Rozbudowa repertuaru figur

Kolejnym etap prac nad mapą będzie dodanie do repertuaru figur łamanej i uniezależnienie układu współrzędnych mapy od układu współrzędnych okna, w którym mapa jest wyświetlana.
Zaczynamy od utworzenia projektu na podstawie szablonu (u mnie jest to `graph_lib_prj`, który utworzyłem na ćwiczeniach 20 marca). Pożytek z tego szablonu sprowadza się w zasadzie do włączenia stosownych katalogów plików nagłówkowych i bibliotek oraz samych bibliotek.
Po utworzeniu katalogu `zad_2` na projekt, zmieniam jeszcze jego nazwę na `zad_2`. `test_graph.cpp` jest do usunięcia od razu. Do katalogu projektu kopiuję natomiast pliki z aktualnego pakietu (`figure_test.cpp` i `mapa_korona.txt`) oraz pliki z implementacją hierarchii figur z poprzedniego zadania (`figure.h` i `figure.cpp`). Błędów mam mnóstwo i trzeba będzie nieco zmodyfikować hierarchię figur.

Prace programistyczne

Prócz uzupełnienia hierarchii klas figur istotna będzie zmiana przekształcenia figur w kształty do wyświetlenia. Zadanie można podzielić na dwa niezależne kroki:

1. Wyznaczenie parametrów transformacji przejścia z układu współrzędnych mapy na współrzędne okna.
2. Zmodyfikowanie metody `get_shape` tak, żeby uwzględniała tę transformację.

Ja zacząłbym od punktu drugiego (bo bez tego pierwsza funkcja testowa daje morze błędów). Nowy prototyp funkcji w klasie bazowej powinien wyglądać tak:

```
virtual Graph_lib::Shape* get_shape(
  const FPoint& scale = {1.0f, 1.0f},
  const FPoint& trans = {0.0f, 0.0f}) const = 0;
```

Wartości domyślne parametrów odpowiadają przekształceniu tożsamościowemu.

Figury z transformacją

Sytuację mamy dość prostą. Dla ustalonych wartości skali i przesunięcia, do współrzędnych punktów definiujących figury trzeba zastosować te przekształcenia (w wymienionej kolejności).
Czyli:

```
p_new.x = p_old.x * scale.x + translation.x
p_new.y = p_old.y * scale.y + translation.y
```

Jest to w zupełności wystarczające dla łamanej (klasa `Line` – tylko jeszcze nie macie jej zaimplementowanej), ale nie dla prostokąta i koła. W przypadku prostokąta zakładam, że pierwszy punkt ma mniejsze współrzędne (tu wchodzą w paradę ujemne skale, a testy są już w pliku `figure_test.cpp`); dla koła trzeba obliczyć długość promienia po transformacji.

Wyznaczanie transformacji

Prototyp funkcji jest następujący:

```
std::pair<FPoint, FPoint> get_transformation(
            const std::pair<FPoint, FPoint>& obj_bbox,
            const std::pair<FPoint, FPoint>& disp_bbox);
```

Para, która stanowi wynik zawiera na pierwszej pozycji (`first`) skalę, a na drugiej (`second`) przesunięcie.

W zasadzie nie ma tu większych problemów: chodzi o to, żeby po zastosowaniu wynikowych transformacji dopasować `obj_bbox` do `disp_bbox`. Ale uwaga: skalowanie ma być proporcjonalne, a dokładniej `scale.x = -scale.y` ze względu na odwrócenie osi Y w obu układach współrzędnych.

Skalę wyznacza się jako iloraz szerokości `disp_bbox` i `obj_bbox` lub iloraz ich wysokości (mniejszy z nich). Przesunięcie to różnica między środkiem `disp_bbox` i środkiem przeskalowanego `obj_bbox`.

Prostokąt otaczający mapy

Prostokąt otaczający mapę (`obj_bbox` z poprzedniej strony) jest unią wszystkich prostokątów otaczających figury mapy. Prototyp funkcji jest następujący:

```
std::pair<FPoint, FPoint> map_bbox(
        const std::vector<figure*>& figures);
```

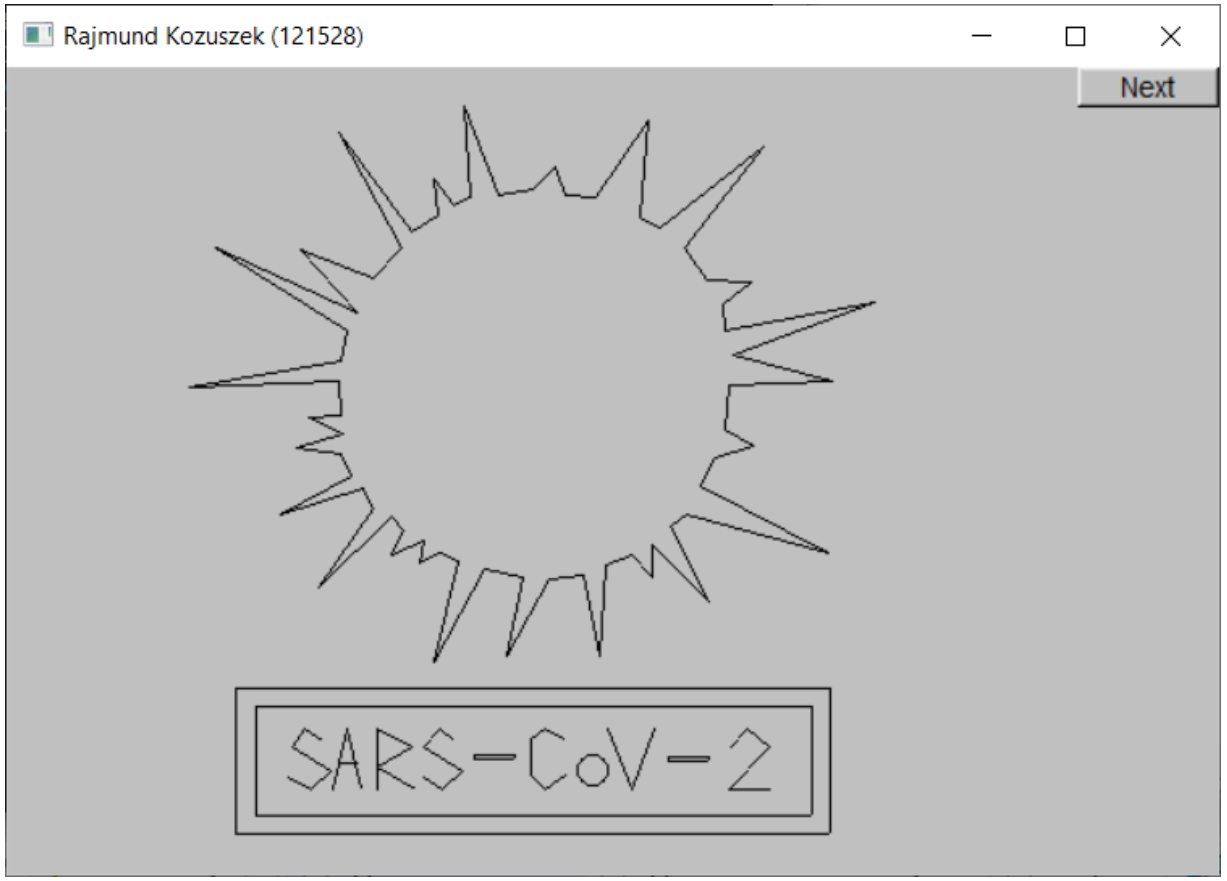
Ta funkcja jest na tyle prosta, że nie wymaga komentarza. Ostatnim elementem, który zostaje do napisania jest klasa `Line`. Z nią też nie powinno być problemów, bo wszystkie elementy składowe mieliście Państwo okazję przećwiczyć dwukrotnie.

Prócz wymienionych jawnie funkcji warto pomyśleć o uzupełnieniu operacji na `FPoint` tak, żeby nowy kod był krótki i czytelny. (W `figure.h` umieszczacie tylko nagłówki funkcji; ich implementacje powinny znaleźć się w `figure.cpp`).

Efekt końcowy

Do zakończenia będzie jeszcze potrzebna funkcja dająca tytuł okna.

Efekt końcowy powinien wyglądać tak:



Oddawanie

Definicje wszystkich klas i prototypy funkcji pomocniczych (globalnych) proszę umieścić w pliku `figure.h`. Implementacje funkcji globalnych i metod klas proszę umieścić w pliku `figure.cpp`.

Rozwiązania (`figure.h` i `figure.cpp`) należy złożyć w `Moodle` do:

2 kwietnia 2023 23:59