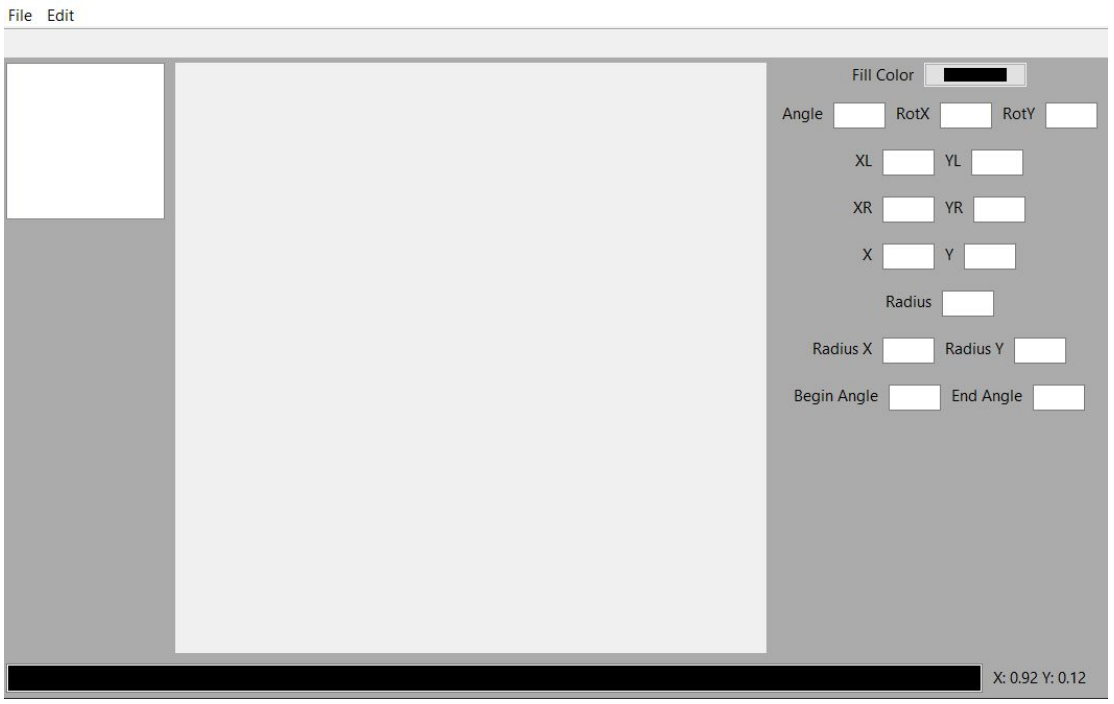
Projekt nr 38
INTERPRETER GRAFIKI WEKTOROWEJ

Maciej Bolsęga
Piotr Brunarski
Hubert Jakubek

1. Opis projektu.

Celem projektu jest nabycie umiejętności programowania dwuwymiarowej grafiki wektorowej. W ramach projektu powinien zostać stworzony program będący interpreterem rysunków wektorowych.

2. Założenia wstępne przyjęte w realizacji projektu.



Interfejs graficzny składa się z:

- jednego dużego obszaru roboczego widocznego na środku, który umożliwia wyświetlenie figur 2D
- konsoli - czarnego paska widocznego na dole
- po lewej stronie znajduje się obszar, w którym są wyświetlane obiekty wraz z numerem id
- po prawej stronie znajduje się obszar, w którym wyświetlane są aktualne parametry zaznaczonego obiektu, który można zaznaczyć poprzez kliknięcie myszką na obszarze wyświetlającym obiekty.
- na pasku menu znajdują się elementy File oraz Edit. File umożliwia nam zapisanie do pliku, zczytanie z pliku oraz zapisanie danych do obrazka, Edit posiada 2 komendy: Range oraz clear all.
- w prawym dolnym rogu obok panelu komend znajduje się aktualne położenie myszki
- po kliknięciu prawym przyciskiem myszy na obiekt w panelu obiektów, wyświetla się panel do usunięcia danego obiektu.

Oś współrzędnych zaczyna się w lewym górnym rogu.

Oczekujemy, że w głównym panelu będą wyrysowywane kształty 2D oraz w panelu po prawej stronie będą pokazywać się aktualne wartości obiektów.

3.Analiza projektu.

Dane wejściowe i dane wyjściowe:

Dane wejściowe są wpisywane w konsoli programu, dane wyjściowe są wyświetlane na odpowiednich panelach. Do dyspozycji użytkownika jest kilka prostych komend:

- **range x1 y1 x2 y2** Przypisuje lewemu dolnemu narożnikowi obszaru roboczego współrzędne (x1,y1), a prawemu górnemu (x2,y2).
- **background c** Ustala kolor tła na c.
- **line x1 y1 x2 y2 c** Rysuje linię od punktu (x1,y1) do punktu (x2,y2) i o zadanym kolorze c.
- **rectangle x1 y1 x2 y2 c** Rysuje prostokąt, którego lewy dolny narożnik znajduje się w punkcie (x1,y1), a prawy górny w punkcie (x2,y2) i o zadanym kolorze c.
- **circle x y r c** Rysuje okrąg o środku w punkcie (x,y) i promieniu r i o zadanym kolorze c.
- **ellipse x y rx ry c** Rysuje elipsę o środku w punkcie (x,y) oraz poziomej i pionowej półosi równej rx i ry oraz o zadanym kolorze c.
- **arc x y rx ry b e** Rysuje fragment łuku opartego na elipsie o środku w punkcie (x,y) i półosiach rx i ry. Łuk rozpoczyna się przy kącie b (wyrażonym w stopniach i liczonym przeciwnie do ruchu wskazówek zegara) a kończy przy kącie e.
- **fill id c** Wypełnia obiekt o identyfikatorze id kolorem c.
- **delete id** Usuwa obiekt o identyfikatorze id.
- **move id x y** Przesuwa obiekt o identyfikatorze id o wektor (x,y).
- **rotate id x y a** Obraca obiekt o identyfikatorze id o kąt a (wyrażony w stopniach, w kierunku przeciwnym do ruchu wskazówek zegara) wokół punktu o współrzędnych (x,y).
- **show id** Zaznacza na krótko (np. pół sekundy) obiekt o identyfikatorze id.
- **clear** Usuwa wszystkie obiekty i wykonuje komendę: range 0 0 1 1.
- **write file** Zapisuje wszystkie obiekty wraz z aktualnymi wartościami zakresu roboczego do pliku o nazwie file.
- **read file** Wczytuje wszystkie obiekty zawarte w pliku file oraz ustawia zakres roboczy na wartości z pliku.
- **save w h file** Zapisuje aktualny obrazek w postaci bitmapy o szerokości w i wysokości h do pliku graficznego o nazwie file.

W każdej z powyższych komend użycie słówka all zamiast identyfikatora id spowoduje wykonanie danej komendy na wszystkich obiektach, na których jest to możliwe.

Struktury danych:

W naszym projekcie skorzystaliśmy z 2 wbudowanych struktur danych:

- **map** - struktura ta została wykorzystana do sprawnej obsługi terminala: wartością KEY jest zmienna typu string- nazwa komendy, a wartością mapowaną są wskaźniki do odpowiednich funkcji obsługujących komendy.
- **vector** - struktura w której w programie przechowywanę są wszystkie aktualnie wyrysowane obiekty. Podczas rysowania dodatkowego elementu lub usuwania go, wektor jest odpowiednio modyfikowany, a zawarte w nim obiekty ponownie rysowane na panelu.
- **vec** - struktura pozwalająca przechowywać współrzędne (x, y) oraz wykonywać na nich operację rotacji, skalowania i przesunięcia o wektor.

Podział projektu:

- Opracowanie logiki projektu i wykonanie jego podstawowej struktury
- Stworzenie klas obiektów.
- Implementacja terminala oraz systemu komend.
- Implementacja funkcji obsługujących poszczególne komendy (wyrysowywanie kształtów)
- Implementacja funkcji modyfikujących wyrysowane obiekty (rotate, move, fill)
- Implementacja funkcji zapisujących oraz zczytujących (write, read,save)
- Modyfikacja częściowo niedziałających funkcji wykrytych podczas pierwszych testów.
- Dalsze testowanie aplikacji
- Ostateczne poprawki w projekcie.
- Napisanie dokumentacji

Narzędzia programistyczne:

Projekt tworzony był w środowisku Windows. Kompilowano go w Microsoft Visual Studio. Projekt został stworzony na podstawie biblioteki wxWidgets. Gui zostało stworzone za pomocą narzędzia wxFormBuilder.

4. Podział pracy i analiza czasowa.

Projekt realizowaliśmy 4 tygodnie. Spotykaliśmy się wspólnie 2 razy w tygodniu na czas 2 godzin, podczas których stopniowo rozszerzaliśmy naszą aplikację. Poniżej znajduje się opis zrealizowanych problemów w danym tygodniu.

Tydzień 1.

- Zaprojektowanie struktury aplikacji oraz stworzenie podstawowego gui z użyciem wxFormBuilder.

Tydzień 2.

- Stworzenie klasy bazowej Shape oraz dziedziczących po niej klas reprezentujących poszczególne figury, których rysowanie program miał obsługiwać.
- Implementacja terminala oraz systemu komend.
- Implementacja funkcji obsługujących poszczególne komendy (wyrysowywanie kształtów)

Tydzień 3.

- Implementacja funkcji modyfikujących wyrysowane obiekty (rotate, move, fill)
- Implementacja funkcji zapisujących oraz zczytujących (write, read,save)
- Modyfikacja częściowo niedziałających funkcji wykrytych podczas pierwszych testów.

Tydzień 4.

- Dalsze testowanie aplikacji
- Ostateczne poprawki w projekcie.
- Napisanie dokumentacji

5. Opracowanie i opis niezbędnych algorytmów.

Rotacja:
Aby obliczyć położenie punktu po rotacji skorzystaliśmy z równań opisujących nowe położenie punktu po rotacji o kąt *a*:

$x' = x * cos(a) - y * sin(a)$

$y' = y * cos(a) + x * sin(a)$

Równania te zostały użyte w metodzie Rotate, która znajduję się w klasie **vec**.

Skalowanie:
Metoda Scale jest również częścią klasy **vec**, jej efektem jest przemnożenie współrzędnych *(x, y)* obiektu klasy **vec** przez odpowiadające im skale:

$x' = x * Sx$

$y' = y * Sy$

Przesunięcie o wektor:
Metoda Translate część klasy **vec**, jej efektem jest dodanie wektora *(Tx, Ty)* do współrzędnych *(x, y)* znajdujących się w obiekcie **vec**:

$x' = x + Tx$

$y' = y + Ty$

Proces rysowania figur:
Rysowanie każdej z figur zaczyna się od obliczenia skali x i y dla obecnego rozmiaru panelu roboczego. Następnie ustawiany jest kolor wypełnienia na podstawie informacji z klasy Shape. Ostatnim krokiem jest obliczenie aktualnej pozycji figury poprzez przesunięcie, rotację oraz skalowanie do rozmiaru obszaru roboczego używając obliczonych wcześniej skali i wyrysowanie figury odpowiednią metodą.

Rysowanie prostokąta, linii oraz okręgu:
Używane do tego są metody klasy wxDC, odpowiednio DrawPolygon, DrawLine oraz DrawCircle. Prostokąt nie jest rysowany za pomocą DrawRectangle, gdyż nie można wtedy uwzględnić jego rotacji.

Rysowanie elipsy/łuku:
Elipsa oraz fragment łuku tworzone są ze zbioru linii, które powstają w pętli po kącie na podstawie równania elipsy. Dla kolejnych wartości kąta *a* kolejne współrzędne linii obliczane są ze wzorów:

$x = x0 + w * cos(a)$

$y = y0 + h * sin(a)$

gdzie:

x0, y0 - współrzędne środka elipsy
w, h - szerokość i wysokość elipsy

Fragment łuku rysowany jest analogicznie, z drobną różnicą - kąt przyjmuje wartości od kąta początku łuku do kąta końca łuku zamiast od 0 do 360 stopni jak to jest w przypadku elipsy.

Zapis i odczyt:
Aby móc łatwo zapisywać obiekty do pliku (metoda write), z którego potem można je wyczytać (metoda read) i dalej je edytować stworzyliśmy bufor do którego są zapisywane wszystkie komendy rysujące obiekty lub je modyfikujące (pomijamy metody write, read, save). Następnie podczas wywołania metody write, bufor ten jest zapisywany do pliku. Natomiast przy wywołaniu metody read plik ten jest odczytywany i kolejne metody zawarte w kolejnych liniach pliku są wywoływane.

6. Kodowanie.

Obiekty:
Podstawowa struktura każdego obiektu mieści się w klasie Shape. Każda instancja Shape zawiera pola:

- color - zmienna typu wxColor, zawiera kolor wypełnienia obiektu
- showColor - zmienna typu wxColor, przechowująca kolor, jest używana do zaznaczania obiektów. Jest odwrotną wartością koloru.
- idText - zmienna typu string reprezentująca obiekt w panelu bocznym
- idStr - zmienna typu string, przechowująca wartość identyfikacyjną w postaci tekstu.
- id - zmienna typu int, przechowująca wartość identyfikacyjną w postaci liczby.
- x0 - współrzędna x lewego górnego rogu panelu roboczego
- x1 -współrzędna x prawego dolnego rogu panelu roboczego
- y0 -współrzędna y lewego górnego rogu panelu roboczego
- y1 -współrzędna y prawego dolnego rogu panelu roboczego
- transformX - współrzędna x wektora przesunięcia
- transformY - współrzędna y wektora przesunięcia
- rotateX - współrzędna x punktu wokół którego obiekt jest obrócony
- rotateY - współrzędna y punktu wokół którego obiekt jest obrócony
- rotateAngle - kąt obrotu
- lastId - id ostatnio stworzonego obiektu

Poniżej lista metod klasy Shape:

- Shape(wxColor _color) konstruktor tworzący nowy obiekt typu Shape, przypisujący mu id oraz wypełnia otrzymanym kolorem poprzez metode Fill.
- DrawShape - rysuje dany obiekt uwzględniając wielkość panelu rysowania
- Fill - wypełnia dany obiekt otrzymanym kolorem
- Show - służy do zmiany wartości show na true, przez co można podświetlić dany obiekt na krótki okres czasu.
- UpdatePanelSize - aktualizuje współrzędne panelu roboczego
- GetIdText - zwraca idText
- GetId - zwraca idStr
- Move - zmienia wartości przesunięcia obiektu
- Rotate - zmienia współrzędne wartości obrotu obiektu.

Po klasie bazowej Shape dziedziczą klasy reprezentujące konkretne kształty wyszczególnione poniżej.

RectagleShape - klasa reprezentująca prostokąt. Oprócz elementów zawartych w klasie bazowej posiada również pola:

- xl - współrzędna x lewego górnego rogu prostokąta w panelu roboczym
- xr - współrzędna x prawego dolnego rogu prostokąta w panelu roboczym
- yl - współrzędna y lewego górnego rogu prostokąta w panelu roboczym
- yr - współrzędna y prawego dolnego rogu prostokąta w panelu roboczym

Dodatkowe metody zawarte w klasie RectangleShape:

- RectangleShape - konstruktor wywołujący konstruktor klasy bazowej, przypisujący wartości współrzędnym rogów prostokąta oraz tworzący string będący reprezentacją obiektu RectangleShape, tj. “Rectangle + ‘id’”.
- DrawShape - nadpisana metoda wyrysowywjąca prostokąt, w której uwzględniona jest ewentualna rotacja obiektu w panelu

LineShape - klasa reprezentująca linię. Oprócz elementów zawartych w klasie bazowej posiada również pola:

- xl - współrzędna x początku linii w panelu roboczym
- xr - współrzędna x końca linii w panelu roboczym
- yl - współrzędna y początku linii w panelu roboczym
- yr - współrzędna y końca linii w panelu roboczym

Dodatkowe metody zawarte w klasie LineShape:

- LineShape - konstruktor wywołujący konstruktor klasy bazowej, przypisujący wartości współrzędnym początka i końca linii oraz tworzący string będący reprezentacją obiektu LineShape, tj. “Line + ‘id’”.
- DrawShape - nadpisana metoda wyrysowywjąca linię, w której uwzględniona jest ewentualna rotacja obiektu w panelu

CircleShape - klasa reprezentująca linię. Oprócz elementów zawartych w klasie bazowej posiada również pola:

- x - współrzędna x środka koła w panelu roboczym
- y - współrzędna y środka koła w panelu roboczym
- r - promień koła

Dodatkowe metody zawarte w klasie CircleShape:

- CircleShape - konstruktor wywołujący konstruktor klasy bazowej, przypisujący wartości środka koła, promienia oraz tworzący string będący reprezentacją obiektu CircleShape, tj “Circle + ‘id’”.
- DrawShape - nadpisana metoda wyrysowywjąca koło , w której uwzględniona jest ewentualna rotacja obiektu w panelu

EllipseShape - klasa reprezentująca elipsę. Oprócz elementów zawartych w klasie bazowej posiada również pola:

- x - współrzędna x środka elipsy w panelu roboczym
- y - współrzędna y środka elipsy w panelu roboczym
- xr - wartość poziomej półosi
- yr - wartość pionowej półosi

Dodatkowe metody zawarte w klasie EllipseShape:

- EllipseShape - konstruktor wywołujący konstruktor klasy bazowej, przypisujący wartości środka elipsy, półosi oraz tworzący string będący reprezentacją obiektu EllipseShape, tj. “Ellipse + ‘id’”.
- DrawShape - nadpisana metoda wyrysowywjąca elipsę, w której uwzględniona jest ewentualna rotacja obiektu w panelu

ArcShape - klasa reprezentująca łuk oparty na elipsie. Oprócz elementów zawartych w klasie bazowej posiada również pola:

- x - współrzędna x środka elipsy bazowej w panelu roboczym
- y - współrzędna y środka elipsy bazowej w panelu roboczym
- xr - wartość poziomej półosi elipsy bazowej
- yr - wartość pionowej półosi elipsy bazowej
- beginAngle, endAngle - zakres kątów w których jest rysowany łuk na podstawie elipsy, kąty wyrażone są w stopniach i liczone przeciwnie do wskazówek zegara.

Dodatkowe metody zawarte w klasie ArcShape:

- ArcShape - konstruktor wywołujący konstruktor klasy bazowej, przypisujący wartości środka elipsy bazowej, jej półosi, kąta początkowego, kąta końcowego oraz tworzący string będący reprezentacją obiektu ArcShape, tj. “Arc + ‘id’”
- DrawShape - nadpisana metoda wyrysowywjąca łuk , w której uwzględniona jest ewentualna rotacja obiektu w panelu

7.Testowanie, wdrożenie, raport i wnioski.

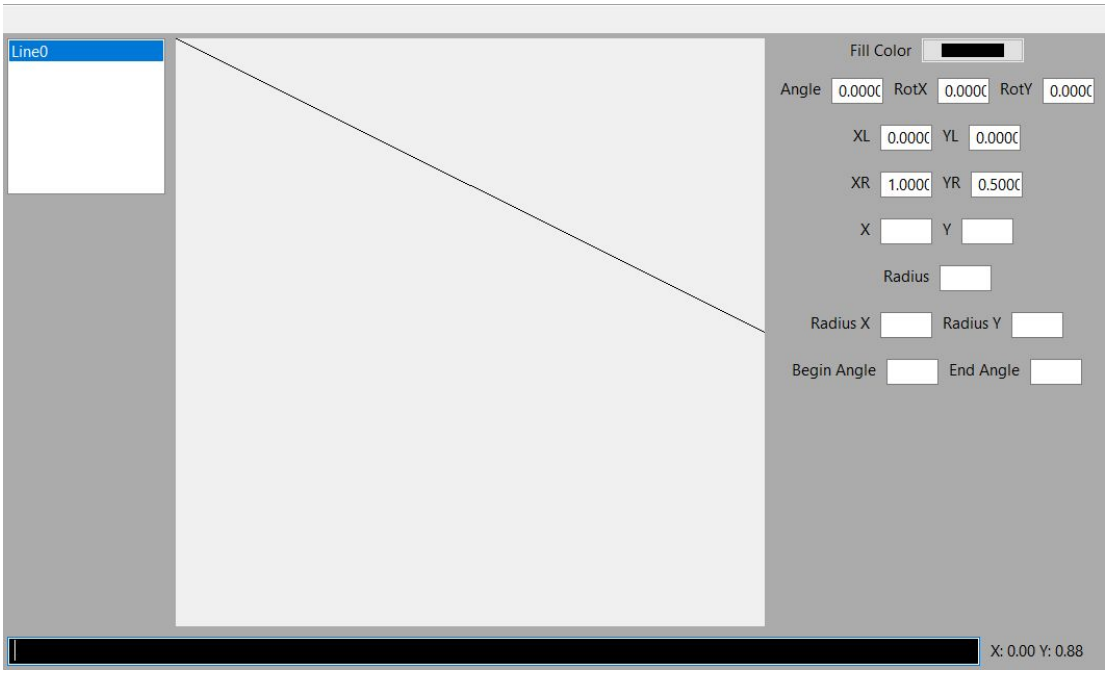
Testowanie

Nasz proces testowania polegał na używaniu komend i obserwowaniu poprawności ich wykonania. Przydatnym narzędziem okazała się komenda "test", która wywoływała komendy, które obecnie testowaliśmy. Komenda "test" pozwoliła nam na szybkie testowanie wyniku komend bez potrzeby wpisywania dłuższych nazw komend (jak rectangle) i pomijając ich argumenty.

Przykłady użycia komend:

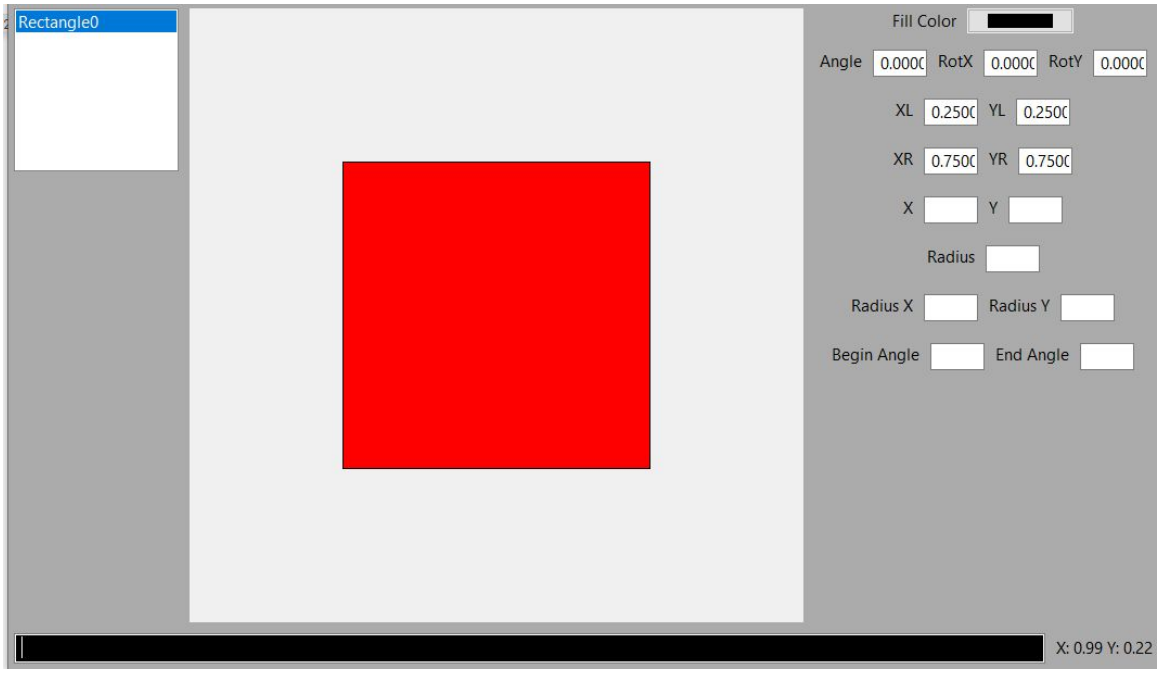
Line

Po poprawnym utworzeniu obiektu linii wyświetla się jej reprezentacja graficzna.



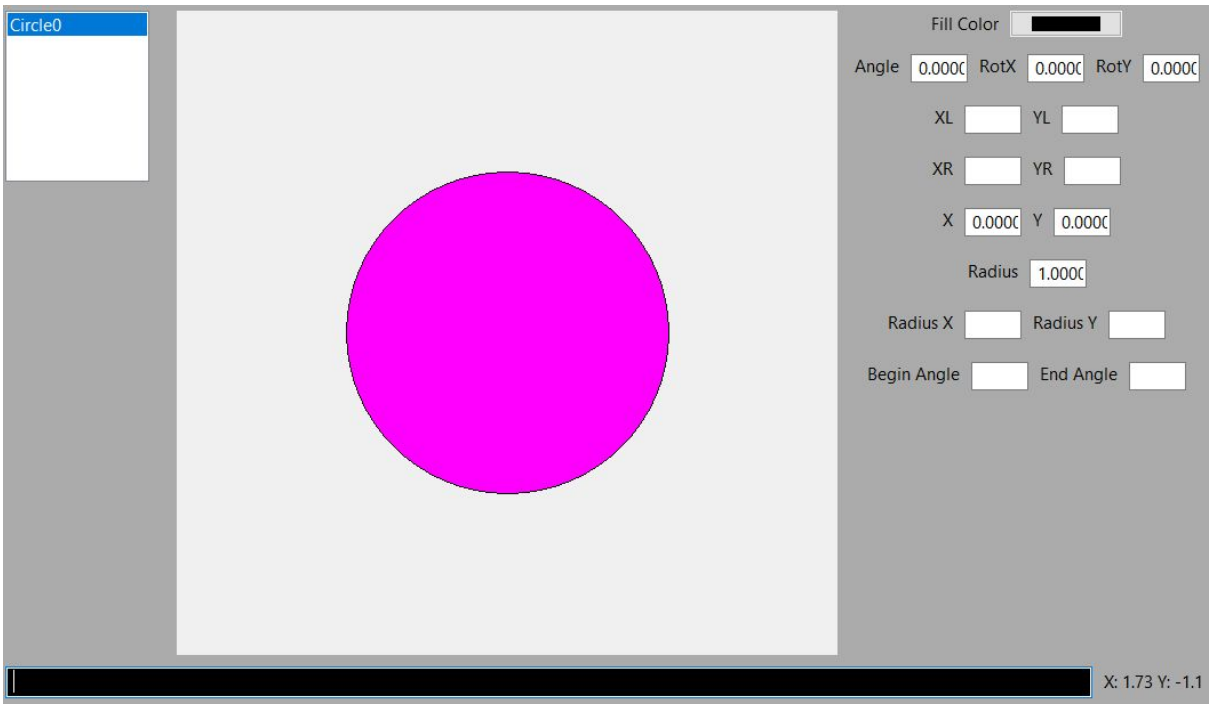
Rectangle

Po poprawnym utworzeniu obiektu prostokąta wyświetla się jego reprezentacja graficzna.



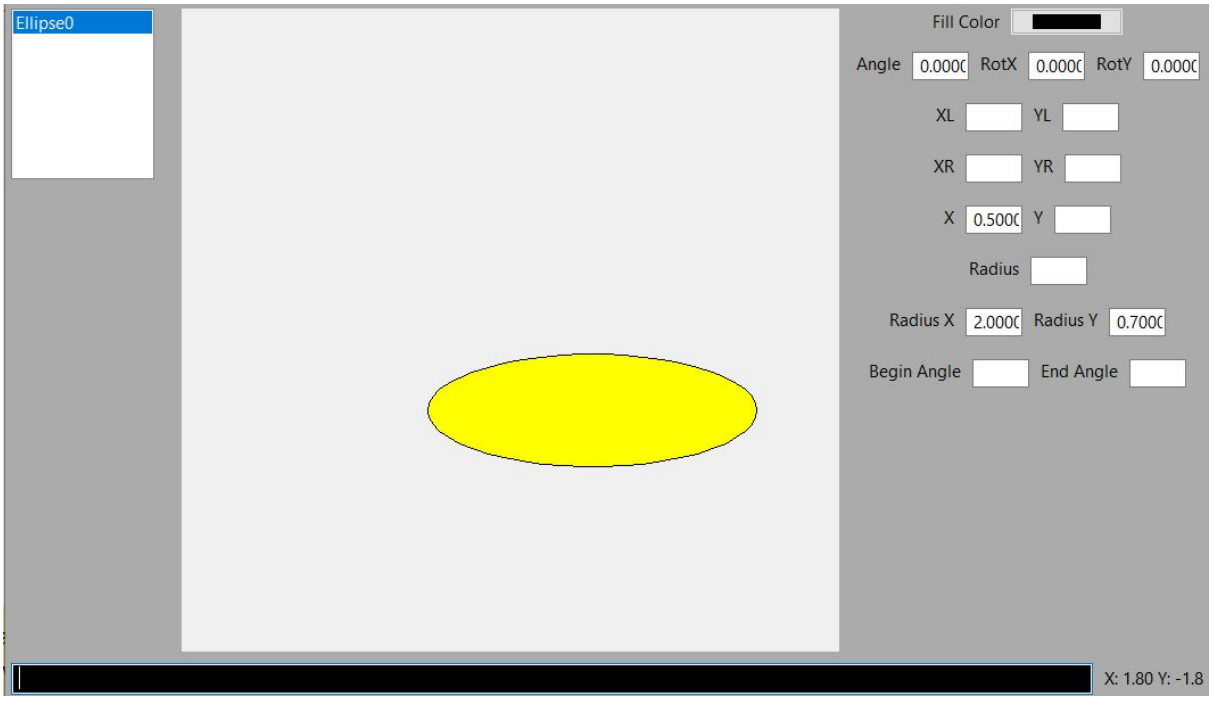
Circle

Po poprawnym utworzeniu obiektu okręgu wyświetla się jego reprezentacja graficzna.



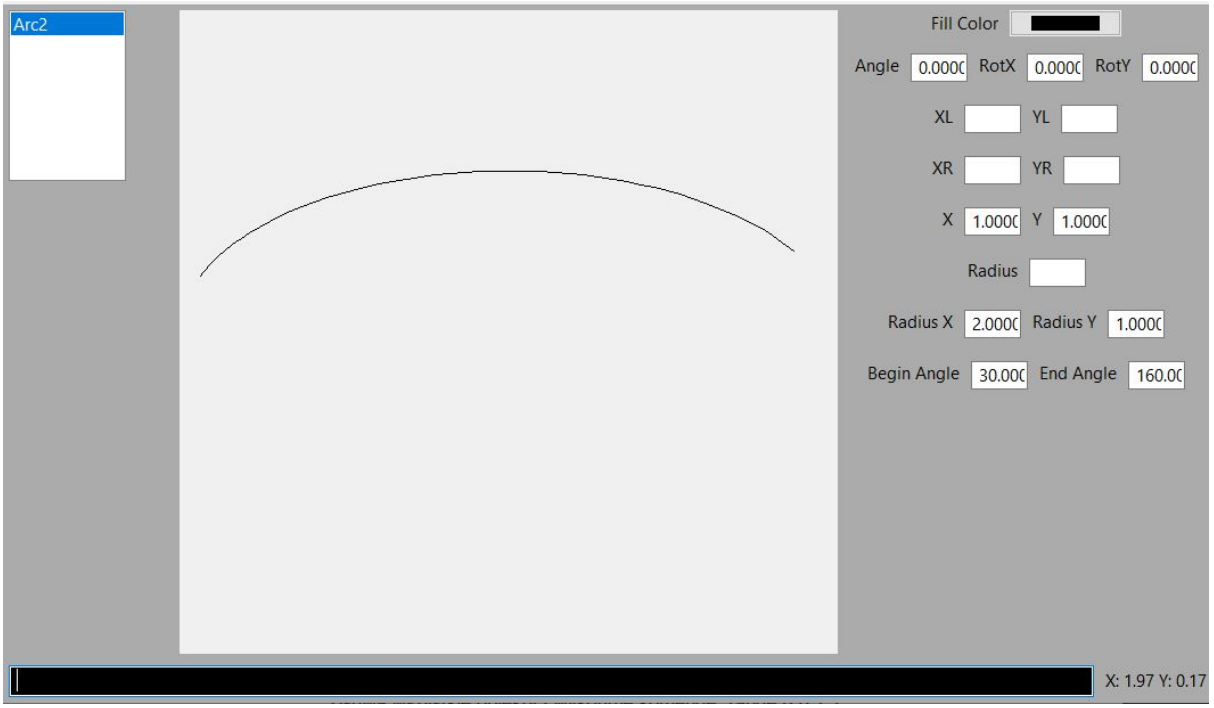
Ellipse

Po poprawnym utworzeniu obiektu elipsy wyświetla się jej reprezentacja graficzna.



Arc

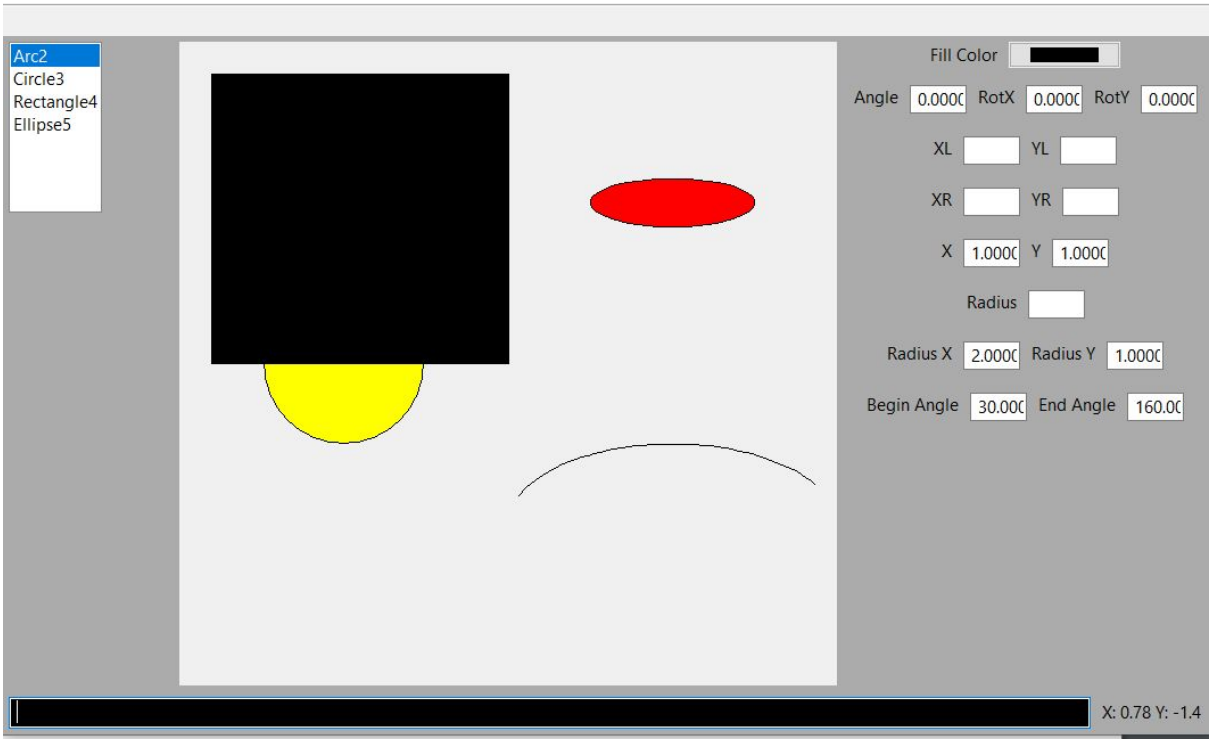
Po poprawnym utworzeniu obiektu łuku wyświetla się jego reprezentacja graficzna.



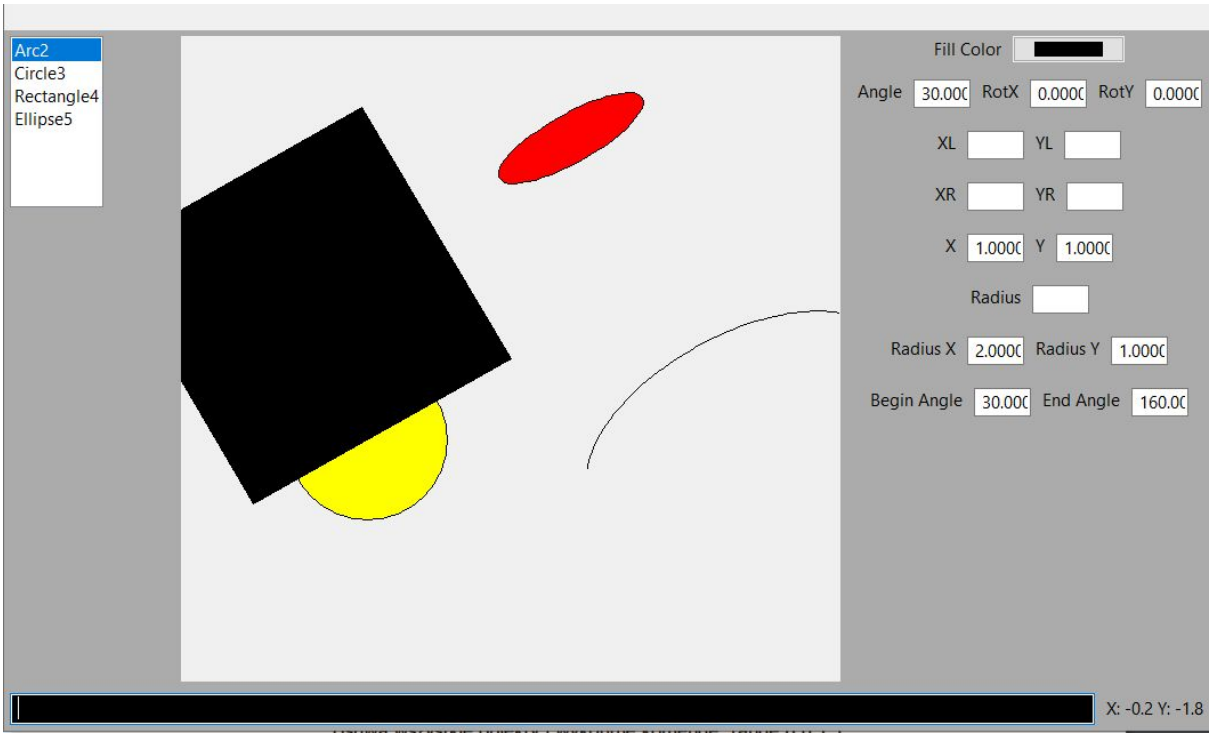
Rotate:

Obraca obiekty wokół podanego punktu o dany kąt.

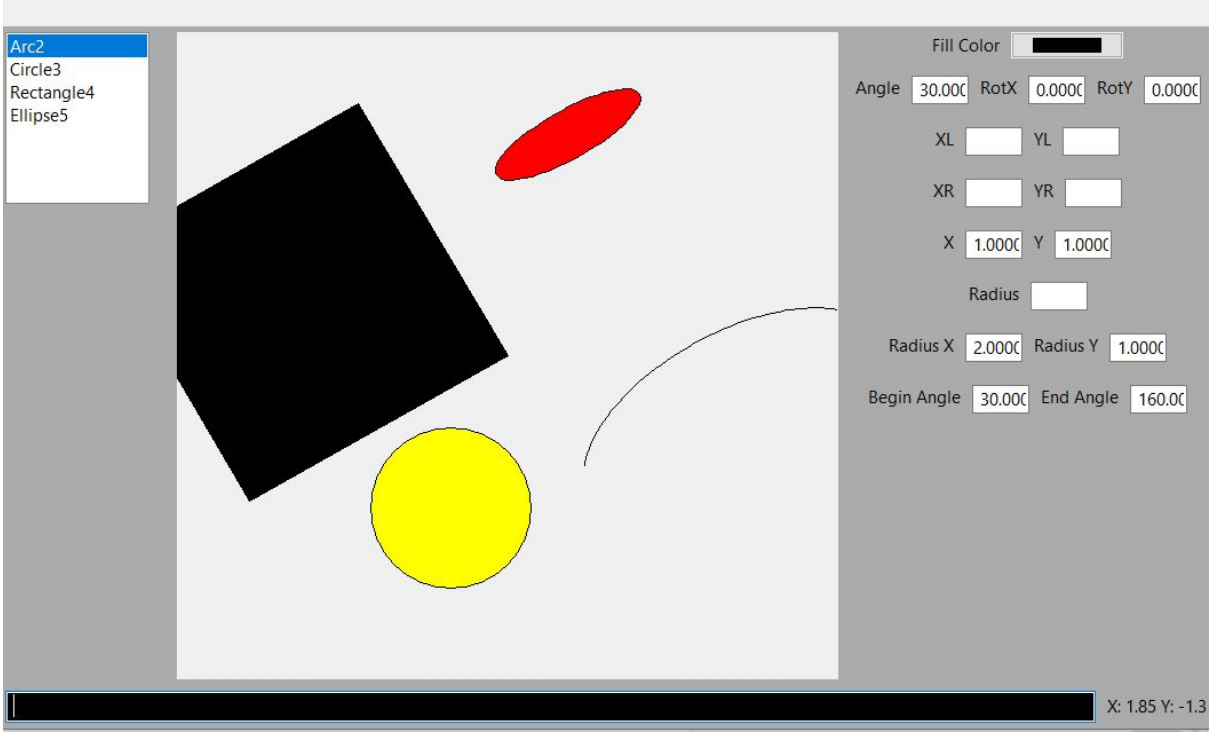
Przed użyciem funkcji Rotate dla losowego przykładu:



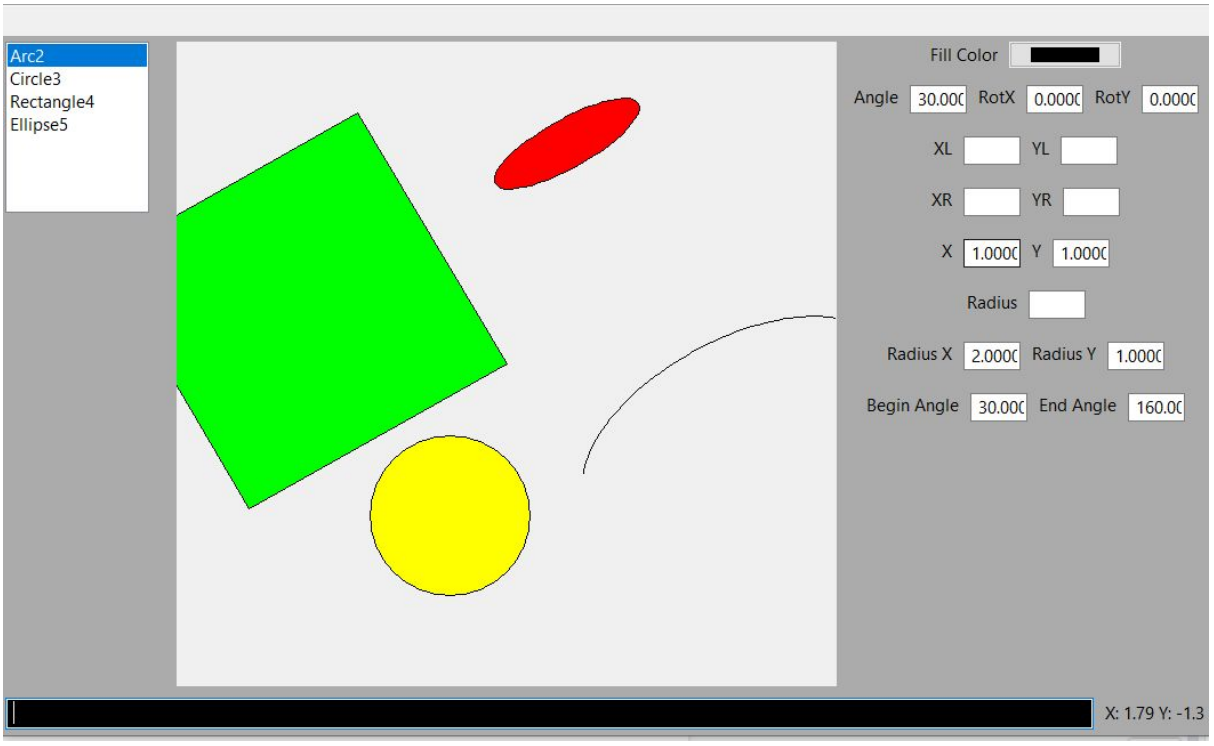
Po użyciu *rotate all 0 0 40*



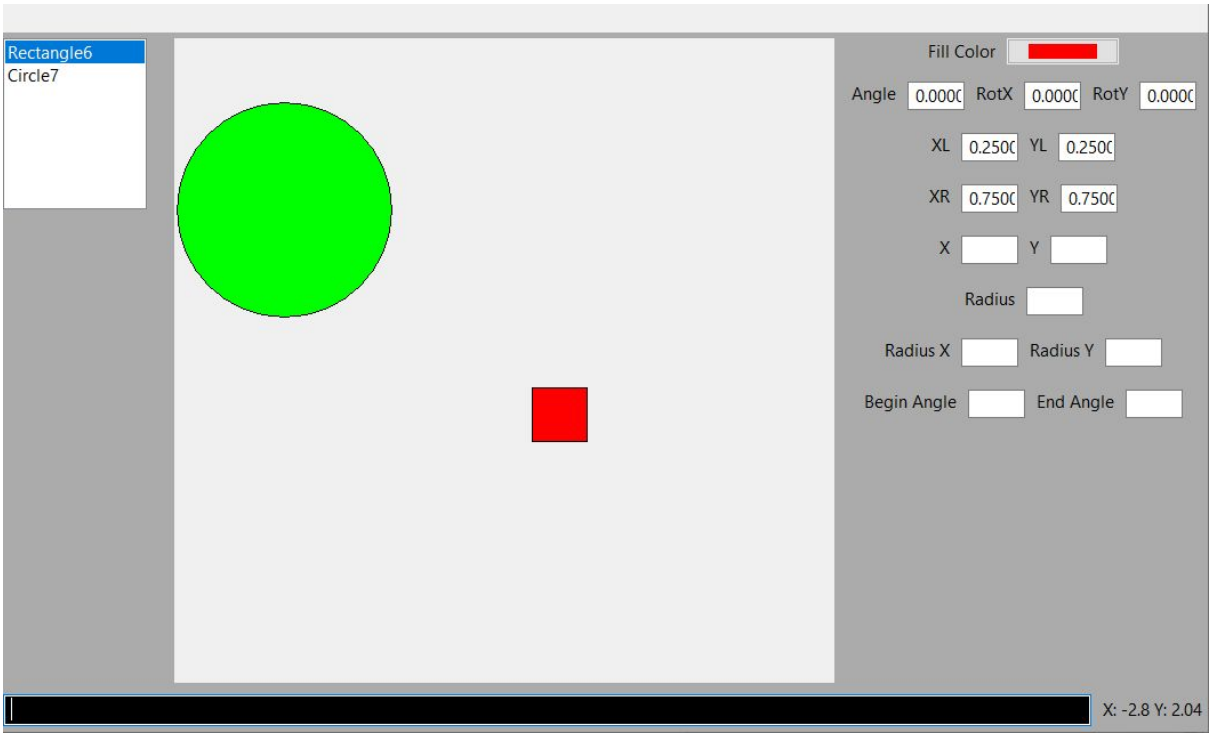
Rotacja dla pojedynczego obiektu
rotate 3 0 0 130



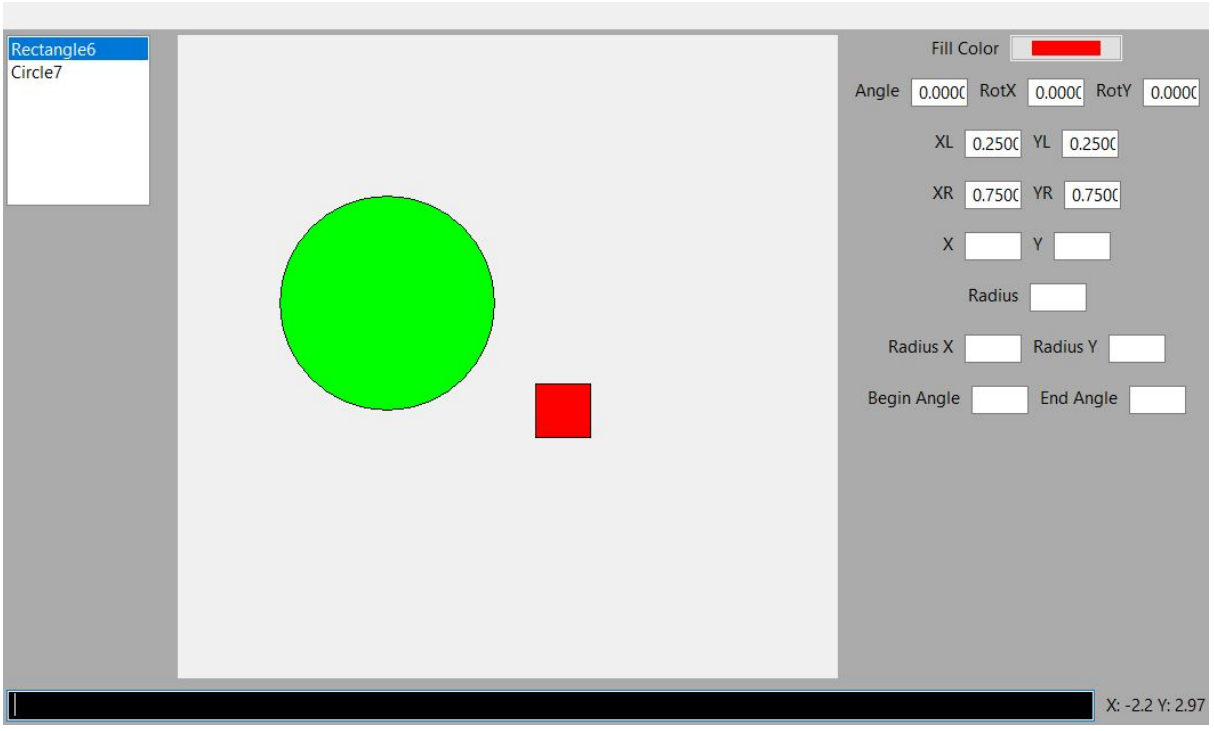
Fill
Wypełnia obiekt z odpowiednim id podanym kolorem.
Przykładowo: *fill 4 #00ff00*



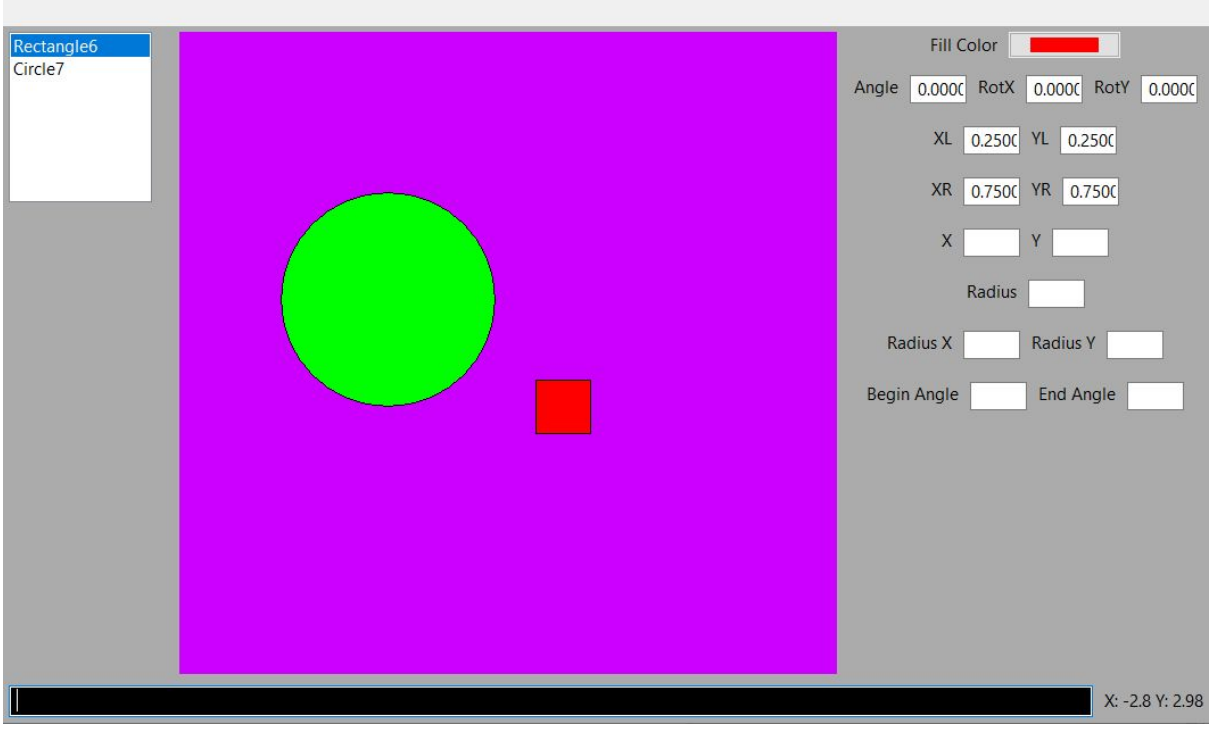
Move:
komenda umożliwia przesuwanie elementów o zadany wektor
move 7 0.9 0.9
przykładowy rysunek przed użyciem komendy:



po użyciu komendy:



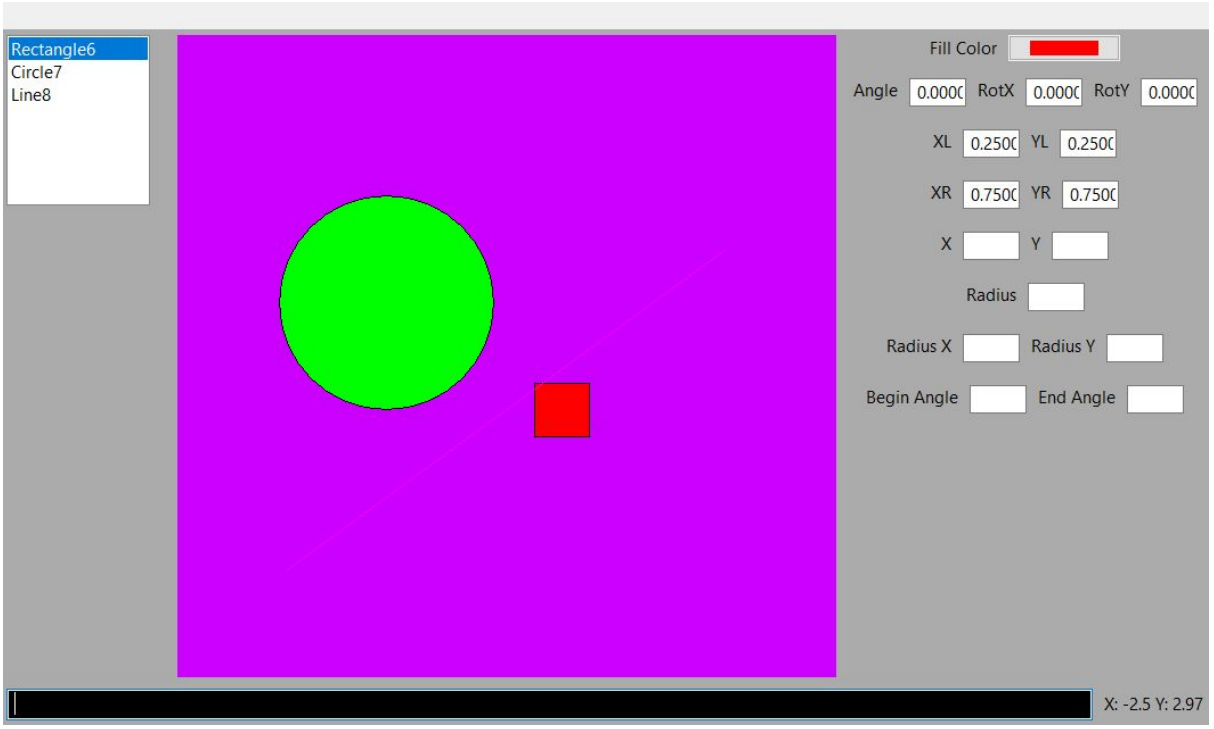
Background:
komenda zmienia tło panelu rysowania
background #cc00ff



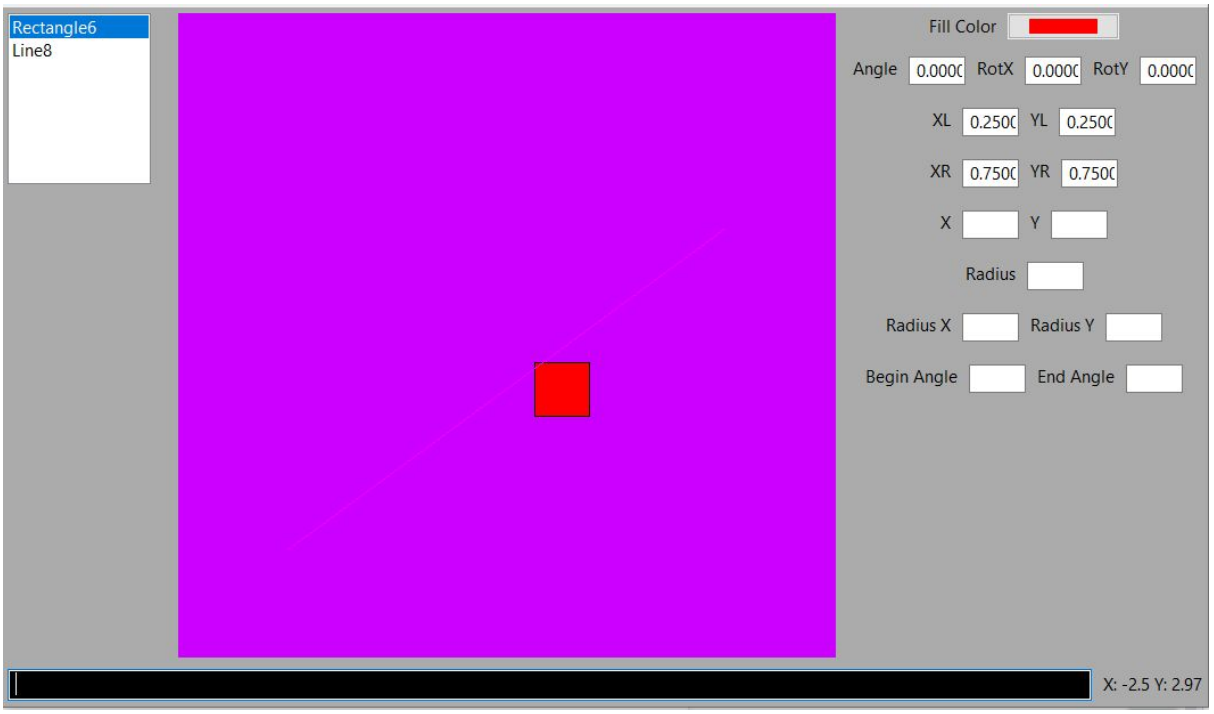
Delete

Po wpisaniu komendy *delete* i odpowiedniego id obiektu dany obiekt znika z ekranu roboczego i listy obiektów.

przykładowy rysunek w celu ukazania przed:



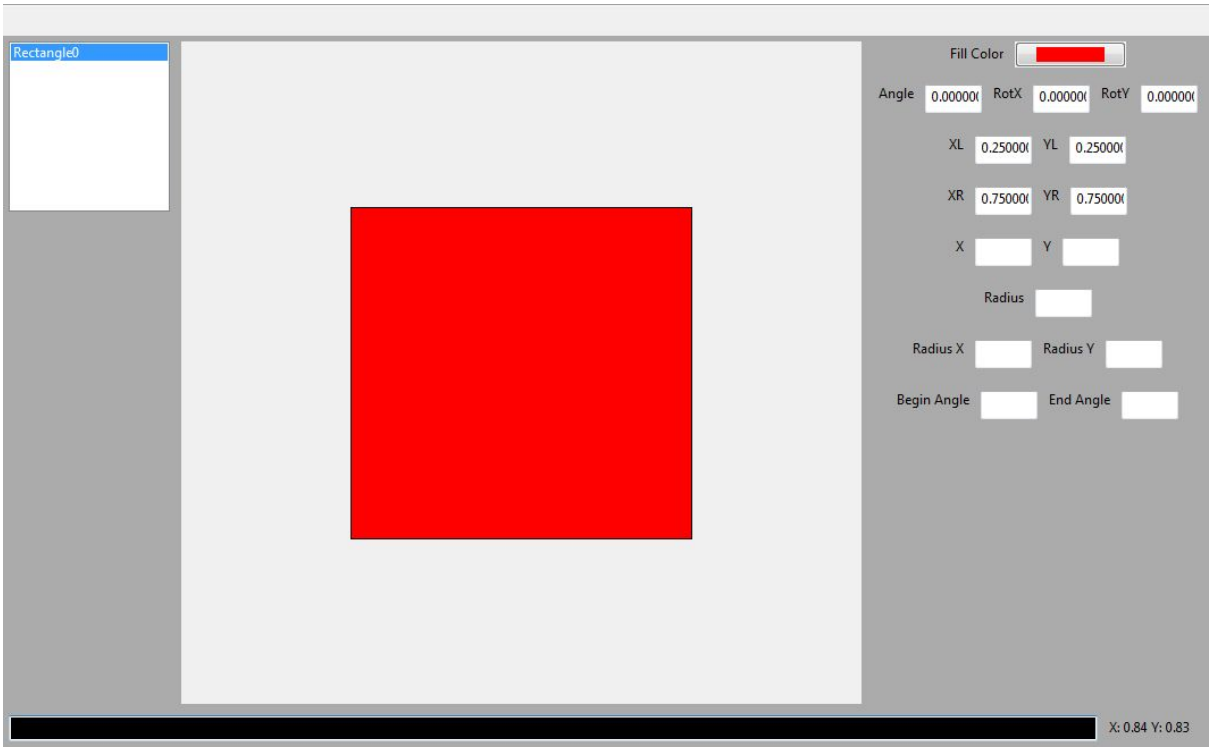
po użyciu komendy *delete 7*



Range

Domyślnie lewy górny narożnik ma współrzędne (0, 0), a prawy dolny narożnik (1, 1), komenda *range* pozwala zmienić te współrzędne, co powoduje odpowiednie przeskalowanie i przesunięcie narysowanych obiektów.

Przed użyciem



Po użyciu komendy *range -1 -1 1 1*

Podsumowanie:

- Niestety nie udało nam się wykonać dodatkowych komend rysujących, które rysowałyby wypełnieniem innym niż kolor dla podanego obiektu.
- Nie udało nam się również zaimplementować obsługi błędów. Podczas złego wpisania komendy, aplikacja nie pokazuje błędu, ponieważ nie sprawdza poprawności wpisywanych komend.
- Z dodatkowych poprawek GUI mogłoby wyglądać lepiej i być bardziej funkcjonalne.