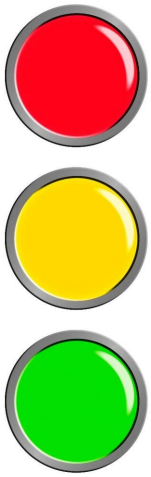
Simulation einer Ampelsteuerung

Valentino Lazarevic, 5CHIF

February 2021



Contents

1 Einführung

Heutzutage sind Ampeln aus der Gesellschaft nicht mehr wegzudenken. Sie regeln den Verkehr in bestimmten Situationen und sorgen für einen sicheren Verkehrsfluss und vermeiden Unfälle. Zu Wissen ist das an einer Kreuzung alle Ampeln miteinander verbunden sind, sodass sie aufeinander abgestimmt umschalten. Die Ampeln funktionieren computergesteuert durch einen Algorithmus, der über einen Schaltkasten, zu den Ampeln der Kreuzung durch Kabel miteinander verbunden ist.

Die Aufgabe bestand aus der Simulation einer Ampel, die vier Straßeneinmündungen besitzt. Weiters sollte das Programm mittels Kommandozeile spezifiert werden können. Damit ist gemeint, wie viele Autos je Zeiteinheit an der Straßeneinmündung ankommen.

Die Lösung der Simulation besteht aus der Ampel-Klasse, der Straßen-Klasse und der Auto-Klasse. Die Ampel muss automatisch schalten und bestimmte Regeln befolgen. Die Straße lässt anhand des aktuellen Ampel Status, die Autos fahren. Kommen wir zur Implementierung.

2 Implementierung

2.1 Enum

"Mit enum können Aufzählungstypen definiert werden. Dazu gehören Wochentage oder Farben. Den Elementen eines Aufzählungstyps werden Namen zugeordnet, obwohl sie intern natürlich als Zahlen codiert werden."[Wil]

In meinem Programm sind folgende enums definiert:

Listing 1: C++ enums.h

```
1 enum TrafficColor {RED = 1, YELLOW = 2, GREEN = 3};
2 enum Directions {NORTH = 1, SOUTH = 2, WEST = 3, EAST = 4};
```

2.2 Ampel - (trafficlight)

Aufgabe

Die **Ampel** ist in meinem Programm zuständig, die richtigen Ausgaben zu tätigen. Um der Straße mitzueilen, welche Straßeneinmündung auf Grün oder Rot bzw. auf Gelb gestellt ist.

Klassenstruktur

Listing 2: C++ trafficlight.h - Klassenstruktur

```
1  class TrafficLight
2  {
3  private:
4      TrafficColor colorNorthSouth;    //enum
5      TrafficColor colorWestEast;      //enum
6      std::mutex l_mutex;
7  public:
8      TrafficColor getNorthSouthColor();
9      TrafficColor getWestEastColor();
10     void startTrafficLight();
11 };
```

Wie wir sehen können besitzt die Klasse drei wichtige Funktionen. Die ersten beiden Funktionen - getNorthSouthColor und getWestEastColor liefern den aktuellen Status der Ampel an der gewünschten Straße. Die Funktion start-TrafficLight startet und verwaltet die Ausgaben der Ampel. Ein Beispiel:

[TrafficLight] North and South Light is now GREEN
[TrafficLight] West and East Light is now RED

2.3 Straße - (street)

Aufgabe

Die **Straße** ist für die Auffüllung der Auto-Queue zuständig. Weiters "schickt" die Klasse die Autos über die Straße. Dies wird mit einer Ausgabe markiert. Ein Beispiel:

[CAR] AUDI HL-B7GF9 drives away from NORTH

Klassenstruktur

Listing 3: C++ street.h - Klassenstruktur

```
1  class Street
2  {
3  private:
4      TrafficLight* light;
5      Directions direction;
6      int generateAmount;
7      std::queue<Car>* carQueue = new std::queue<Car>();
8  public:
9      Street(int generateAmount, TrafficLight* light,
10             Directions direction);
11     void startStreet();
12     void fillCarQueue();
13 };
```

Die Struktur enthält die wichtige *carQueue*, die die Autos beinhaltet. Weiters enthält die Klasse die Menge - (Zahl) an nach-generierte Autos. Die Richtung in die die Autos fahren. Wie bei den Aufgaben schon erkärt. Die beiden

Funktionen sind am Namen selbst erklärend, startStreet ist für die Ausgabe der fahrenden Autos zuständig bzw. löscht diese nach der Überfahrt aus der Queue heraus. Die Methode fillCarQueue füllt die Queue mit generierten Autos.

2.4 Auto - (car)

Aufgabe

Die **Auto** Klasse stellt das generierte Auto zur Verfügung. Diese beinhaltet das Kennzeichen, die Automarke und die Zeit wie lang das Auto benötigt um über die Straße zu fahren.

Klassenstruktur

Listing 4: C++ car.h - Klassenstruktur

```
1  class Car
2  {
3  private:
4      std::string name;
5      std::string licensePlate;
6      int speed;
7  public:
8      static Car generateCar();
9
10     Car(std::string name, std::string licensePlate, int speed) {
11         this->name = name;
12         this->licensePlate = licensePlate;
13         this->speed = speed;
14     }
15
16     std::string getLicensePlate();
17     std::string getName();
18     int getSpeed();
19 };
```

Wie auch bei der Aufgabe beschrieben stellt die Klassenstruktur das generierte *Auto* und ihre *Eigenschaften* zur Verfügung. Die Methode generateCar

erstellt das Auto, dieses beinhaltet die zufällig generierten Eigenschaften. Zur der genauen Erklärung kommen wir später: 4.2

3 Bedienung

Zur besseren Bedienung wurde die Header-only Datei CLI11.hpp verwendet. Diese implementiert ein userfreundliches Kommandozeilen Interface über das man das Programm konfigurieren kann. Alle eingegebenen Werte werden vor dem Start der Simulation überprüft und falls notwendig auch Fehler geworfen.

3.1 –help, -h

Mittels ./project -h oder –help, werden die nötigen Parameter bzw. Funktionen angezeigt. In meinem Beispiel würde die Ausgabe folgendermaßen Aussehen:

Listing 5: Ausgabe - –help, -h

```
1  TrafficLight-Simulation
2  Usage: ./project [OPTIONS] cars respawntime
3
4  Positionals:
5    cars INT:NUMBER REQUIRED
6    How many cars after each respawn time respawns
7
8    respawntime INT:NUMBER REQUIRED
9    The time interval in which new cars spawn
10
11 Options:
12   -h,--help
13   Print this help message and exit
```

3.2 cars

Verpflichtend ist die Eingabe von **cars**. Damit ist die Menge an Autos gemeint, die nach einer gewissen Zeit wieder an die Straße kommen. Dieser Parameter muss ein Integer sein.

3.3 respawntime

Die **respawntime** ist ebenfalls verpflichtend. Diese regelt die Zeit, in der Autos generiert werden. Zum Beispiel: ./projects 1 10 würde jede 10 sek ein Auto generieren lassen. Dieser Parameter muss ein Integer sein.

4 Zustatzfunktion

4.1 Logger

Der **Logger** ist für das loggen bestimmter Funktionen zuständlich. Diese Informationen schreibt die Funktion in eine externe Datei hinein. Der Dateiname besteht aus dem log_ und den aktuellen Tag, an den der Logger fungiert hat. Ein Beispiel: log_2021-02-10.txt. Sollte an dem Tag der Logger bereits geschrieben haben, wird diese Datei erweitert. Um solche Logs zu unterscheiden steht in der Datei die aktuelle Uhrzeit des Logs. Diese Datei wird im Log Ordner abgespeichert. Am folgenden Beispiel können Sie sehen, wie der Logger aufgebaut ist:

Listing 6: C++ Logger Funktionen

```
1
2  inline std::string getCurrentTime(std::string s){
3      time_t now = time(0);
4      struct tm tstruct;
5      char buf[80];
6      tstruct = *localtime(&now);
7      if(s=="now"){
8          strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);
9      }else if(s=="date"){
10         strftime(buf, sizeof(buf), "%Y-%m-%d", &tstruct);
11     }
12     return std::string(buf);
13 }
14
15 inline void logger(std::string logMsg){
16     std::string filePath = "../log/log_" +
17                                getCurrentTime("date") + ".txt";
18     std::string now = getCurrentTime("now");
19     std::fstream ofs;
20     ofs.open(filePath, std::ios_base::out | std::ios_base::app );
21     if(!ofs.good()){
22         std::cout << "Logging has a problem!";
23     }
24     ofs << now << '\t' << logMsg << '\n' << std::flush;
```

```
25      ofs.close();
26  }
```

Die Methode **getCurrentTime** stellt jeweils einen String zur Verfügung der entweder den aktuellen Tag oder die aktuelle Uhrzeit zurück gibt.

Die Methode **logger** erstellt die Datei mit der Verwendung der vorherigen Methode und schreibt den angegeben Parameter in diese Datei hinein.

4.2 Auto - Generierung

Die Auto Generierung erfolgt in der Car Klasse siehe: 2.4.

Besonders dabei ist die Generierung des **Kenzeichen**, da erstens alle Bezirksabkürzungen des Bundeslandes Niederösterreich zur Verfügung stehen. Weiters wird das Kennzeichen mittels einer Funktion die zufällige Zeichenketten zurückgibt erstellt. Im folgenden Code Stück sehen Sie wie dies funktioniert:

Listing 7: C++ Funktion zur Erstellung des Kennzeichens

```
1  string random_string( size_t length )
2  {
3      auto randchar = []() -> char
4      {
5          const char charset[] =
6          "0123456789"
7          "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
8          const size_t max_index = (sizeof(charset) - 1);
9          return charset[ rand() % max_index ];
10     };
11     string str(length,0);
12     generate_n( str.begin(), length, randchar );
13     return str;
14 }
```

Die Ermittlung der **Marke** erfolgt durch einen zufälligen Zugriff auf einen Vector der mit Automarken befüllt ist.

Die Ermittlung der **Zeit**, dass das Auto für die Überquerung benätigt wird von einer Zufälligen Zeit bestimmt. Diese liegt zwischen 1 und 2.5 Sekunden.

References

[Wil] Arnold Willemer. *Aufzählungstyp enum*. Besucht am 09.02.2021. URL: `willemer.de/informatik(cpp/enum.htm`.