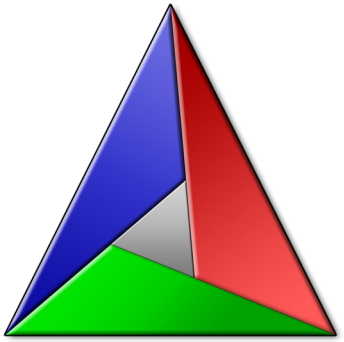
Hugo Martel

Projet de C++, Simulation de Course

JUNIA ISEN


Vim



Démo


003049
Prussian Blue
D62828
Maximum Red
F77F00
Orange
FCBF49
Maximum Yellow Red
EAE2B7
Lemon Meringue

vim coureur.cpp

```cpp
/*------------
 SPEED UPDATE
------------*/
int Coureur::updateSpeed(const Parcours& p) {
    if (currentCheckpoint != p.getCheckpointAmount()) {
        //Calculate the actual feltWindSpeed for the runner : front/back wind
        float anglesRelativePosition = std::abs(p.getWindDirection() - p.getAngle(currentCheckpoint));
        float feltWindSpeed = 0;

        if (anglesRelativePosition <= 45)
            feltWindSpeed = -p.getWindStrength();//wind is boosting
        else if (anglesRelativePosition <= 225 and anglesRelativePosition >= 135)
            feltWindSpeed = p.getWindStrength();//wind is an obstacle


        //Simplified formula
        speed = averageSpeed + ALPHA * (height/mass) * (std::pow(averageSpeed, 3) - speed*std::pow(speed+feltWindSpeed, 2));
        //p.getWindStrength()

        float slope = p.getSlope(currentCheckpoint);
        if (slope < 0) {
            // SPEED BOOST -> 0.35% per 1.5% slope
            speed *= 1 + 0.0035 * std::abs(slope / 1.5);
        } else {
            // SPEED REDUCED -> 1% per 1.5% slope
            speed *= 1 - 0.0015 * (slope / 1.5);
        }


        // speed deduced from the shoeWeight 1.1e-4 deduced per gram
        speed *= 1 - ((int)(shoeWeight - 100.0))*0.00011;


        // speed deduced from the hydration
        if (hydrationImpactOnSpeed <= 0.9 and hydrationImpactOnSpeed >= 0.4) {
            speed *= 0.99 - 0.38*(0.9-hydrationImpactOnSpeed);
        }


        // speed deduced from the prepWeeks
        if (distanceRan > p.getTotalDistance()/2) {
            speed *= 1 - ((2*distanceRan/p.getTotalDistance()) - 1)*((16-prepWeeks)*0.025);
        }
    }
    //std::cout << name << ": " << speed << "\n";//DEBUG
    return EXIT_SUCCESS;
}
```

NORMAL coureur.cpp unix | utf-8 | cpp 18% 58:9

Formule initiale :

$$v_{new} = \frac{P_{t_{max}} - P_a}{0.98M}$$

Après simplifications,

$$v_{new} = v_{moyenne} + \alpha \frac{h}{M} \left(v_{moyenne}^3 - v_{precedente} (v_{moyenne} - v_{vent})^2 \right)$$

Où $\alpha = 0.08422704082$

Soient d la distance totale du parcours, D la distance parcourue par le coureur, $D \in [\frac{d}{2}, d]$, s le nombre de semaines de préparation, $s \in [8, 16]$ et v la vitesse du coureur
L'équation régissant la diminution de vitesse en fonction du temps de préparation est telle que :

$$v_{reel} = v_{avant} \times \left(1 - \frac{(16 - s)2.5}{100} \times \left(\frac{2D}{d} - 1 \right) \right)$$

Soit en notation C++ :

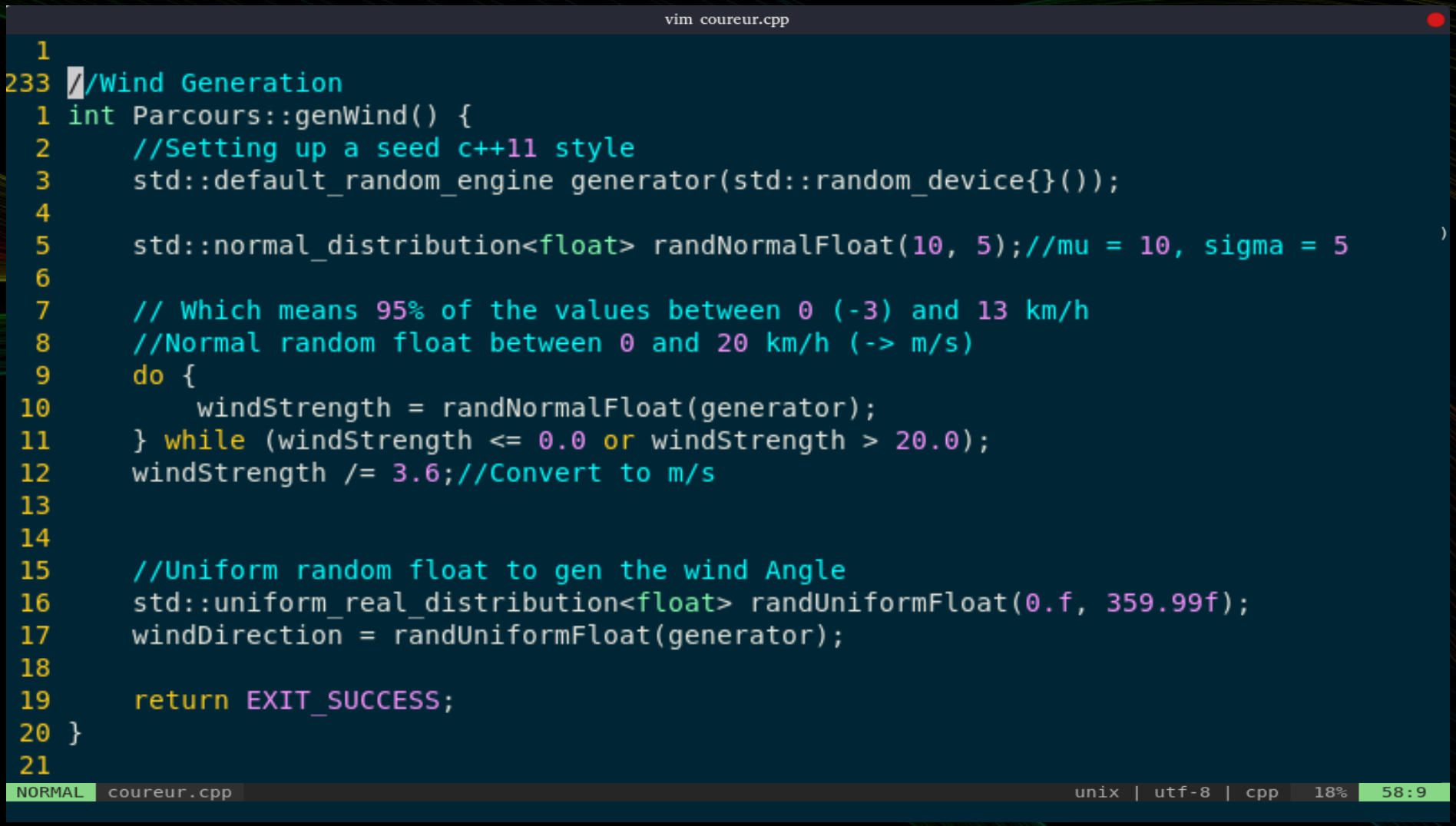
$$v* = 1 - (2 * D/d - 1) * ((16 - s) * 0.025);$$

Soit H le taux d'hydratation du coureur, la vitesse v du coureur diminue linéairement entre 1 et 20% lorque ce taux est sur $[0.4, 0.9]$.
On obtient alors la formule suivante :

$$v_{reel} = v_{avant} \times \left(1 - \left(0.01 + 0.19 \times \frac{0.9 - H}{0.9 - 0.4} \right) \right)$$

Soit en notation C++ :

$$v* = 0.99 - 0.38 * (0.9 - H)$$


vim coureur.cpp
1
233 //Wind Generation
1 int Parcours::genWind() {
2 //Setting up a seed c++11 style
3 std::default_random_engine generator(std::random_device{}());
4
5 std::normal_distribution<float> randNormalFloat(10, 5);//mu = 10, sigma = 5
6
7 // Which means 95% of the values between 0 (-3) and 13 km/h
8 //Normal random float between 0 and 20 km/h (-> m/s)
9 do {
10 windStrength = randNormalFloat(generator);
11 } while (windStrength <= 0.0 or windStrength > 20.0);
12 windStrength /= 3.6;//Convert to m/s
13
14
15 //Uniform random float to gen the wind Angle
16 std::uniform_real_distribution<float> randUniformFloat(0.f, 359.99f);
17 windDirection = randUniformFloat(generator);
18
19 return EXIT_SUCCESS;
20 }
21
NORMAL coureur.cpp
unix | utf-8 | cpp 18% 58:9

s = 5

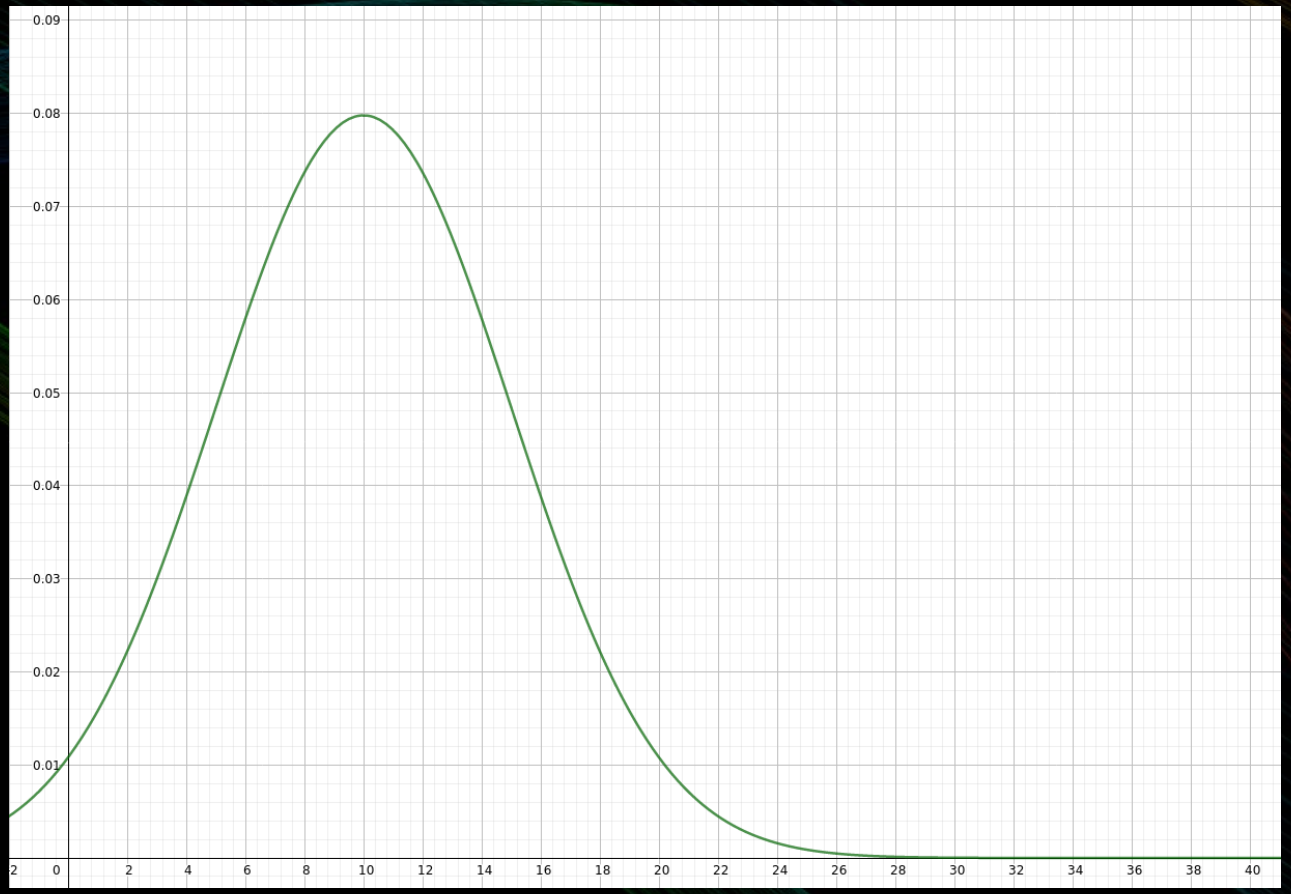
-5

m = 10

-5

$$f(x) = \frac{1}{s\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2}$$

$$\rightarrow \frac{1}{5\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-10}{5}\right)^2}$$

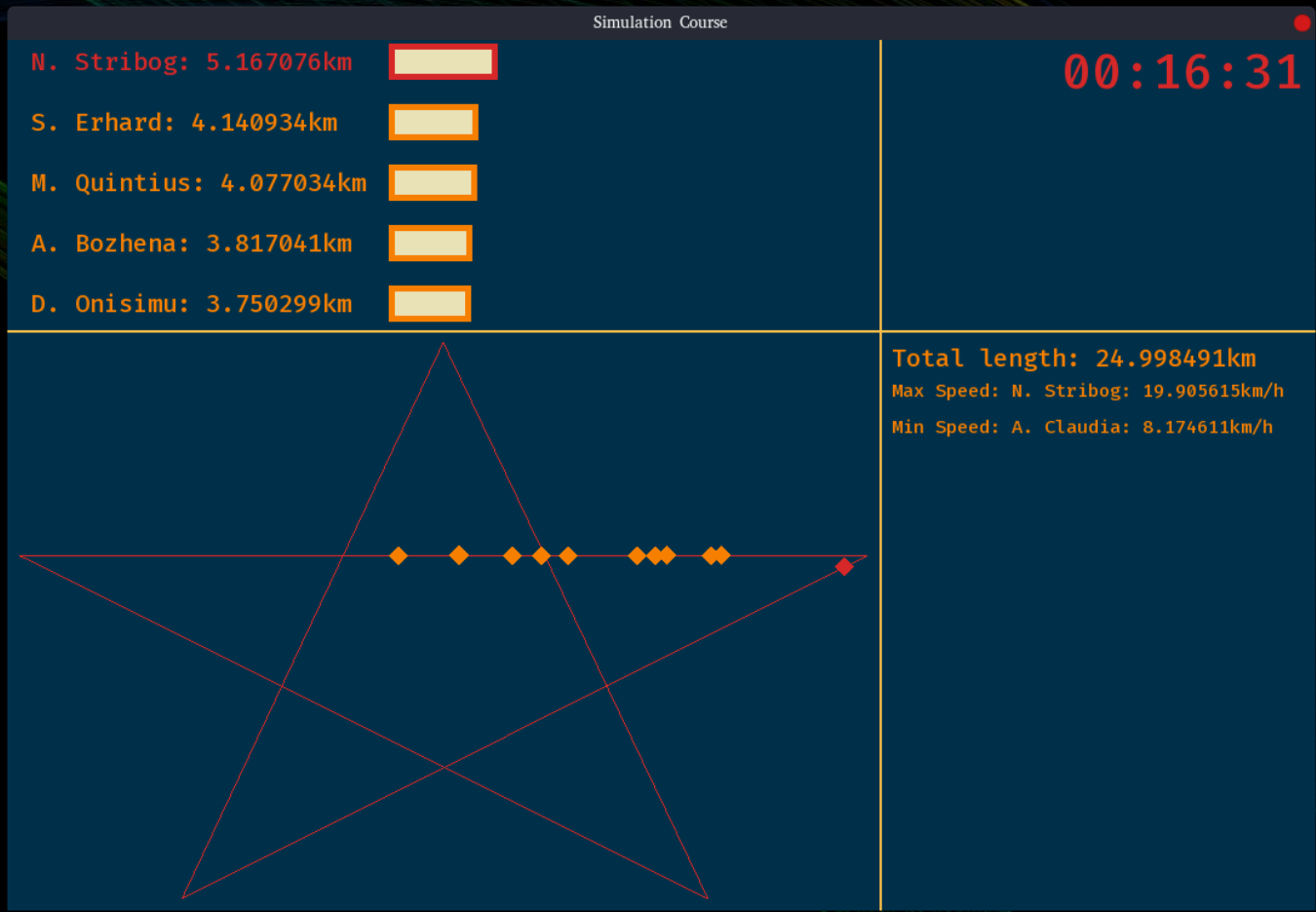


```
6
{
0
1.82
0
1
},
{
5
1.82
0
1
},
{
0.96
4.75
0
1
},
{
2.5
0
0
1
},
{
4.06
4.75
0.2
1
},
{
0
1.82
0
1
}
```

Soit P_n le point du n-ième checkpoint sur l'affichage du parcours, p le point du coureur sur le parcours, D_n la distance jusqu'au n-ième checkpoint et d la distance parcourue par le coureur.

On peut alors calculer la position du coureur ayant passé le n-ième checkpoint sur le parcours affiché à l'aide de la formule suivante :

$$p = P_n + \frac{P_{n+1} - P_n}{D_{n+1} - D_n} \times (d - D_n)$$



Hugo Martel

Projet de C++, Simulation de Course

JUNIA ISEN