Portfolio

Formales zur Abgabe:

- Die Abgabe erfolgt in Moodle.
- Abgabefrist: Siehe Moodle
- Es muss eine Archiv-Datei hochgeladen werden.

(nur folgende Archive sind gestattet: zip, rar, 7z)

- Legen Sie eine **VisualStudio Solution oder ein CMake Projekt** für die Portfolio Lösung an.

- Visual Studio Solutions müssen **ohne Debug und Release Verzeichnisse (ohne build Verzeichnisse)** abgebeben werden.

- Ihre Abgabe muss eine **schriftliche Abgabe** (pdf, word) beinhalten.

Diese enthält:
  - In welchen Sourcen (Dateien), welche Aufgaben bearbeitet werden.

- Andere Arten von Abgaben sind nicht gestattet!

- Optional: Geben sie gebaute Windows binarys mit ab.

(Diese müssen in einem separaten Ordner liegen und die schriftliche Abgabe muss auf sie verweisen)

Formales zur Abgabe – Anmerkungen:

- In den Aufgaben werden die Mindestanforderungen beschrieben, es kann darüber hinaus Bonuspunkte geben.

- Beispiele:
  2D-Version mit eigenen Grafiken statt Konsolen-Version. Mehr konkrete Kindklassen. Erweitertes Interface. Mehr Konfigurationsmöglichkeiten/Mehr Attribute.

- Es steht Ihnen frei englische oder deutsche Bezeichner zu verwenden

- Teilweise ist die Verwendung von Bibliotheken erlaubt.
  Sourcen immer referenzieren.

- Code Separierung und Strukturierung ist verpflichtend.
  - Trennung Deklaration und Definition.
    - *.h und *.cpp (Funktionen und Klassen)
    - *.h und *.hpp (bei templates)
  - Eine Klasse pro Header (Ausnahme sind lediglich Hilfsklassen oder Typendeklarationen, wie z.B. Enums)

  →Nichteinhaltung führt zu Punktabzug.

- Alle Diagramme (UML, PAP) müssen in einem separaten Ordner direkt im Archiv liegen (root)

Aufgabe 1 – PAP

Erstellen Sie für folgende Aufgaben ein PAP-Diagramm (kein source code notwendig!)

Designe einen einfachen "Endless Runner" Spielablauf.

Beschreibung:

Ein "Endless Runner" ist ein Genre, bei dem der Spieler eine Figur steuert, die unaufhörlich vorwärts läuft, während Hindernisse vermieden werden müssen. Deine Aufgabe ist es, den Spielablauf dieses einfachen "Endless Runner"-Spiels zu entwerfen.

Anforderungen:

1. Der Spieler soll eine Figur (z. B. ein Läufer, ein Auto, ein Tier usw.) steuern können, die sich von links nach rechts bewegt.

2. Es sollen Hindernisse auf dem Bildschirm erscheinen, die der Spieler durch links rechts bewegen ausweichen kann.

3. Der Spieler soll Punkte erhalten, je weiter er kommt.

4. Das Spiel soll endlos fortgesetzt werden, bis der Spieler ein Hindernis berührt.

5. Der Punktestand soll ausgegeben werden, wenn er sich ändert.

Bonuspunkte (optional):

1. Füge Power-Ups oder spezielle Objekte hinzu, die dem Spieler helfen oder ihn herausfordern.

2. **Programmiere das designte Spiel**

Aufgabe 2 – Sprite

Erstellen Sie eine Struktur „Sprite“

Die Struktur soll folgenden Informationen beinhalten:

- x, y, z – Koordinate (Diese sollen mittels eines Datentypes abgebildet werden, z.B. struct)
- drawable (wahr/falsch)
- animationFrames (ganze positive Zahl)
- width (ganze positive Zahl)
- height (ganze positive Zahl)
- currentFrame (ganze positive Zahl)
- pointer auf einen container oder array mit *frames*
- enum type (animate, character, object, character_animation)

  Anmerkung: Ein frame ist ein eigener Datentyp und soll als leere Struktur implementiert werden.

Aufgabe 3 – Würfelspiel

Implementiere ein textbasiertes Würfelspiel.

Entwickle ein einfaches textbasiertes Würfelspiel, bei dem der Spieler gegen den Computer antritt. Der Spieler und der Computer werfen jeweils mit einem Würfel und derjenige mit der höheren Augenzahl gewinnt.

Anforderungen:

1. Erstelle eine Funktion `wuerfeln()`, die einen Würfelwurf simuliert und die gewürfelte Augenzahl zurückgibt.
2. Der Spieler soll zuerst wählen können, wie viele Augen der Würfel haben soll (z. B. 6, 10, 20). Verwende dazu den Datentyp `int` für die Anzahl der Augen.
3. Implementiere eine Funktion `spielRunde(int augen)`, die eine Spielrunde durchführt. Diese Funktion sollte den Spieler- und Computerwurf ausgeben und den Gewinner der Runde bestimmen.
4. Implementiere eine Funktion `main()`, die das Spiel steuert und die Rundenabfolge regelt. Diese Funktion sollte auch die Eingabe des Spielers für die Anzahl der Würfelaugen verarbeiten und das Spiel starten. Sie sollte zudem eine Funktion aufrufen gameStartInfo(), die den Spieler über das Spiel und dessen Bedienung informiert. Sie hat zudem die Aufgabe den Spieler über notwendige Eingaben zu informieren und diese einzulesen.
5. Das Spiel soll mehrere Runden ermöglichen, bis der Spieler entscheidet, aufzuhören. Verwende dazu einen geeigneten Datentyp, um den Spielerstatus (z. B. "weiter spielen" oder "aufhören") zu speichern.

6. Implementiere eine einfache Punktezählung, um den Gewinner über mehrere Runden zu ermitteln. Verwende dazu einen geeigneten Datentyp, um die Punktzahl zu speichern.
7. Füge eine Funktion hinzu, die dem Spieler erlaubt, seine Würfel zu wechseln, bevor er wirft.

Optional:

- Erweitere das Spiel um eine einfache Benutzeroberfläche, die dem Spieler die Auswahl des Würfels und das Würfeln erleichtert.

Hinweis:

Verwende Funktionen, um den Code zu strukturieren und die Wiederverwendbarkeit zu erhöhen. Separiere den Code auf mehrere Dateien, wenn sinnvoll. Wähle geeignete Datentypen für die verschiedenen Parameter und Variablen aus, um eine klare und effiziente Implementierung zu ermöglichen.

Aufgabe 4 –Pflanzenklassen

1. Erstellen Sie eine Klassen-Hierarchie für Pflanzen.
2. Erstellen Sie eine Basisklasse für die Pflanzen,
3. sowie die jeweiligen abgeleiteten Klassen Baum, Blume, Strauch und Gras.
4. Lagern Sie alle Basisfunktionalitäten in die Basisklasse aus.
5. Stellen Sie sicher, dass die Basisklasse eine Methode zur Berechnung der Wuchshöhe deklariert.
6. Schreiben Sie eine main Funktion, die alle Klassen verwendet.
7. Testen Sie das Programm mit unterschiedlichen Werten für alle Klassen.
8. Folgende weitere Attribute soll eine Pflanze haben:
    - Wurzeln
        - Länge
    - Blätter
        - Farbe
    - Samen
        - Durchmesser

Das Programm muss folgende Eigenschaften und Funktionalitäten besitzen:

1. Benutzerausgabe darüber, dass hier Informationen zu Pflanzen abgefragt werden können.
2. Der Benutzer (des Programms) kann sich für eine der oben genannten

Pflanzenarten entscheiden (Auswahl).

Die Klassen müssen folgende Eigenschaften und Funktionalitäten besitzen:

1. Methoden zur Berechnung der durchschnittlichen Wuchshöhe in Abhängigkeit eines Parameters *t, der Parameter t steht für die Zeit, die die Pflanze Zeit hatte zu wachsen*.

   (der Parameter t wird durch eine Benutzereingabe gesetzt).

2. Generierung einer Benutzerausgabe.

Beispielausgabe:

Die durchschnittliche Wuchshöhe des Baums beträgt 15 Meter.

Anmerkungen zur Implementierung:

- Informationen, die eine Klasse zur Berechnung benötigt, werden in Klassenattributen gespeichert.
- Die Berechnungen finden in Methoden der Klassen statt.
- Stellen Sie sicher, dass:
- Negative Zahlen für die Wuchshöhe nicht akzeptiert werden.
- Keine Zeichen(ketten) für die Wuchshöhe eingegeben werden können.
- Es wird eine gültige Eingabe erwartet.
- Die Eingabe wird so lange wiederholt, bis sie gültig ist.
- Geben Sie adäquate Ausgaben aus.
- Die Berechnungen werden korrekt durchgeführt.

Aufgabe 5 – Custom dynamic Array

Aufgabenteil a)

Wir haben Arrays in C kennengelernt. Jetzt da wir OOP kennen und somit Klassen in C++, besteht der Drang uns eine komplexe Array Klasse zu erstellen.

Erstellen Sie eine Array Klasse MyCppArray.

Diese Klasse soll die folgenden Eigenschaften haben:

1. Die Klasse soll vorläufig nur Integers unterstützen.
2. Die Klasse soll ihre eigene Größe kennen.
3. Folgende Abfragen werden unterstützt:
    a. Was ist der kleinste Wert meines Arrays
    b. Was ist der größte Wert meines Arrays
    c. Gibt es einen konkreten Wert im Array? Wenn ja, dann soll die Position ausgeben werden.
4. Die Array Klasse darf von der Größe her limitiert sein (nicht dynamisch)
5. Schreiben Sie eine main Routine die alle oben genannten Features demonstriert.

Aufgabenteil b)

1. Modifizieren Sie die Aufgabe so, dass man den Array dynamisch setzen kann. Das heißt die Größe des Arrays dynamisch zur Laufzeit angepasst werden kann
2. Modifizieren Sie die main Funktion so, dass der dynamische Array automatisch mit unterschiedlichen Werten gefüllt wird.

   Dasselbe Array Objekt soll dabei zwei Mal mit unterschiedlich vielen Werten befüllt werden und dessen Größe korrekt angepasst werden.
3. Die gesamte Funktionalität aus a soll erhalten bleiben.

Aufgabe 6 – Auto

Erstellen Sie ein UML-Diagramm des Klassenkonzeptes für diese Aufgabe

Erstellen Sie eine abstrakte Basis-Klasse BaseCar, die das interface für alle Autoarten definiert.

Erstellen Sie mindesten zwei Konkrete Auto-Klassen die von BaseCar ableiten

Jedes Auto besitzt folgende Komponenten (die wiederum als Klassen oder Strukturen implementiert werden müssen)

Wheels/Reifen

Gears/Getriebe

Engine/Motor

Breaks/Bremsen

Frame/Rahmen

Suspension/Federung

Schreiben Sie jeweils zwei konkrete Auto Klassen, die unterschiedliche engines, brakes, wheels, usw verwendet.

(unterschiedlich heißt:zwei unterschiedliche Klassen für konkrete Motoren, zwei unterschiedliche Klassen für konkrete bremsen,…)

Folgende Funktionalität soll das Programm haben:

1. Die Autos befinden sich im Namespace car.
2. Es muss mit dem base class pointer der Auto Klasse gearbeitet werden. Arbeiten Sie zum Erzeugen der Ausgabe mit einem Basisklassenpointer.
3. Es müssen smart pointer (e.g. shared pointer) verwendet werden.
4. Dem Benutzer wird mitgeteilt welche konkreten Autos zur Verfügung stehen. Auf Anfrage des Benutzers wird das konkrete Auto auf dem heap erstellt und im Basisklassenpointer gespeichert.
5. Das komplette Interface zur Steuerung des Autos wird von der Basisklasse definiert.
6. Es müssen Methoden zum Vorwärtsbeschleunigen, Bremsen, Rückwärtsbeschleunigen, nach links oder rechts Abbiegen und zum Starten und Stoppen des Motors beinhalten.
7. Die konkrete Implementierung kann in der Basisklasse oder den Kindklassen stattfinden (die Wahl sollte natürlich bewusst getroffen werden – Was macht am meisten Sinn?)
8. Die Methoden sollen auf die Attribute (also Motor, Bremsen…) zugreifen, um eine spezifische Ausgabe zu generieren. (Siehe Beispielausgabe)
9. Ermöglichen Sie dem Benutzer das Starten/Stoppen des Motors, Vorwärts/Rückwärtsfahren sowie das Bremsen und Links-/Rechtsabbiegen. (mehr ist natürlich erlaubt)

Komponenten Anmerkung:

10. Die Auto-Komponenten (engines, brakes, wheels…) müssen selbst als Klassen implementiert werden.

    Die Auto-Komponenten müssen das Fahrverhalten des Autos beeinflussen und müssen somit in den Methoden in (Siehe Punkt7) verwendet werden.

    Beispiel: Die Vorwärtsbeschleunigung ist abhängig von der konkreten Leistung des Motors und der Kraftübertragung der Reifen.

    Überlegen sie sich wie die einzelnen Komponenten das Fahrverhalten konkret beeinflussen. Geben Sie den einzelnen Komponenten ihres Autos entsprechende Attribute und Methoden, um auf diese Werte zuzugreifen.

    Beispiel:

    spCar->getEngine->startEngine()

    spCar->getEngine->gethorsePower()

11. Die Komponenten befindet sich im Namespace car::component.

Auszug einer Beispielausgabe des Programms.

```
…
Benutzer hat einen <Autotyp> erstellt.
Benutzer hat <Motortyp> gestartet
<Autotyp> fährt los und beschleunigt mit
<n meter/s^2, abhängig von dem Motor und abhängig
von den Reifen >
Benutzer stoppt Beschleunigung und hat <n km/h,
abhängig von Beschleunigungszeit>
<Autotyp> fährt rückwärts und beschleunigt mit <n
meter/s^2, abhängig von dem Motor und abhängig von
den Reifen >
Benutzer stoppt Beschleunigung und hat <n km/h,
abhängig von Beschleunigungszeit>
…
<Autotyp> fährt nach rechts
…
<Autotyp> fährt nach links
…
<Autotyp> bremst, hat <n meter, abhängig von den
Bremsen und reifen> zum Stoppen gebraucht.
…
<Motor> gestoppt
 …
```

Aufgabe 7 – Raylib Sprite Klasse

Erfordert die Nutzung der Raylib Bibliothek.

Es kann folgende Vorlage genutzt werden:

https://github.com/chfhhd/raylibstarter-minimal

Erstelle eine Klasse Sprite, die folgenden Anforderungen genügt:

Die Klasse hat folgende public Attribute:

```
int pos_x_;
int pos_y_;
Texture2D texture_;
```

- Ein Objekt der Klasse darf nicht über den leeren Standardkonstruktor instanziierbar sein.

- Implementiere den Konstruktor:

```
Sprite(int pos_x, int pos_y, Texture2D texture);
```

- Implementiere einen weiteren Konstruktor, dem keine Texture2D übergeben wird, sondern stattdessen ein Parameter über den der Dateiname der zu ladenden Textur angegeben wird. Der Konstruktor soll die Textur dann selbst über die Raylib Funktion LoadTexture() laden.

- Implementiere einen Destruktor, der die die Texture des Sprite Objekts mit der Raylib Funktion UnloadTexture() aus dem VRAM entlädt.

- Die Klasse befindet sich im Namespace game.

Aufgabe 8 – Eine Level Klasse

Erfordert die Nutzung der Raylib Bibliothek.

Aufgabenteil a)

Erstelle eine Klasse Level, die folgenden Anforderungen genügt:

Die Levelklasse befindet sich im Namespace game und besitzt ein public Attribut sprites_ vom Typ:

```
std::vector<std::shared_ptr<game::Sprite>>
```

Die Klasse besitzt eine public Methode zum Zeichnen aller Sprites auf den Bildschirm mit Hilfe der Raylib Funktion DrawTexture().

Die Methode besitz darüber hinaus noch eine public Methode positionRandomly() mit denen alle Sprites zufällig auf dem Bildschirm verteilt werden. Die Sprites dürfen sich dabei überlappen.

Aufgabenteil b)

Erweitere das bisheriges Programm wie folgt:

Über die main() Methode sollen in einer for-Schleife 20 Sprites in das sprites-Attribute eines Level-Objekts gelegt werden und anschließend mit positionRandomly() zufällig positioniert werden.

Die Main Game Loop gibt die Sprite dann über die Zeichenfunktion der Levelklasse aus.

Aufgabenteil c)

Erweitere das bisheriges Programm wie folgt:

- Erstelle eine Update() Methode für die Level Klasse (wird in der Main Game Loop aufgerufen). Die Update() Methode soll jeden einzelnes Sprite mit unterschiedlicher Geschwindigkeit und unterschiedlicher Richtung bewegen.
- Die Sprites können an den Bildschirmrändern abprallen oder auf der gegenüber liegenden Bildschirmseite wieder auftauchen.

Implementiere zuletzt noch eine der folgenden Erweiterung:

- Die Sprite prallen voneinander ab.
- Die Sprites löschen sich bei Kollision gegenseitig aus.
- die Sprites zerfallen bei Kollision in kleinere Objekte (wie bei Asteroids)
- ... die letzte, Erweiterung, wenn Du wirklich schnell bist: Erweitere das Ganze zu einem einfachen Asteroids Klon :-) ...