`class Vector2 final`

Opis:

Klasa umożliwiająca pracę na płaszczyźnie wektorowej opisanej przez oś poziomą X rosnącą od lewej do prawej, oraz pionową Y rosnącą od góry do dołu.

Pola:

`static const Vector2 ZERO`
Wektor zerowy [0 0].

`static const Vector2 ONE`
Wektor jedynkowy [1 1].

`static const Vector2 RIGHT`
Wektor kierunku prawego [1 0].

`static const Vector2 DOWN`
Wektor kierunku dolnego [0 1].

`static const Vector2 LEFT`
Wektor kierunku lewego [-1 0].

`static const Vector2 UP`
Wektor kierunku górnego [0 -1].

`float x`
Współrzędna pozioma wektora.

`float y`
Współrzędna pionowa wektora.

Metody:

`static Vector2 towards(float angleDeg)`
Zwraca wektor kierunku prawego [1 0] obrócony o *angleDeg* stopni zgodnie ze wskazówkami zegara.

`Vector2()`
Tworzy instancję wektora zerowego.

`Vector2(float x, float y)`
Tworzy instancję wektora o podanej współrzędnej poziomej *x* oraz pionowej *y*.

`Vector2(const Vector2& other)`
Tworzy instancję wektora kopiując właściwości podanego wektora *other*.

`Vector2 operator+(const Vector2& other) const`
Zwraca wektor wynikowy sumy wektorów obecnego oraz *other*.

`Vector2 operator-(const Vector2& other) const`
Zwraca wektor wynikowy różnicy wektorów obecnego oraz *other*.

`Vector2 operator*(float scalar) const`
Zwraca wektor wynikowy skalowania obecnego wektora przez skalar *scalar*.

`Vector2 operator/(float scalar) const`
Tak samo jako operator `*`, zwraca efekt skalowania obecnego wektora przez skalar *scalar* ale w sposób odwrotny.

`Vector2 operator=(const Vector2& other)`
Kopiuje właściwości podanego wektora *other* do obecnego.

`bool operator==(const Vector2& other) const`
Zwraca informację, czy podany wektor *other* ma takie same właściwości jak obecny.

Funkcje:

`Vector2 operator*(float scalar, const Vector2& other)`
Dokonuje identycznych operacji co operator instancji `*`. Celem tej funkcji jest umożliwienie mnożenia lewostronnego dowolnego wektora *other* przez skalar *scalar*.

`std::ostream& operator<<(std::ostream& os, const Vector2& v)`
Przekazuje reprezentację tekstową wektora *v* w formacie [<x> <y>] do określonego strumienia wyjściowego *os*.

`std::istream& operator>>(std::istream& is, Vector2& v)`
Inicjuje właściwości wekora *v* danymi zczytanymi z określonego strumienia wejściowego *is*.

Dokumentacja techniczna narzędzia *Terminal Clock*

`struct DataPoint`

Opis:

Struktura opisująca pojedynczy punkt na płaszczyźnie. Punkt złożony jest z dwóch znaków kodu ASCII.

Pola:

`uint8_t left`
Lewa część punktu, pierwsza składowa danej.

`uint8_t right`
Prawa część punktu, druga składowa danej.

`class Plane`

Opis:

Klasa pozwalająca na dokonywanie podstawowych operacji ustawiania i zczytywania danych z płaszczyzny o określonej wielkości. Płaszczyzna zbudowana jest z instancji struktury `DataPoint`. Można ją przedstawić w formie graficznej za wykorzystaniem znaków ASCII.

Metody:

`Plane(uint8_t width, uint8_t height, std::ostream& receiver = std::cout)`
Tworzy instancję płaszczyzny o określonej szerokości *width* oraz wysokości *height* (wyrażanych w punktach danych). Podany strumień wyjściowy *receiver* uważany jest za odbiornik reprezentacji graficznej płaszczyzny – jest mu ona wysyłana za wolą użytkownika klasy, stosując metodę `show`.

`virtual ~Plane() = default`
Usuwa instancję płaszczyzny w sposób kontrolowany. To rozwiązanie umożliwia klasie dziedziczącej wprowadzenie własnego destruktora.

`DataPoint get(int x, int y)`
`DataPoint get(float x, float y)`
`DataPoint get(Vector2 v)`
Zwraca punkt danych obecny na współrzędnych określonych w sposób separatywny (parametry *x* oraz *y*) albo wektorowy (parametr *v*).

`uint8_t height()`
Zwraca szerokość płaszczyzny w punktach danych.

`uint8_t width()`
Zwraca wysokość płaszczyzny w punktach danych.

`void set(DataPoint dp)`
`void set(uint8_t data)`
Ustawia wszystkie punkty danych płaszczyzny na określoną wartość. Jeśli wybrany został wariant z parametrem *data*, zostanie to zinterpretowane jako wola ustawienia lewej i prawej składowej punktów danych na tę samą wartość *data*.

`void set(int x, int y, DataPoint dp)`
`void set(float x, float y, DataPoint dp)`
`void set(Vector2 v, DataPoint dp)`
`void set(int x, int y, uint8_t data)`
`void set(float x, float y, uint8_t data)`
`void set(Vector2 v, uint8_t data)`
Ustawia punkt danych na współrzędnych określonych w sposób separatywny (parametry *x* oraz *y*) albo wektorowy (parametr *v*), na określoną wartość *dp* albo *data* (wtedy składowe punktu danych przyjmują tę samą wartość).

`void show()`
Przekazuje strumieniowi wyjściowemu podanemu przy konstrukcji instancji ciągi znaków w taki sposób, że efekt tego przekazania graficznie zaprezentuje płaszczyznę przez interpretację jej punktów danych jako par znaków ASCII.

Dokumentacja techniczna narzędzia *Terminal Clock*

`class Clock : protected Plane`

Opis:

Klasa wykorzystująca klasę płaszczyzny `Plane` jako emulację ekranu na buforze konsoli, w celu prezentacji analogowego zegara z trzema wskazówkami (sekundnik, minutnik i godzinnik) ustawionego na aktualny czas zegarowy.

Metody:

`static std::time_t getEpochTime()`
Zwraca ilość sekund, która upłynęła od dnia 01.01.1970 r.

`Clock(uint8_t diameter)`
Tworzy instancję zegara o średnicy *diameter* wyrażonej w punktach danych.

`void update()`
Czyści płaszczyznę i ustawia jej punkty danych tak, że całokształt przedstawia analogowy zegar skonfigurowany w sposób odwzorowywujący aktualny czas zegarowy obliczany na podstawie zwrotu metody `getEpochTime`.

`void show()`
Prezentuje zegar poprzez standardowy strumień wyjściowy, przyjmuje się że powinna być to w przeważającej ilości przypadków konsola.

Bartłomiej Wilk, rok 2025