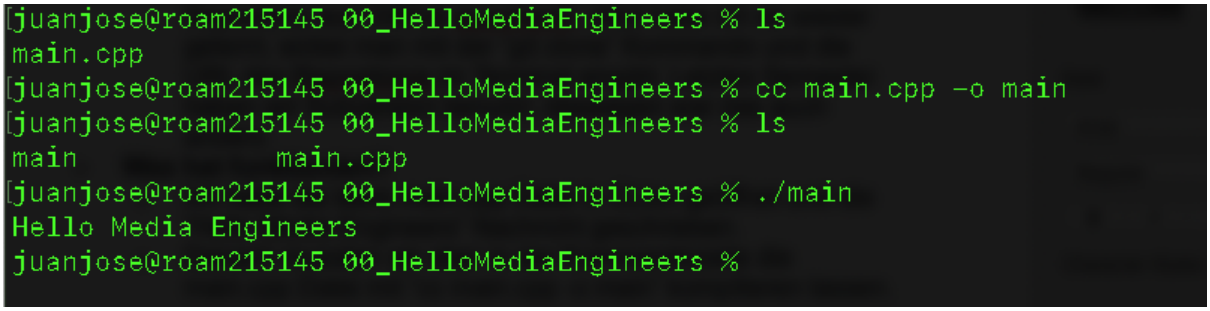
16.01.24

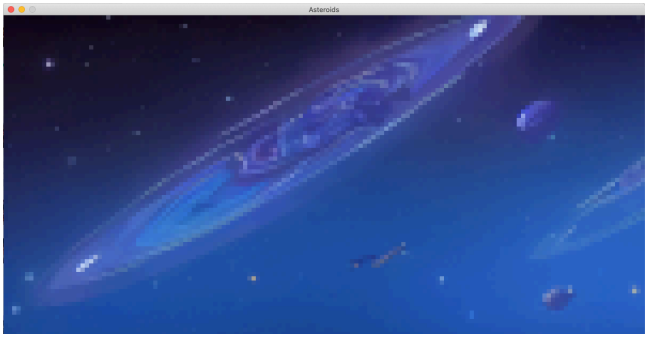
Programmier 3-B Protokoll

Übung 1

- **Aufgabe00**:
  - **Was haben Sie gelernt?**
    - Ich habe aus dem letzen Semester vergessen, wie man en Repository aus GitHub auscheckt. Jetzt habe ich es wieder gelernt, wobei man mit der “git clone” Kommando und die URL des Repositorys im Terminal eingibt. Letztes Semester haben wir Subversion benutzt, deswegen war das auch anders.
  - **Was hat funktioniert?**
    - Ich habe die leere main.cpp Datei in CLion geöffnet und die “Hello Media Engineers” Nachricht geschrieben.
    - Danach mussten wir aber in der Kommandozeile die main.cpp Datei mit “cc main.cpp -o main” kompilieren lassen.
    - Danach ./main ausgeführt.
  - **Was hat nicht funktioniert?**
    - Alles hat geklappt.
  - **Was haben Sie probiert?**
    - Wir haben zusätzlich mit CLion zu kompilieren versucht. Es hat auch funktioniert.



- **Aufgabe01**:
  - **Was haben Sie gelernt?**
    - Wir haben gelernt dass raylibcpp ein Wrapper fur raylib ist. Es macht die Nutzung für die originale Bibliothek einfacher um es in C++ zu programmieren. Dieser hat nur additionale Header Dateien die im Zusammenhang mit ray lib funktioniert.
  - **Was hat funktioniert?**
    - Wir könnten dann das leere Program erst kompilieren als wir es in CLion geöffnet haben.
    - Danach mussten wir ein Fenster initialisieren mit dem vorgegebenen Hintergrund.
  - **Was hat nicht funktioniert?**
    - Als das Fenster sich öffnet, es ist offensichtlich, dass das Hintergrundbild viel zu gross ist. Wir müssen es zur Grosse des Fensters noch genau anpassen.
  - **Was haben Sie probiert?**
    - Für den Moment, unsere Lösung ist bei der Angabe des Fensters Breite und Höhe manuell einzugeben.




```cpp
int main()
{
    // Initialize the window
    int screenWidth = 1200;
    int screenHeight = 600;
    raylib::Window window( width: screenWidth,  height: screenHeight,  title: "Asteroids");

    // Load background
    raylib::Texture2D background( fileName: "resources/background.png");
```

```cpp
// Game loop
while (!window.ShouldClose()) {
    // Start drawing
    BeginDrawing();
    window.ClearBackground( color: RAYWHITE);

    // Draw the background
    background.Draw( posX: 0,  posY: 0,  tint: WHITE);
```

- **Aufgabe02**:
  - **Was haben Sie gelernt?**
    - Wir haben gelernt, dass mit raylib solche Strukturen wie Vektor oder Texture zu implementieren. Diese waren nötig um die Position und Darstellung des Spaceships im Fenster anzuzeigen.
  - **Was hat funktioniert?**
    - Es hat geklappt das Spaceship im Fenster zu initialisieren. Wir haben auch ein paar Objekte für das Spiel eingelistet.
  - **Was hat nicht funktioniert?**
    - Das Spaceship kann sich noch nicht bewegen mit Tastatureingabe.
  - **Was haben Sie probiert?**
    - Wir haben dem Spaceship eine Draw() Methode gegeben, sodass es wie bei dem Hintergrund im Fenster angezeigt werden kann.

Objekt: Raumschiff:

- Daten: Position, Geschwindigkeit, Gesundheit, Waffen, Punkte.
- Verhalten: Bewegung, Schießen, Kollisionserkennung, Punktezählung.

Objekt: Asteroid:

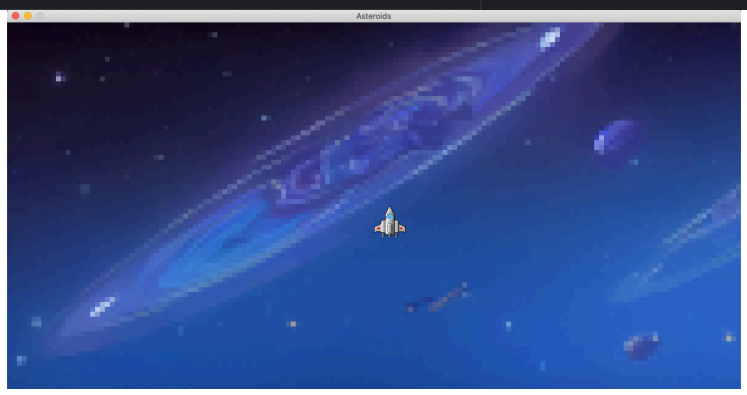
- Daten: Position, Geschwindigkeit, Größe.
- Verhalten: Bewegung, Zerstörung, Aufspaltung in kleinere Asteroiden, Kollisionserkennung.

Objekt: Laserstrahl:

- Daten: Position, Richtung, Geschwindigkeit.
- Verhalten: Bewegung, Kollisionserkennung.

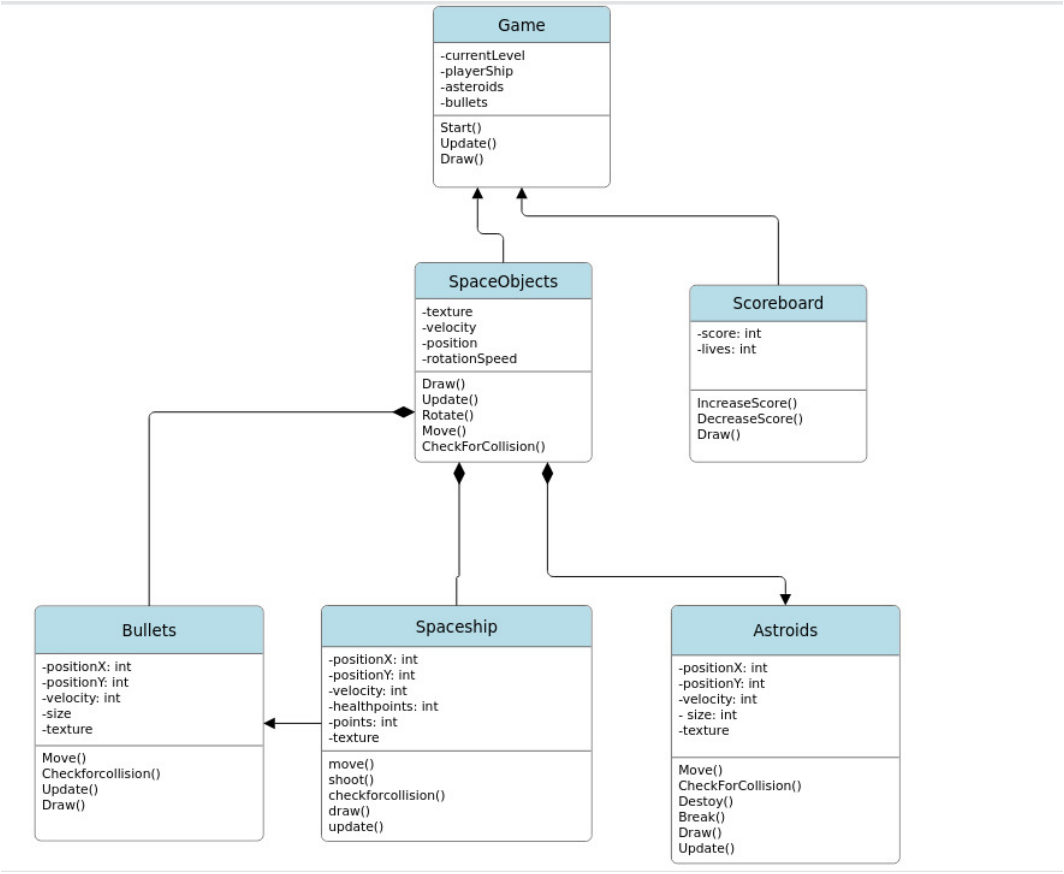
```cpp
class Spaceship {
public:
    raylib::Vector2 position;
    raylib::Texture2D texture;
    float speed;
    int healthPoints;

    Spaceship(const std::string& texturePath, raylib::Vector2 startPos) {
        healthPoints = 100;
        position = startPos;
        speed = 0.0f;

        // Load Spaceship texture
        texture = raylib::Texture2D( fileName: texturePath);
    }
```

```cpp
// Game loop
while (!window.ShouldClose()) {
    // Start drawing
    BeginDrawing();
    window.ClearBackground( color: RAYWHITE);

    // Draw the background
    background.Draw( posX: 0,  posY: 0,  tint: WHITE);

    // Draw the spaceship
    playerShip.Draw();
```



Übung 2

- **Aufgabe01**:
  - **Was haben Sie gelernt?**
    - Wir könnten die Grundlagen vom UML-Diagrammen wieder lernen. Wir haben die Konzepte von deren Beziehungen in den Spiel implementiert.
  - **Was hat funktioniert?**
    - Mittels einen Online-Tool hats geklappt ein Diagramm zu machen.
  - **Was hat nicht funktioniert?**
    - Es hat alles geklappt
  - **Was haben Sie probiert?**
    - Es war ein bisschen unklar wie welche Art von Klassen man braucht insgesamt in dem Diagramm. Also zusätzlich zu dem Spaceship, Asteroid und Bullet, man hat noch eine Game Klasse gebraucht, die den Spiellogik handelt, als auch eine SpaceObject Klasse die die erste drei Klassen umfasst und von denen sie vererben würden.



- **Aufgabe02**:
  - **Was haben Sie gelernt?**
    - Wir haben die Implementation von Vererbung in C++ gelernt.
  - **Was hat funktioniert?**
    - Dafür mussten wir also eine neue Klasse erstellen die bei uns SpaceObject heisst. Von diese Klasse haben Spaceship, Asteroid und Bullet Attribute und Methoden vererbt.
    - Wir haben noch eine Game Klasse erstellt, die die allgemeine Spiellogik handelt. Sie zeigt erst nur die Lebensanzahl von der Spaceship und den Score. Es kann aber zu Kollisionen noch nicht reagieren.
  - **Was hat nicht funktioniert?**
    - Die Asteroiden haben alle noch die gleiche Grösse und bewegen sich noch ein bisschen komisch.
  - **Was haben Sie probiert?**
    - Es war ein bisschen unklar wie welche Art von Klassen man braucht insgesamt in dem Diagramm. Also zusätzlich zu dem Spaceship, Asteroid und Bullet, man hat noch eine Game Klasse gebraucht, die den Spiellogik handelt, als auch eine SpaceObject Klasse die die erste drei Klassen umfasst und von denen sie vererben würden.

```
class GameManager
{
    using GameObjectContainer = std::list<std::shared_ptr<GameObject>>;

private:
    int score_;

    GameObjectContainer objects_;
    std::shared_ptr<Spaceship> player_;
    raylib::Vector2 generateRandomPos();

    void shoot();

public:
    GameManager();
    void AddSpaceship();
    void update();
    void drawObjects() const;
```

```cpp
void GameManager::AddSpaceship() {
    raylib::Vector2 shipPos = generateRandomPos();

    auto ship :shared_ptr<Spaceship> = std::make_shared<Spaceship>( initialPos: shipPos);
    objects_.push_back( x: ship);
    player_ = ship;
}

void GameManager::AddAsteroids() {
    raylib::Vector2 asteroidPos = generateRandomPos();
    float initialRot = 0.0f;
    raylib::Vector2 asteroidVelocity{
            x: static_cast<float>(GetRandomValue( min: -3,  max: 3)),
            y: static_cast<float>(GetRandomValue( min: -3,  max: 3))
    };

    objects_.push_back( x: std::make_shared<Asteroid>( initialPos: asteroidPos, initialRot,
}
```

```cpp
#include "asteroids.h"
#include "bullet.h"

Asteroid::Asteroid(raylib::Vector2 initialPos, float initialRot, raylib::Vector2 initialVelocity)
        : GameObject(initialPos,  texturePath: texturePath_,  initialScale: 1.f, initialRot)
        , rotationSpeed(2.0f)
        , movementSpeed(2.0f)
        , velocity_(initialVelocity){

}

void Asteroid::update() {

    move( moveVec: velocity_);
    rotate( deg: rotationSpeed);
    GameObject::update();
}

void Asteroid::handleCollision(std::shared_ptr<GameObject> other) {
    auto bullet :shared_ptr<Bullet> = std::dynamic_pointer_cast<Bullet>( r: other);
    if (bullet) {
        // Mark asteroid for removal
    }
}
```

Übung 3

- **Aufgabe01**:
  - **Was haben Sie gelernt?**
    - Wir haben gelernt Ableitung mittels PhysicsObject im unseren Projekt einzusetzen.
  - **Was hat funktioniert?**
    - Jetzt bewegt sich die Spaceship mehr wie im Weltraum. Also der Spaceship rutscht ein wenig, selbst nachdem der Spieler aufgehört hat, den Thrustknopf zu drücken.
  - **Was hat nicht funktioniert?**
    - Am Anfang haben wir versucht, die vorgeschlagenen Werte für Thrust und Reibung zu verwenden, aber das Spaceship flog in weniger als einer Sekunde einfach aus dem Bildschirm.
  - **Was haben Sie probiert?**
    - Wir haben die Werte angepasst bei Thrust = 60 und die Reibung = 0.07. Jetzt bewegt sich besser.

```
void Spaceship::update()
{
    if(IsKeyDown( key: KEY_W))
    {
        float thrust = 60.0f;
        accelerate( acceleration: raylib::Vector2{ x: 0.f,  y: -thrust}.Rotate( degrees: DEG2RAD * rot_));
    }
    if(IsKeyDown( key: KEY_A))
    {
        rotate( deg: -rotationSpeed_);
    }
    if(IsKeyDown( key: KEY_D))
    {
        rotate( deg: rotationSpeed_);
    }

    PhysicsObject::update();
}
```

```
PhysicsObject::PhysicsObject(raylib::Vector2 initialPos, std::string texturePath,
                             float initialScale, float initialRot, float friction)
        : GameObject(initialPos, texturePath, initialScale, initialRot), friction_(friction) {}

void PhysicsObject::update() {
    const float dt = 1.0f / 60.0f;

    velocity_ = velocity_ + acceleration_ * dt; // Beschleunigung

    velocity_ = velocity_ - velocity_ * friction_; // Reibung

    pos_ += velocity_; // Position aktualisieren

    acceleration_ = raylib::Vector2( x: 0,  y: 0); //Beschleunigung zurücksetzen

    GameObject::update();
}
```

- **Aufgabe02**:
  - **Was haben Sie gelernt?**
    - Wir könnten die Kollisionserkennung in unserem Spiel so implementieren, dass der GameManager die Erkennung für die Objekte verwaltet, anstatt dies für jedes einzelne Objekt unabhängig implementieren zu müssen.
  - **Was hat funktioniert?**
    - Die Spaceship erkennt wenn sie mit einen Asteroid kollidiert und subtrahiert Health Points wenn es passiert.
    - Auch wenn eine Kollision zwischen Projektil und Asteroid wird erkannt und das wurde den Asteroid zerstören.
  - **Was hat nicht funktioniert?**
    - Obwohl es in unserem Spiel kleinere Asteroiden gibt. Das Spaceship erkennt noch immer die ursprüngliche Größe des Asteroiden. Aus diesem Grund kommt es bei einigen Asteroiden zu einer Erkennung und die Lebenspunkte werden reduziert, obwohl sie sich nicht wirklich berühren.
    - Das gleiche problem kommt zwischen Kollision von Asteroiden und Projektilen.
  - **Was haben Sie probiert?**
    - Da diese Probleme Teil der zusätzlichen Übung sind, haben wir beschlossen, es dabei zu belassen. Das liegt nur daran, dass wir die Zeit brauchen, um für die bevorstehenden Prüfungen zu lernen.

```
void GameManager::checkCollisions() {
    for (auto& obj1 : shared_ptr<GameObject> & : objects_) {
        for (auto& obj2 : shared_ptr<GameObject> & : objects_) {
            if (obj1 != obj2 && isColliding(obj1, obj2)) {
                std::cout << "Collision" << std::endl;
                obj1->handleCollision( other: obj2);
                obj2->handleCollision( other: obj1);
            }
        }
    }
}

bool GameManager::isColliding(const std::shared_ptr<GameObject> &obj1, const std::shared_ptr<GameObject> &obj2) {
    Vector2 pos1 = obj1->getPosition();
    Vector2 pos2 = obj2->getPosition();

    float radius1 = obj1->getRadius();
    float radius2 = obj2->getRadius();

    return CheckCollisionCircles( center1: pos1, radius1,  center2: pos2, radius2);
}
```

```cpp
void Spaceship::handleCollision(std::shared_ptr<GameObject> other) {
    std::shared_ptr<Asteroid> asteroid = std::dynamic_pointer_cast<Asteroid>( r: other);
    if (asteroid != nullptr) {
        health_--;
    }
}
```

```cpp
void Asteroid::handleCollision(std::shared_ptr<GameObject> other) {
    std::shared_ptr<Projectile> projectile = std::dynamic_pointer_cast<Projectile>( r: other);
    if (projectile != nullptr) {
        markForDeletion();
    }
}
```

```cpp
void Projectile::handleCollision(std::shared_ptr<GameObject> other) {
    std::shared_ptr<Asteroid> asteroid = std::dynamic_pointer_cast<Asteroid>( r: other);
    if (asteroid != nullptr) {
        markForDeletion();
    }
}
```