UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

11. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2014 / 2015

Aufgabe 1: STL-Funktionen (20 Punkte)

1. Schreiben Sie ein einfaches Objekt, das ein `unary_function`-Modell implementiert. Das Objekt sollte einen internen Zustand haben, den man auf jedes Element des Vektors mit Integerwerten anwenden kann. Extrahieren Sie die Summe des Vektors und verwenden Sie dabei das Konstrukt `for_each`. Geben Sie die Summe auf dem Bildschirm aus (10 Punkte)
2. Schauen Sie sich das Code-Fragment `aufg1_2.cpp` (StudIP) an. Die `compose1` Funktion soll sich im `__gnu_cxx` namespace befinden. Kommentieren Sie den Code, um darzustellen, welche Intention der Autor hatte. Erklären Sie, warum der Code so nicht funktioniert. Welche Änderungen müssen vorgenommen werden? Selbstverständlich sollen Sie den Code so abändern, dass er übersetzt und das tut, was ursprünglich geplant war. Die STL-Dokumentation hilft Ihnen die Funktionen `bind2nd` und `compose1` zu verstehen (10 Punkte).

Aufgabe 2: Threads in C++ (80 Punkte)

Ziel der Übung:

Ziel der Übung ist es, sich mit Threads in C++ auseinander zu setzen. In dieser Übung werden wir die Software um die Funktionalität erweitern, die Raumschiffmodelle schießen zu lassen. Jede abgefeuerte Kugel wird dazu in einen in einem eigenen Thread verwaltet.

Aufgabenstellung:

In dieser Übung werden wir zwei neue Klassen entwickeln: `Bullet` und `Fighter`. Die Klasse `Bullet` hat zunächst folgende Grundsignatur:

```
class Bullet: public TriangleMesh
{

public:

    /**
     * @brief Contructor. Build a bullet on the given Fighter's
     *                    position. The bullet will move on the
     *                    given axis.
     * @param   fighter_position   Position of the fighter that shoots this bullet
     * @param   fighter_axis   Axis the bullet will move on
     */
    Bullet(Vertex<float> fighter_position, Vertex<float> fighter_axis);

    ~Bullet();

    /**
     * @brief Moves the bullet until it's lifetime is over.
     */
       void run();

       /**
        * @brief Starts bullet movement
        */
       void start();



    /*
     * @brief Stops bullet movement
     */
       void stop();

    /**
     * @brief Renders the bullet via glutSolidSphere.
     */
```

```
        void render();

    /**
     * @brief Returns the status of this bullet.
     * @return false, if the bullet's lifetime is over and true otherwise
     */
        bool isAlive();

private:
      // Lifetime, i.e., how many timesteps the bullet visible
        static const int m_lifetime = 9000;

        // True, if the bullet's lifetime isn't over yet
        bool m_alive;

        // Flight direction of the bullet
      Vertex<float> m_fighterAxis;

      /// TODO: ADD TIMING AND THREAD FUNCTIONALITY MEMBERS!!
};
```

Beschreibung der zu implementierenden Funktionalität:

Der Konstruktor der Klasse bekommt als Parameter die aktuelle Position und Ausrichtung des Modells, von dem die Kugel abgeschossen wurde. In der `run()`-Methode läuft eine Schleife `m_lifetime` Iterationen. In jedem Schleifendurchgang wird die Position der Kugel um eine Einheit entlang `m_flightAxis` bewegt (10 Punkte). Nach Beendigung der Schleife wird `m_alive()` von `true` auf `false` gesetzt, um anzuzeigen, dass die Kugel ihre maximale Schussdistanz erreicht hat und nun nicht mehr aktiv ist (15 Punkte). Nutzen Sie zur Abarbeitung der `run()`-Methode einen `std::thread`. Fügen Sie diesen der Klasse hinzu. Achten Sie darauf, dass der Thread erst startet, nachdem `start()` aufgerufen wurde. Ein Aufruf von `stop()` joint den Thread und setzt `m_alive` auf `false` (10 Punkte).

Ob die Kugel ihre Endposition erreicht hat, soll von `isAlive()` berichtet werden. Dazu wird schlicht der Status von `m_alive` zurückgegeben (5 Punkte). Stellen Sie sicher, dass die Schleife in `run()` nicht zu schnell läuft, indem Sie den Thread nach jeder Iteration für 1000 Mikrosekunden schlafen legen (10 Punkte). Dazu können versenden sie die Funktion `std::this_thread::sleep_for` mit einer entsprechenden Dauer. Warum ist das notwendig und sinnvoll (10 Punkte) ?

Die `Fighter`-Klasse, die Sie implementieren sollen, ist eine Spezialisierung der Klasse `TexturedMesh` und erweitert diese um die Methode `shoot()`:

```
/**
 * @brief   Represents a fighter that can shoot bullets.
 */
class Fighter : public TexturedMesh
{

public:
    /**
     * @brief Adds a bullet to the fighter's vector of bullets.
     */
       void shoot();

    /**
     * @brief Renders the fighter and calls the bullets' render method.
     */
       void render();

private:
    // A vector with the bullets this fighter has shot.
       vector<Bullet*> m_bullets;


};
```

Beschreibung der zu implementieren Funktionalität:

Jeder Aufruf von `shoot()` erzeugt eine neue Bullet-Instanz und fügt sie in `m_bullets` ein. In jedem Aufruf von `render()` wird zunächst das Modell gezeichnet. Anschließend wird durch die Liste der Kugeln iteriert und alle aktiven Kugeln werden an ihrer aktuellen Position gerendert. Sobald eine Kugel nicht mehr aktiv ist (`isAlive()` gibt `false` zurück), soll sie aus der Liste der zu rendernden Kugeln entfernt werden. Nutzen Sie dazu einen geeigneten Iterator auf `m_bullets` (10 Punkte). Eine vorhandene Kugel können Sie mittels `erase()` aus dem Vektor löschen (10 Punkte). Wenn Ihre Implementierung sich an das vorgegebene Interface hält, wird nach Integration in die in StudIP bereit gestellte Vorgabe beim Drücken der Space-Taste eine Kugel vom Mittelpunkt des geladenen Modells in dessen Blickrichtung losgeschossen.

Abgabe

Die Dateien der Klassen `Fighter` und `Bullet` sowie Ihre Erklärungen und STL-Programme sind bis Montag den 12. Januar 08:00 Uhr mit der Angabe Ihrer Gruppe an die Adresse cpp@informatik.uni-osnabrueck zu mailen.