer am meisten Punkte Wert. Jede Reihe wird auch in verschiedenen Farben gerendert. Ebenfalls ist es bei meiner Implementierung gleich wie bei dem Original und zwar dass wenn die Blöcke von der Seite getroffen werden der y Wert des Richtungsvektors des Balls negiert wird (und nicht der x-Wert wie es bei manchen Neu Auffassungen ist).

Ball

Der Ball wird auch standartmäßig Rot gerendert. Je nachdem wo der Ball den Schläger trifft ändert sich sein Richtungs-Winkel. Wenn er den Boden trifft wird das Spiel beendet. Wenn er die Ränder des Bildschirms berührt wird entsprechend seine Richtung negiert.

Utilities

Sehr wichtige Klasse. Enthält globale Variablen wie z.B Fenstergröße, Schlägergröße, Ballgröße, eine Methode um das Spiel zu verschnellern je nach erreichter Punktezahl des Spielers, Methode um Farbe zu wechseln, die erreichte Punktezahl an sich, Blockgröße und vieles mehr.

Weitere Hinweise

Das Fenster ist immer auf 800x600 gelockt. Man kann das Fenster weder größer noch kleiner machen. Dies habe ich mit `set_prop_allow_resize(false);` erreicht. Ebenfalls werden die Punkte nicht auf die Spielfläche gerendert sondern ich nutze die set_status_text Methode von der Window Klasse aus.

Probleme

Folgende Probleme gibt es: Keine Random Start-Position des Balls, Nicht die Sinnvollste Lösung um Hitbox-Probleme zu beheben, könnte bei größeren „Maps“ zu Problemen führen

CODE:

```
#pragma once

#include <random>
#include "shape.h"

namespace breakout {
	class ball final : public shape {
	public:
		int speed{ 1 };
		/*int dx{ rand() % (5 - (-5) + 1) + (-5) };
		int dy{ rand() % (3 - 1 + 1) + 1 };*/
		int dx{ 2 };
		int dy{ 1 };
		explicit ball(wxRect box, const wxPen& pen, const wxBrush& brush) :
shape{ box, pen, brush } {}

		using context_t = ml5::paint_event::context_t;

		void auto_move(int steps) {
			int x = m_box.GetPosition().x +dx;
			int y = m_box.GetPosition().y +dy;

			m_box.SetPosition({x+(dx*steps),y+(dy*steps)});
		}

	protected:
		void do_draw(context_t& context) const override {
			context.DrawEllipse(m_box);
		}
	};
}


#pragma once

#include "shape.h"

namespace breakout {
	class block final : public shape {
	public:
		int points{ 0 };
		explicit block(wxRect box, const wxPen& pen, const wxBrush& brush) :
shape{ box, pen, brush } {}

		using context_t = ml5::paint_event::context_t;

	protected:
		void do_draw(context_t& context) const override {
			context.DrawRectangle(m_box);
		}
	};
}


#pragma once
```

```
#include <ml5/ml5.h>
#include "draw_window.h"

class draw_application : public ml5::application {
    std::unique_ptr<ml5::window>make_window() const override {
        return std::make_unique<draw_window>();
    }
};
```

```
#pragma once
#include <ml5/ml5.h>
#include <iostream>
#include <memory>   //for unique_ptr, but included in ml5 anyway
#include <cmath>
#include "shape.h"
#include "rectangle.h"
#include "Utilities.h"
#include "racket.h"
#include "ball.h"
#include "blocks.h"

using std::cout;
using std::endl;

class draw_window : public ml5::window {
public:
	enum class shape_t { line, rectangle, ellipse };

	draw_window() : ml5::window{ std::string{"breakout game"} } {
		set_prop_allow_resize(false);
		set_prop_initial_size({ utilities.window_width,utilities.window_height
});
		set_prop_background_brush(*wxBLACK_BRUSH);
	}


	void fillmap() {
		utilities.score = 0;
		for (int i = 0; i < utilities.rows; i++)
		{
			wxBrush currentColor = utilities.NextColor();
			for (int j = 0; j < utilities.cols; j++)
			{
				wxPoint block_pos{ utilities.x_edge_offset + (j *
utilities.brick_x_size), utilities.y_edge_offset + (i * utilities.brick_y_size) };
				wxPoint block_size{ block_pos.x + utilities.brick_x_size,
block_pos.y + utilities.brick_y_size };
				m_current_shape =
std::make_unique<breakout::block>(wxRect{ block_pos,block_size }, *wxBLACK_PEN,
currentColor);
				m_current_shape->points = (utilities.rows - i) * 10;
				m_shapes.add(std::move(m_current_shape));
			}
		}
	}
```

```
	void on_init() override {
		srand(time(NULL));
		rack = std::make_unique<breakout::racket>(wxRect{ utilities.rakstart,
utilities.rakstart }, *wxRED_PEN, *wxRED_BRUSH);
		rack->set_right_bottom(utilities.raksize);
		fillmap();

		game_ball = std::make_unique<breakout::ball>(wxRect{
utilities.ballstart,utilities.ballstart }, *wxRED_PEN, *wxRED_BRUSH);
		game_ball->set_right_bottom(utilities.ballsize);
		start_timer(utilities.time);

		refresh();
		add_menu("&Racket Options", {
				{ "&Increase Size","Increses Racket Size"},
				{ "&Decrease Size","Decreses Racket Size"}
			});
		add_menu("&Ball Options", {
				{ "&Increase Speed","Increses Ball Speed"},
				{ "&Decrease Speed","Decreses Ball Speed"}
			});
		add_menu("&Map Options", {
				{ "&Small Size","Small sized Map"},
				{ "&Medium Size","Medium Sized Map"},
				{ "&Large Size","Large Sized Map"}
			});
	}

	void on_menu(ml5::menu_event const& event) override {
		std::string menu_item = event.get_title_and_item();
		if (menu_item == "Racket Options/Increase Size") {
			changeablesize += 10;
			utilities.raksize.x = rack->getPos().x + changeablesize;
			rack->m_box.SetRightBottom(utilities.raksize);
			cout << rack->getPos().x << endl;
			refresh();
		}
		else if (menu_item == "Racket Options/Decrease Size") {
			changeablesize -= 10;
			utilities.raksize.x = rack->getPos().x + changeablesize;
			rack->m_box.SetRightBottom(utilities.raksize);
			cout << rack->getPos().x << endl;
			refresh();
		}
		else if (menu_item == "Ball Options/Increase Speed") {
			game_ball->speed++;
		}
		else if (menu_item == "Ball Options/Decrease Speed") {
			game_ball->speed--;
		}
		else if (menu_item == "Map Options/Large Size") {
			utilities.cols = 16;
			utilities.rows = 6;
			m_shapes.clear();
			utilities.readjust_block_size();
			fillmap();
			refresh();
		}
```

```
		else if (menu_item == "Map Options/Medium Size") {
			utilities.cols = 11;
			utilities.rows = 4;
			m_shapes.clear();
			utilities.readjust_block_size();
			fillmap();
			refresh();
		}
		else if (menu_item == "Map Options/Small Size") {
			utilities.cols = 6;
			utilities.rows = 3;
			m_shapes.clear();
			utilities.readjust_block_size();
			fillmap();
			refresh();
		}
	}

	void on_mouse_move(ml5::mouse_event const& event) override {
		if (m_current_shape) {
			m_current_shape->set_right_bottom(event.get_position());
			refresh();				//damit die Linie wirklich gezeichnet
wird
		}

		rack->move(event.get_position());
		refresh();
	}

	void on_key(ml5::key_event const& event) override {
		wxPoint cur_pos = rack->getPos();
		switch (event.get_key_code())
		{
		case WXK_LEFT: {
			cur_pos.x -= 35;
			rack->move(cur_pos);
			refresh();
			break;
		}
		case WXK_RIGHT: {
			cur_pos.x+=35;
			rack->move(cur_pos);
			refresh();
			break;
		}
		default:
			break;
		}
	}

	void on_paint(ml5::paint_event const& event) override {
		set_status_text("Score: " + std::to_string(utilities.score));
		for (auto &ptr_shape : m_shapes) {
			if (ptr_shape != nullptr) {
				ptr_shape->draw(event.get_context());
			}
		}
		rack->draw(event.get_context());
```

```
		game_ball->draw(event.get_context());
	}

	void on_timer(ml5::timer_event const& event) override {
		int counter = 0;
		bool col = false;
		game_ball->auto_move(game_ball->speed);
		//collision with borders
		if (game_ball->getPos().x >= 800 || game_ball->getPos().x < 0)
		{
			game_ball->dx *= -1;
			utilities.hit_wall = true;
		}
		if (game_ball->getPos().y >= 600) {
			game_ball->dy *= -1;
			cout << "You lost restart the game" << endl;
			exit(1);
		}
		if (game_ball->getPos().y < 0) {
			game_ball->dy *= -1;
			cout << "hit top" << endl;
			utilities.hit_wall = true;
		}
		//Collision with racket
		if (utilities.racket_hit)
		{
			if (game_ball->getPos().y + 7.5 >= rack->getPos().y)
			{
				if (game_ball->getPos().x >= rack->getPos().x &&
game_ball->getPos().x <= rack->getPos().x + (changeablesize / 2)) { //If collision
with left half of racket
					game_ball->dx--;
					game_ball->dy *= -1;
					utilities.hit_wall = true;
					utilities.racket_hit = false;
				}
				if (game_ball->getPos().x > rack->getPos().x +
(changeablesize / 2) && game_ball->getPos().x <= rack->getPos().x + changeablesize)
{//If collision with right half of racket
					game_ball->dx++;
					game_ball->dy *= -1;
					utilities.hit_wall = true;
					utilities.racket_hit = false;
				}
			}

		}
		//Collision with bricks
		for (auto& ptr_shape : m_shapes) {
			if (utilities.hit_wall)
			{
				if ((game_ball->getPos().y - 7.5 <= ptr_shape->getPos().y
+ utilities.brick_y_size && game_ball->getPos().y + 7.5 >= ptr_shape->getPos().y) &&
(game_ball->getPos().x + 5 >= ptr_shape->getPos().x && game_ball->getPos().x - 5 <=
ptr_shape->getPos().x + utilities.brick_x_size))
				{
					game_ball->dy *= -1;
					col = true;
```

```
                                utilities.hit_wall = false;
                                utilities.racket_hit = true;
                                break;
                        }
                        else if ((game_ball->getPos().x - 5 <= ptr_shape-
>getPos().x + utilities.brick_x_size) && (game_ball->getPos().x + 5 >= ptr_shape-
>getPos().x) && (game_ball->getPos().y <= ptr_shape->getPos().y +
utilities.brick_y_size && game_ball->getPos().y >= ptr_shape->getPos().y)) {
                                game_ball->dy *= -1;
                                col = true;
                                utilities.hit_wall = false;
                                utilities.racket_hit = true;
                                break;
                        }
                        counter++;
                }
        }
        if (col)
        {
                utilities.bricks_destroyed++;
                utilities.bricks_destroyed_count++;
                utilities.score += m_shapes[counter]->points;
                m_shapes.remove(m_shapes[counter]);
                if (m_shapes.size() == 0)
                {
                        cout << "You won the Game congratz" << endl;
                        exit(1);
                }
                restart_timer(utilities.decrease_Time());
                col = false;
                counter = 0;
        }
        refresh();
    }

private:
    util utilities{};
    std::unique_ptr<breakout::racket>  rack{};
    std::unique_ptr<breakout::ball> game_ball{};

    shape_t
    m_shape_type{shape_t::rectangle};
    std::unique_ptr<breakout::block>                    m_current_shape{}; //{}
init with nullptr
    ml5::vector<std::unique_ptr<breakout::block>>m_shapes{};
};



#pragma once
#include "shape.h"

class ellipse : public shape {
public:
    explicit ellipse(wxRect box, const wxPen& pen, const wxBrush& brush) : shape{
box, pen, brush } {
    }

protected:
```

```
	void do_draw(context_t& context) const override {
		context.DrawEllipse(m_box);
	}
};
```

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

namespace breakout {
	class racket final : public shape {
	public:
		explicit racket(wxRect box, const wxPen& pen, const wxBrush& brush) :
shape{ box, pen, brush } {}

		using context_t = ml5::paint_event::context_t;


	protected:
		void do_draw(context_t& context) const override {
			context.DrawRectangle(m_box);
		}
	};
}
```

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

class rectangle : public shape {
public:
	explicit rectangle(wxRect box, const wxPen& pen, const wxBrush& brush) :
shape{ box, pen, brush } {
	}

protected:
	void do_draw(context_t& context) const override {
		context.DrawRectangle(m_box);
	}
};
```

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

class shape : public ml5::object {
public:
	using context_t = ml5::paint_event::context_t;
	wxRect  m_box{};
	explicit shape(wxRect box, const wxPen& pen, const wxBrush &brush) : m_box{
box }, m_pen{ pen }, m_brush{ brush } {
	}

	void set_right_bottom(wxPoint new_end) {
		m_box.SetRightBottom(new_end);
	}

	void move(wxPoint new_pos) {
		if (new_pos.x + changeablesize >= 800) {
```

```
                        new_pos.y = 580;
                        new_pos.x = 800 - changeablesize;
                        m_box.SetPosition(new_pos);
                }
                else {
                        new_pos.y = 580;
                        m_box.SetPosition(new_pos);
                }
                if (new_pos.x <= 0)
                {
                        new_pos.y = 580;
                        new_pos.x = 0;
                        m_box.SetPosition(new_pos);
                }
                else {
                        new_pos.y = 580;
                        m_box.SetPosition(new_pos);
                }
        }

        wxPoint getPos() {
                return m_box.GetPosition();
        }

        wxRect get_mbox() {
                return m_box;
        }

        bool empty() const {
                return m_box.GetWidth() == 0 && m_box.GetHeight() == 0;
        }

        void draw(context_t& context) const {
                if (!empty()) {
                        context.SetPen(m_pen);
                        context.SetBrush(m_brush);
                        do_draw(context);
                }
        }


protected:
        virtual
        void do_draw(context_t &context) const = 0;


        wxPen   m_pen{};
        wxBrush m_brush{};
};



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

int changeablesize = 50;

class util : public ml5::object {
public:
```

```
	util() {};
	~util() {};

	int bricks_destroyed = 0;
	int bricks_destroyed_count = 0;

	const int x_edge_offset = 1;
	const int y_edge_offset = 50;
	const int window_width = 800;
	const int window_height = 600;
	const int window_height_offset = (window_height - 20);

	int score = 0;

	std::chrono::milliseconds time{ 20 };
	std::chrono::milliseconds min_time{ 7 };
	int cols = 16;
	int rows = 6;

	bool hit_wall = true;//if wall has hit a wall before
	bool racket_hit = true;//if ball has hit the racket before

	//TODO READJUST SIZE WHEN CHANGING MAP SIZE
	int brick_x_size = window_width / cols;
	int brick_y_size = brick_x_size /2;

	void readjust_block_size() {
		brick_x_size = window_width / cols;
		brick_y_size = brick_x_size / 2;
	};

	wxPoint rakstart{ window_width / 2,window_height_offset };
	wxPoint raksize{ (window_width / 2) + changeablesize, window_height_offset -
10 };

	wxPoint ballstart{ window_width / 2,window_height_offset / 2 };
	wxPoint ballsize{ (window_width / 2) + 10, (window_height_offset / 2) - 15 };

	std::chrono::milliseconds decrease_Time() {
		if (time == min_time)
		{
			return time;
		}
		if ( bricks_destroyed_count >= trunc((rows*cols)/20))
		{
			bricks_destroyed_count = 0;
			return time--;
		}
		return time;
	};

	wxBrush NextColor() {
		index++;
		if (index == colors.size()+1)
		{
			index = 1;
		}
```

```
		return colors[index-1];
	}

private:
	int index{ 0 };
	std::vector<wxBrush> colors{
*wxYELLOW_BRUSH,*wxWHITE_BRUSH,*wxRED_BRUSH,*wxBLUE_BRUSH,*wxGREEN_BRUSH };
};
```

```
#include <ml5/ml5.h>
#include "draw_application.h"

int main () {
	try
	{
		draw_application app{};
		app.run();
	}
	catch (const std::exception& x)
	{
		std::cout << x.what() << std::endl;
	}
}
```

Tests:

1)Allgemeine Tests: Verlieren, Punktevergabe, Vergrößern/kleinern vom Schläger, Verschnellern/langsamen vom Ball, Verkleiner/größern der Map, Kollisionen, Score, uvm

https://filebox.fhooecloud.at/index.php/s/sZQEL9z5oKMxJLF

2)Gewinnen (Bin sehr schlecht in diesem Spiel deshalb habe ich es nur für den Testfall ausgeschaltet dass man verlieren kann xD)

https://filebox.fhooecloud.at/index.php/s/e7KMGZCb7LJXMqa

Falls das mit den Videos nicht funktioniert bitte schreib mich auf Teams an!