Przykład problemu – kolizja obiektów; Konstruktory i destrukotry, lista inicjalizacyjna

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Wydział Elektroniki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejszy dokument zawiera materiały do wykładu dotyczącego programowania obiektowego. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych prywatnych potrzeb i może on być kopiowany

Niniejsza prezentacja została wykonana przy użyciu systemu składu LaTeX *oraz stylu* `beamer`, *którego autorem jest Till Tantau.*

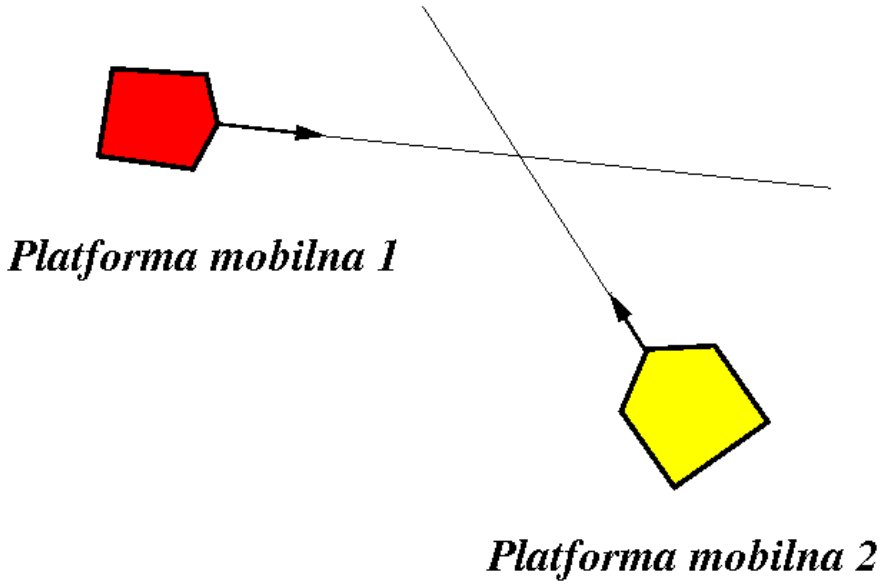
Strona domowa projektu `Beamer`:
`http://latex-beamer.sourceforge.net`

Plan prezentacji

Plan prezentacji

Opis problemu

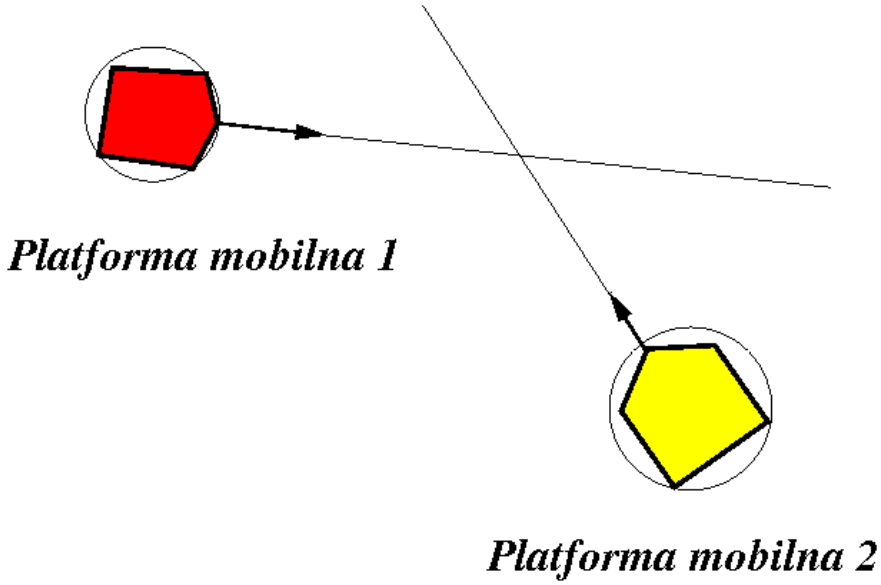
Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.



Opis problemu

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.



Plan prezentacji

Etapy osiągania rozwiązania

- **Analiza**
- **Projektowanie**
- **Konstrukcja**

Etapy osiągania rozwiązania

- **Analiza** – jest odwzorowaniem rzeczywistego świata na jego model koncepcyjny
- **Projektowanie**
- **Konstrukcja**

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

$$d(t) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

$$d(t) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$\begin{cases} x_1(t) = v_{x,1}t + x_{0,1} \\ y_1(t) = v_{y,1}t + y_{0,1} \end{cases} \qquad \begin{cases} x_2(t) = v_{x,2}t + x_{0,2} \\ y_2(t) = v_{y,2}t + y_{0,2} \end{cases}$$

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

$$d(t) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$\begin{cases} x_1(t) = v_{x,1}t + x_{0,1} \\ y_1(t) = v_{y,1}t + y_{0,1} \end{cases} \qquad \begin{cases} x_2(t) = v_{x,2}t + x_{0,2} \\ y_2(t) = v_{y,2}t + y_{0,2} \end{cases}$$

Analityczne rozwiązanie problemu

Rozwiązaniem problemu jest wyznaczenie najmniejszej odległości na jaką zbliżą się do siebie platformy. Kolizja nie nastąpi jeżeli odległość ta będzie większa niż suma promieni okręgów opisanych na korpusach obu platform.

$$d(t) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$\begin{cases} x_1(t) = v_{x,1}t + x_{0,1} \\ y_1(t) = v_{y,1}t + y_{0,1} \end{cases} \qquad \begin{cases} x_2(t) = v_{x,2}t + x_{0,2} \\ y_2(t) = v_{y,2}t + y_{0,2} \end{cases}$$

$$\Downarrow$$

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

Analityczne rozwiązanie problemu

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

$$\Downarrow$$

$$d(t) = \sqrt{(v_{x,12}t + x_{0,12})^2 + (v_{y,12}t + y_{0,12})^2}$$

gdzie $v_{x,12} = v_{x,1} - v_{x,2}$ analogicznie $x_{0,12}, v_{y,12}, y_{0,12}$.

Analityczne rozwiązanie problemu

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

$$\Downarrow$$

$$d(t) = \sqrt{(v_{x,12}t + x_{0,12})^2 + (v_{y,12}t + y_{0,12})^2}$$

gdzie $v_{x,12} = v_{x,1} - v_{x,2}$ analogicznie $x_{0,12}, v_{y,12}, y_{0,12}$.

Szukamy wartość t, dla której funkcja $d(.)$ osiąga minimum.

Analityczne rozwiązanie problemu

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

$$\Downarrow$$

$$d(t) = \sqrt{(v_{x,12}t + x_{0,12})^2 + (v_{y,12}t + y_{0,12})^2}$$

gdzie $v_{x,12} = v_{x,1} - v_{x,2}$ analogicznie $x_{0,12}, v_{y,12}, y_{0,12}$.

Szukamy wartość t, dla której funkcja $d(.)$ osiąga minimum.

$$\frac{\mathrm{d}d(t)}{\mathrm{d}t} = 0 \quad \longrightarrow \quad v_{x,12}(v_{x,12}t + x_{0,12}) + v_{y,12}(v_{y,12}t + y_{0,12}) = 0$$

Analityczne rozwiązanie problemu

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

$$\Downarrow$$

$$d(t) = \sqrt{(v_{x,12}t + x_{0,12})^2 + (v_{y,12}t + y_{0,12})^2}$$

gdzie $v_{x,12} = v_{x,1} - v_{x,2}$ analogicznie $x_{0,12}, v_{y,12}, y_{0,12}$.

Szukamy wartość t, dla której funkcja $d(.)$ osiąga minimum.

$$\frac{\mathrm{d}d(t)}{\mathrm{d}t} = 0 \quad \longrightarrow \quad v_{x,12}(v_{x,12}t + x_{0,12}) + v_{y,12}(v_{y,12}t + y_{0,12}) = 0$$

Analityczne rozwiązanie problemu

$$d(t) = \sqrt{((v_{x,1} - v_{x,2})t + (x_{0,1} - x_{0,2}))^2 + ((v_{y,1} - v_{y,2})t + (y_{0,1} - y_{0,2}))^2}$$

$$\Downarrow$$

$$d(t) = \sqrt{(v_{x,12}t + x_{0,12})^2 + (v_{y,12}t + y_{0,12})^2}$$

gdzie $v_{x,12} = v_{x,1} - v_{x,2}$ analogicznie $x_{0,12}, v_{y,12}, y_{0,12}$.

Szukamy wartość t, dla której funkcja $d(.)$ osiąga minimum.

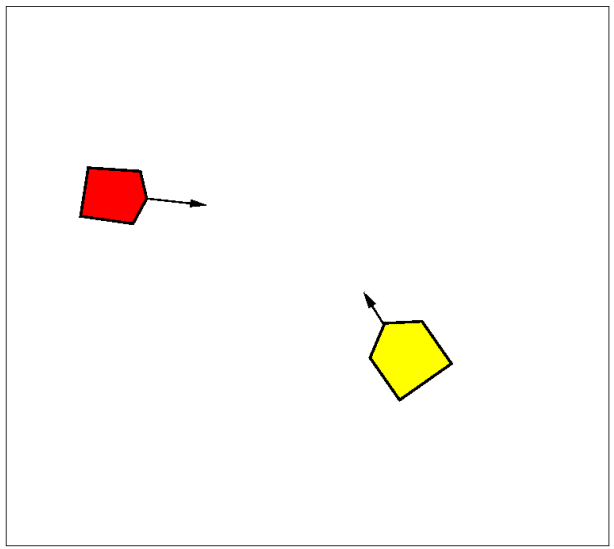
$$\frac{\mathrm{d}d(t)}{\mathrm{d}t} = 0 \quad \longrightarrow \quad v_{x,12}(v_{x,12}t + x_{0,12}) + v_{y,12}(v_{y,12}t + y_{0,12}) = 0$$

$$t = \frac{v_{x,12}x_{0,12} + v_{y,12}y_{0,12}}{v_{x,12}^2 + v_{y,12}^2}$$

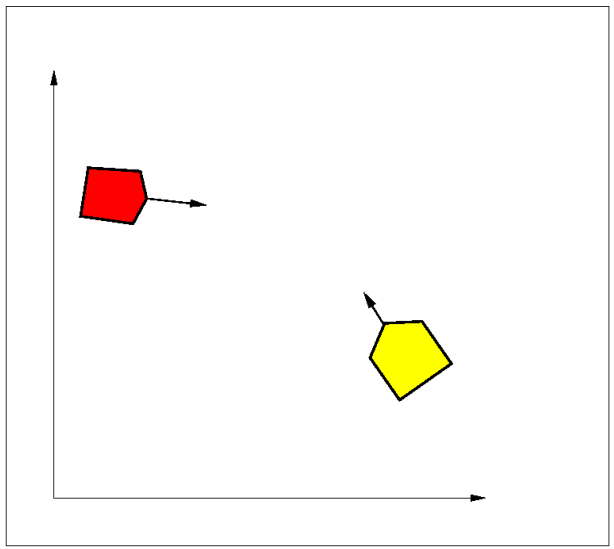
Finalne rozwiązanie problemu

$$d = \left(\left((v_{x,1} - v_{x,2}) \frac{v_{x,12}x_{0,12} + v_{y,12}y_{0,12}}{v_{x,12}^2 + v_{y,12}^2} + (x_{0,1} - x_{0,2}) \right)^2 + \left((v_{y,1} - v_{y,2}) \frac{v_{x,12}x_{0,12} + v_{y,12}y_{0,12}}{v_{x,12}^2 + v_{y,12}^2} + (y_{0,1} - y_{0,2}) \right)^2 \right)^{\frac{1}{2}}$$

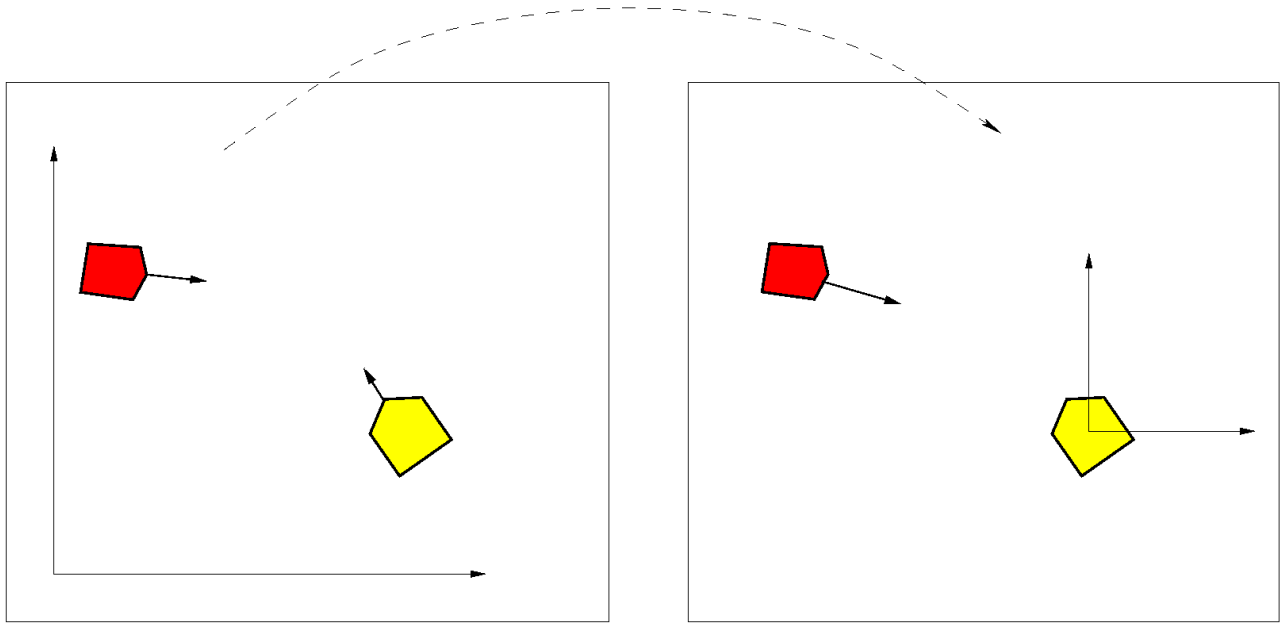
Zmiana układu współrzędnych



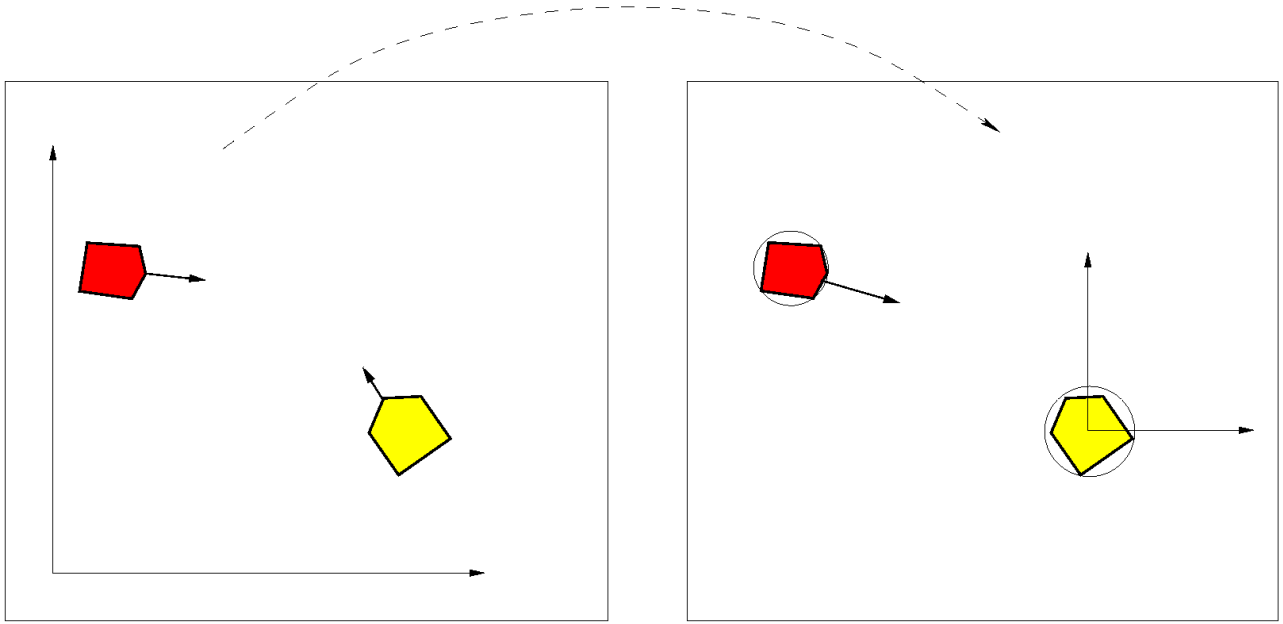
Zmiana układu współrzędnych



Zmiana układu współrzędnych



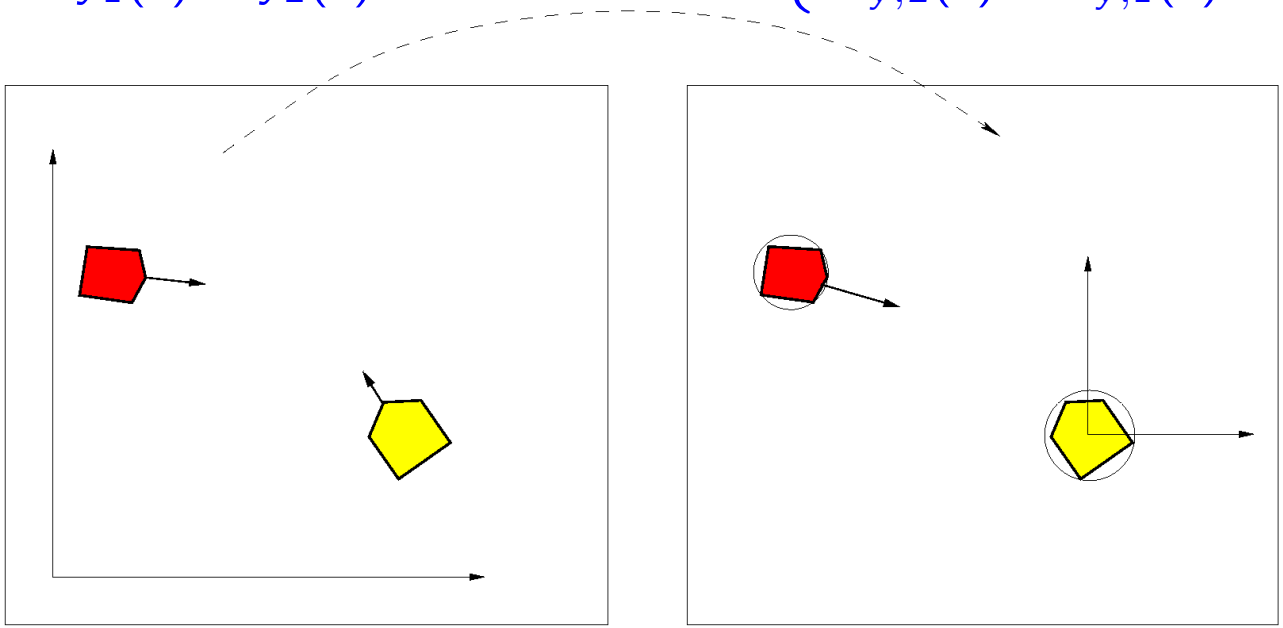
Zmiana układu współrzędnych



Zmiana układu współrzędnych

Transformacja do lokalnego układu współrzędnych platformy nr 2 związanego ze środkiem okręgu opisanego na obrysie jej korpusu.

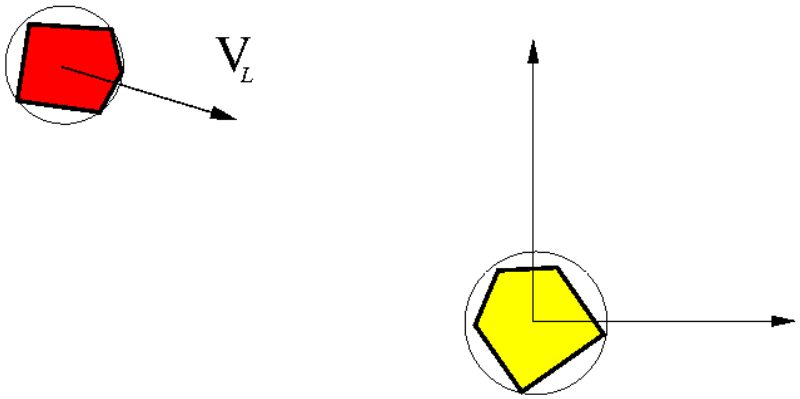
$$\begin{cases} x_L(t) = x_1(t) - x_2(t) \\ y_L(t) = y_1(t) - y_2(t) \end{cases} \qquad \begin{cases} v_{x,L}(t) = v_{x,1}(t) - v_{x,2}(t) \\ v_{y,L}(t) = v_{y,1}(t) - v_{y,2}(t) \end{cases}$$



Dzięki zastosowanej transformacji rozwiązanie problemu znacznie się upraszcza. Nie trzeba liczyć pochodnej, gdyż w tym przypadku problem jest natury geometrycznej.

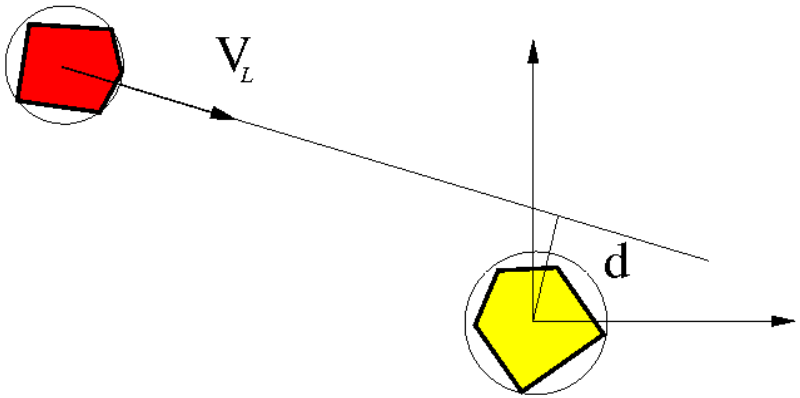
Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



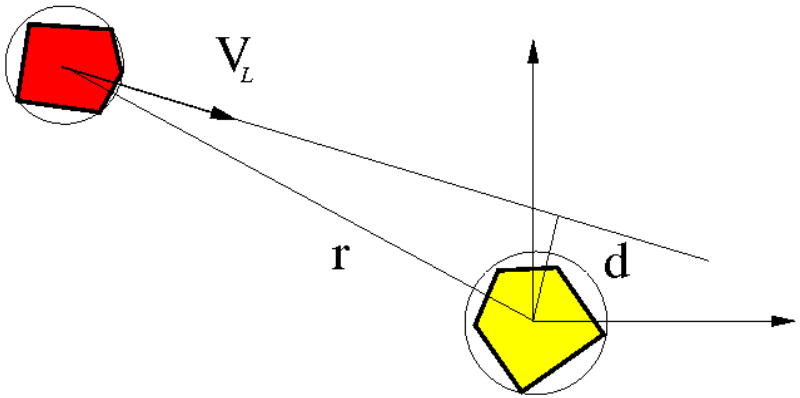
Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



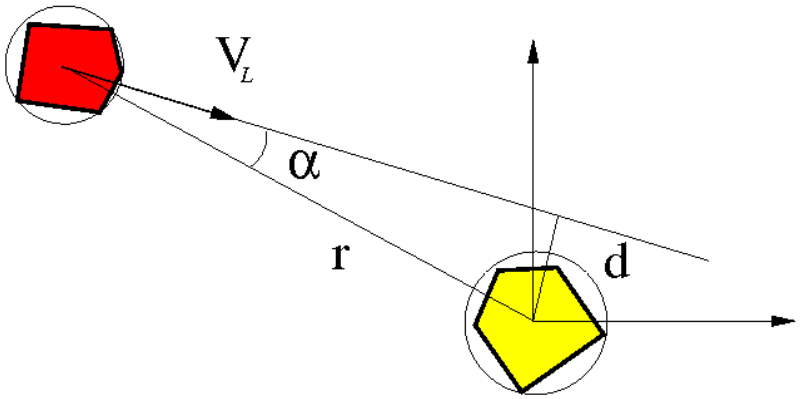
Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



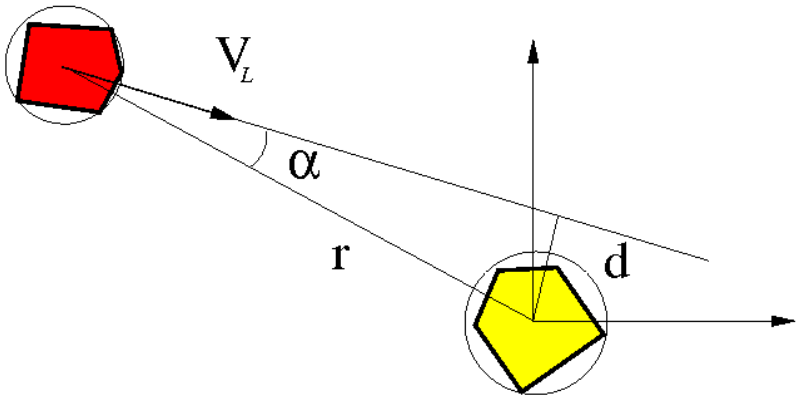
Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



Rozwiązanie w układzie lokalnym

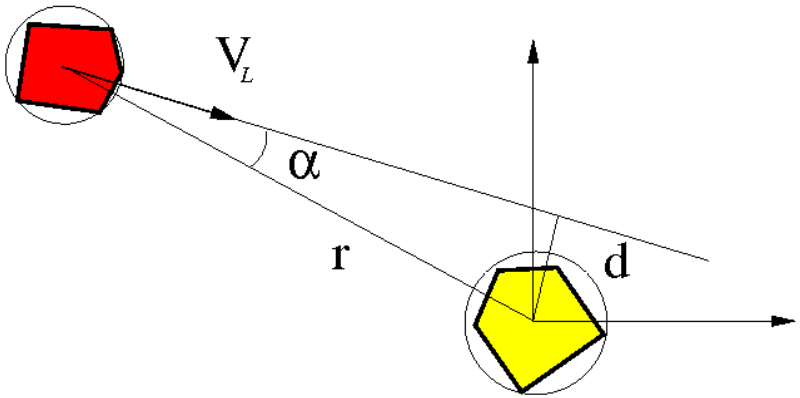
W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



$$d = | \, r \sin \alpha \, |$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.

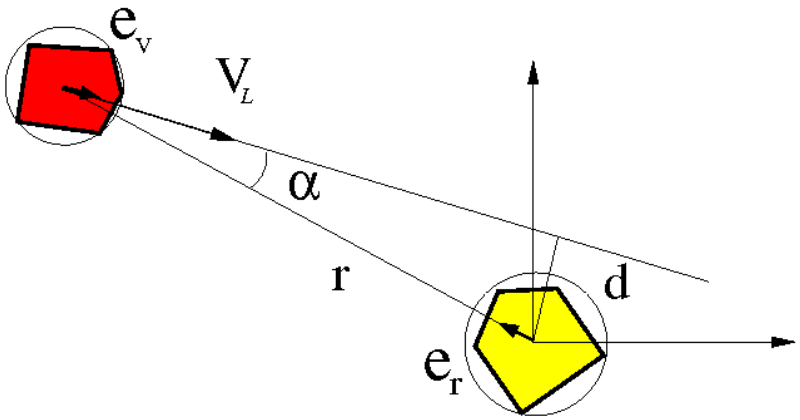


$$d = | \, r \sin \alpha \, |$$

$$d = | \, r \sin \alpha \, | = | \, r(\mathbf{e}_r \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \frac{\mathbf{V}_L}{|| \, \mathbf{V}_L \, ||})_z \, |$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.

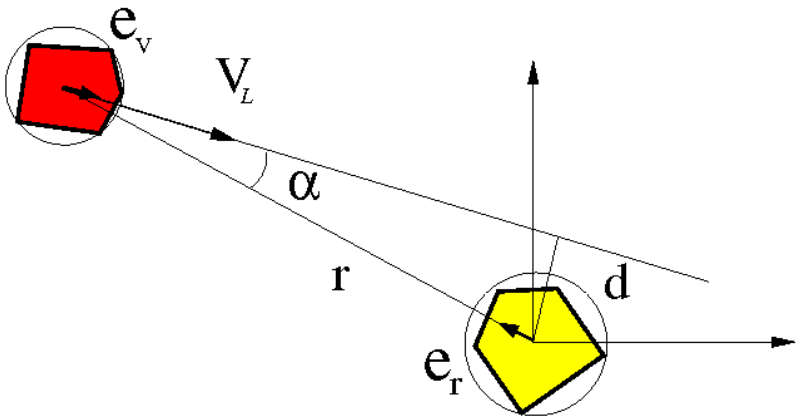


$$d = | \, r \sin \alpha \, |$$

$$d = | \, r \sin \alpha \, | = | \, r(\mathbf{e}_r \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \frac{\mathbf{V}_L}{|| \, \mathbf{V}_L \, ||})_z \, |$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.

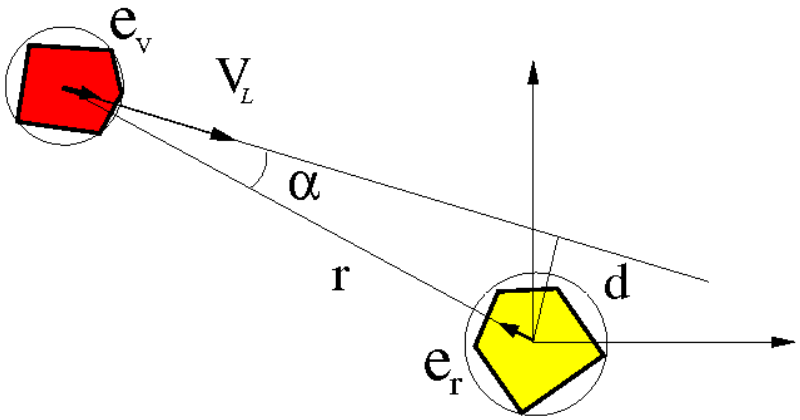


$$d = | \, r \sin \alpha \, |$$

$$d = | \, r \sin \alpha \, | = | \, r(\mathbf{e}_r \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \mathbf{e}_V)_z \, | = | \, (\mathbf{r} \times \frac{\mathbf{V}_L}{|| \, \mathbf{V}_L \, ||})_z \, |$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.

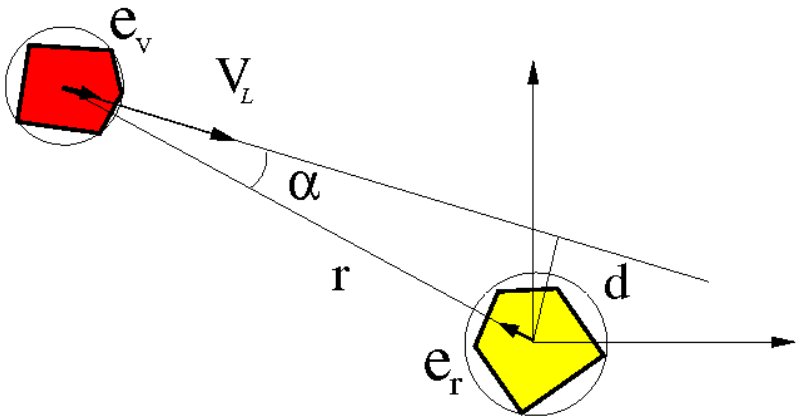


$$d = | r \sin \alpha |$$

$$d = | r \sin \alpha | = | r(\mathbf{e}_r \times \mathbf{e}_V)_z | = | (\mathbf{r} \times \mathbf{e}_V)_z | = | (\mathbf{r} \times \frac{\mathbf{V}_L}{|| \mathbf{V}_L ||})_z |$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.

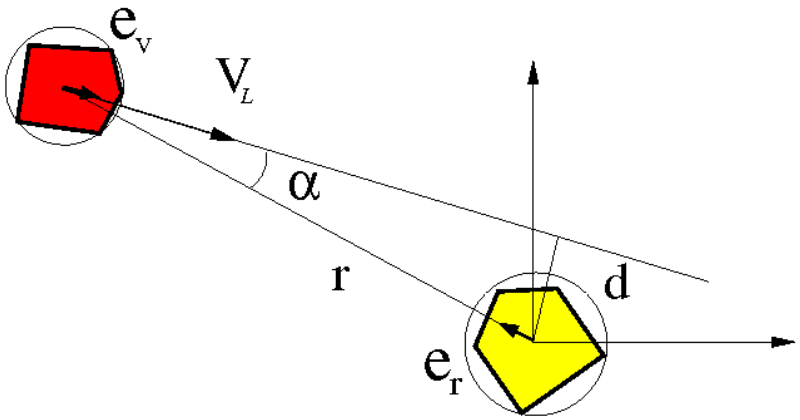


$$d = \mid r \sin \alpha \mid$$

$$d = \mid r \sin \alpha \mid = \mid r(\mathbf{e}_r \times \mathbf{e}_V)_z \mid = \mid (\mathbf{r} \times \mathbf{e}_V)_z \mid = \mid (\mathbf{r} \times \frac{\mathbf{V}_L}{\| \mathbf{V}_L \|})_z \mid$$

Rozwiązanie w układzie lokalnym

W lokalnym układzie współrzędnych jednej z platform problem sprowadza się do wyznaczenia odległości prostej od początku układu współrzędnych.



$$d = \mid r \sin \alpha \mid$$

$$d = \mid r \sin \alpha \mid = \mid r(\mathbf{e}_r \times \mathbf{e}_V)_z \mid = \mid (\mathbf{r} \times \mathbf{e}_V)_z \mid = \mid (\mathbf{r} \times \frac{\mathbf{V}_L}{\mid\mid \mathbf{V}_L \mid\mid})_z \mid$$

$$d = \frac{\mid (\mathbf{r} \times \mathbf{V}_L)_z \mid}{\mid\mid \mathbf{V}_L \mid\mid}$$

Porównanie rozwiązań

$$d = \left(\left((v_{x,1} - v_{x,2}) \frac{v_{x,12}x_{0,12} + v_{y,12}y_{0,12}}{v_{x,12}^2 + v_{y,12}^2} + (x_{0,1} - x_{0,2}) \right)^2 + \left((v_{y,1} - v_{y,2}) \frac{v_{x,12}x_{0,12} + v_{y,12}y_{0,12}}{v_{x,12}^2 + v_{y,12}^2} + (y_{0,1} - y_{0,2}) \right)^2 \right)^{\frac{1}{2}}$$

$$d = \frac{\mid (\mathbf{r} \times \mathbf{V}_L)_z \mid}{\mid\mid \mathbf{V}_L \mid\mid}$$

Plan prezentacji

Etapy osiągania rozwiązania

- **Analiza**
- **Projektowanie** – jest odwzorowaniem modelu koncepcyjnego na model implementacji.
- **Konstrukcja**

Określenie przypadków użycia

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Określenie przypadków użycia

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Stworzona procedura będzie wykorzystywana w *module detekcji kolizji* oraz *module planowania trajektrorii*.

Diagram przypadków użycia

Diagram przypadków użycia służy do obrazowania zachowania systemu, podsystemu lub klasy w taki sposób, żeby użytkownicy mogli zrozumieć, jak z tego bytu korzystać, a programiści mogli go zaimplementować.

Diagram przypadków użycia

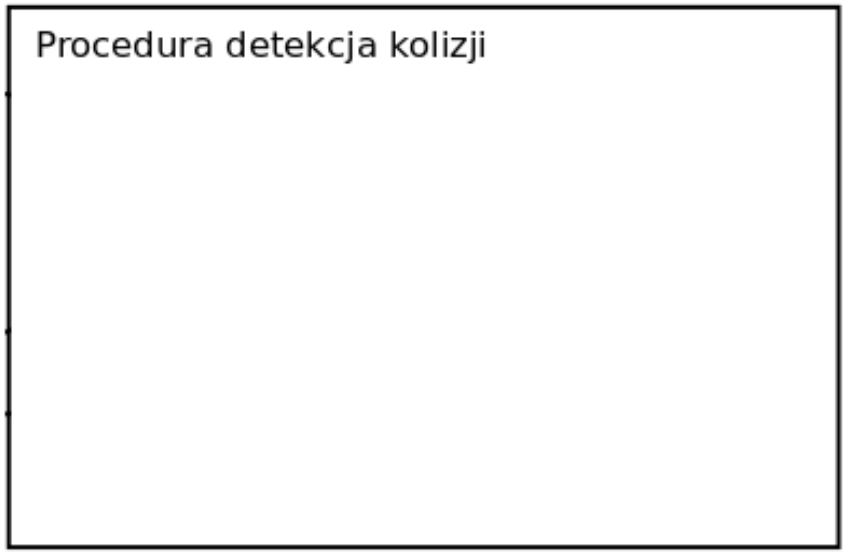


Diagram przypadków użycia

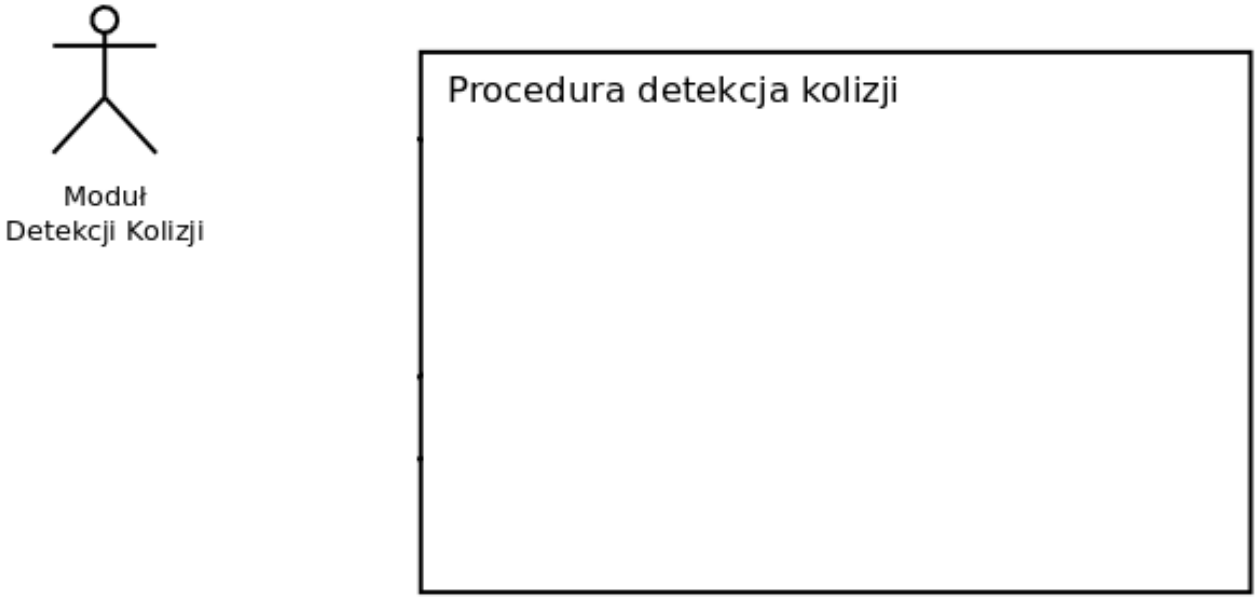


Diagram przypadków użycia

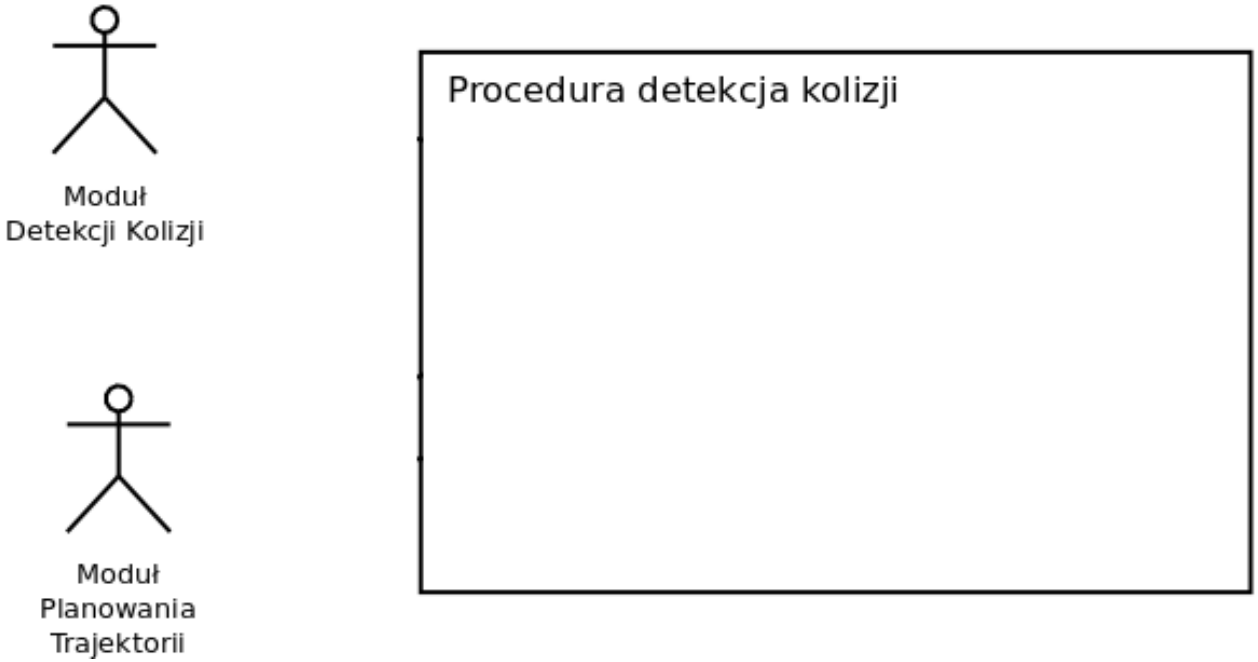


Diagram przypadków użycia

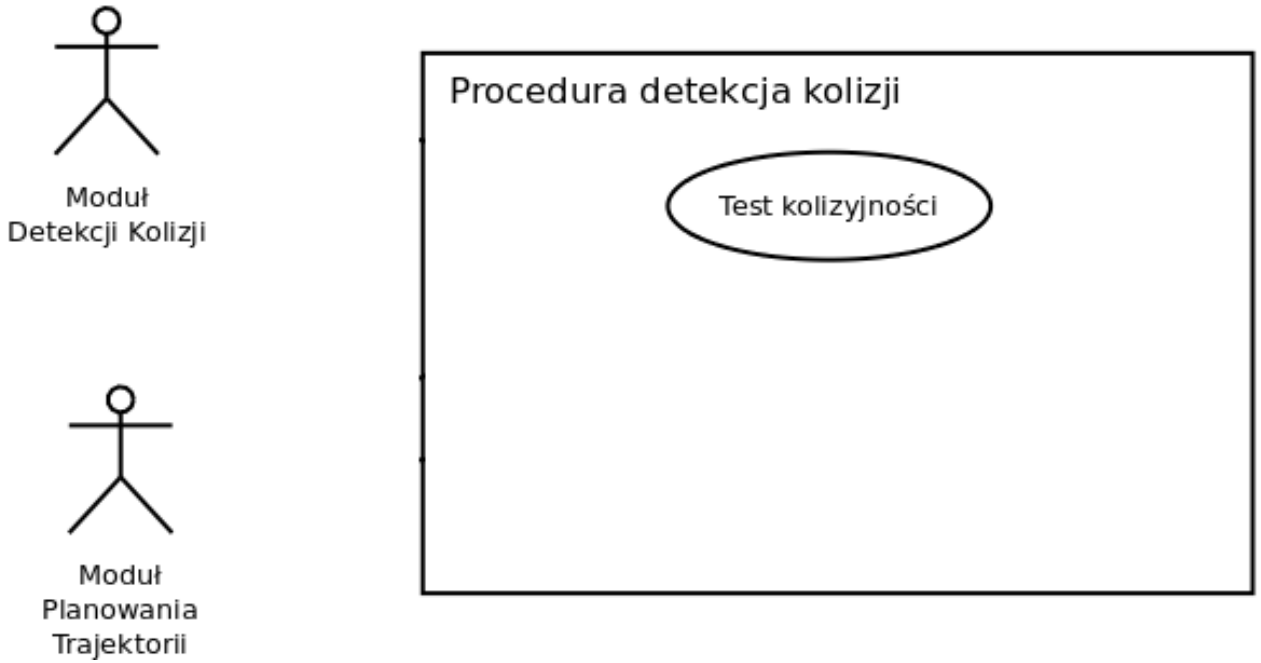


Diagram przypadków użycia

Diagram przypadków użycia

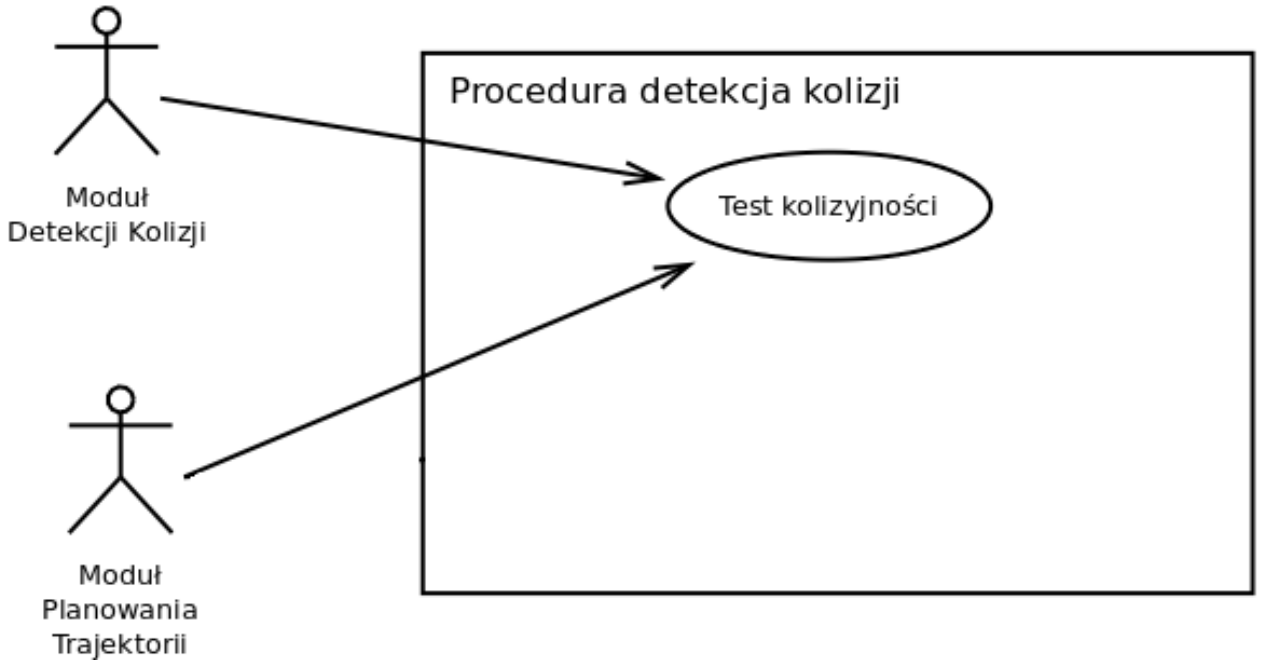


Diagram przypadków użycia

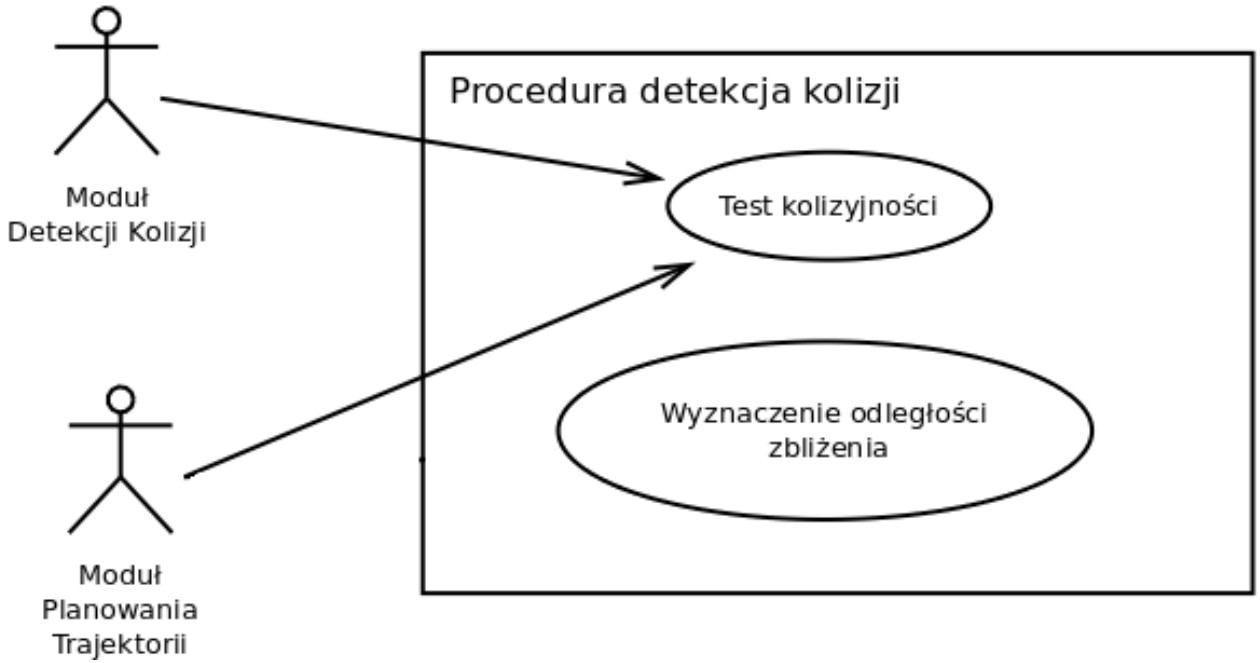


Diagram przypadków użycia

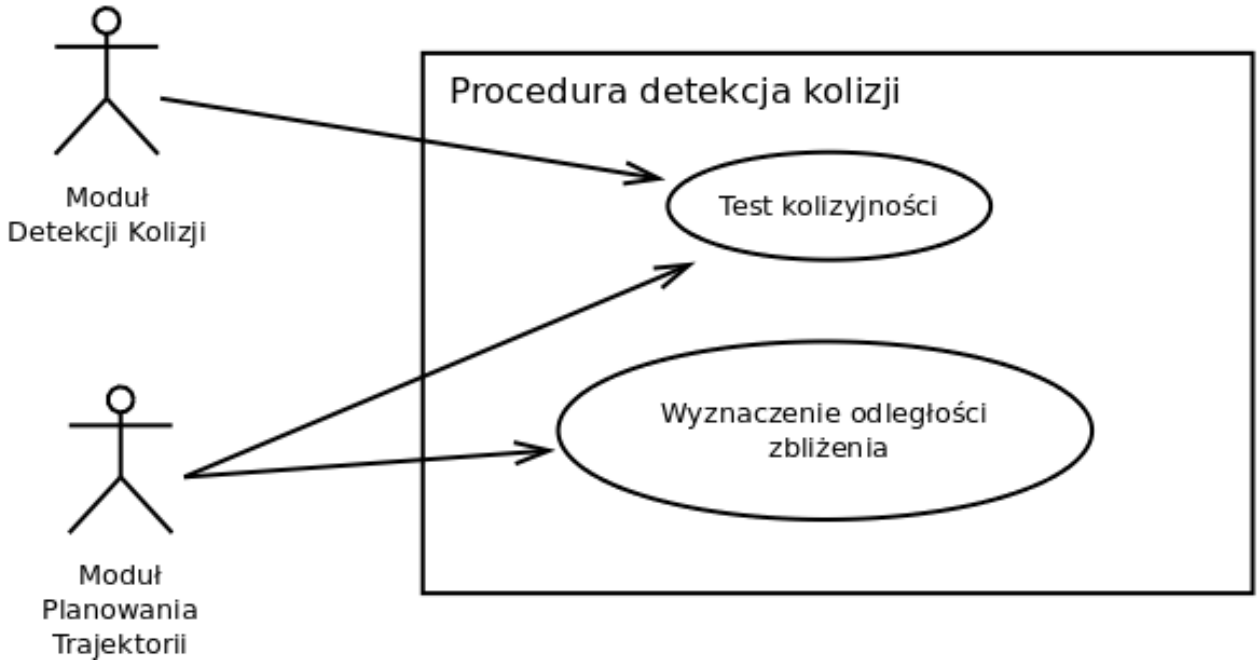


Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.

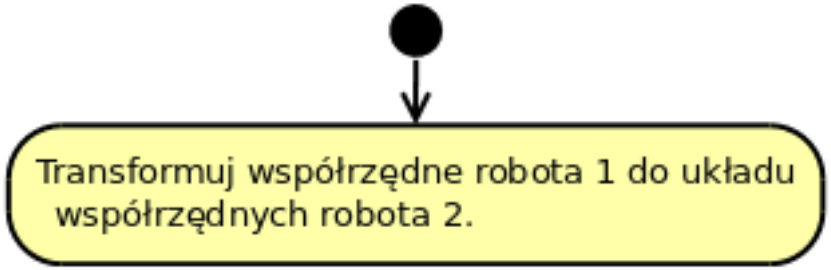
Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.

Start opisu czynności

Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.



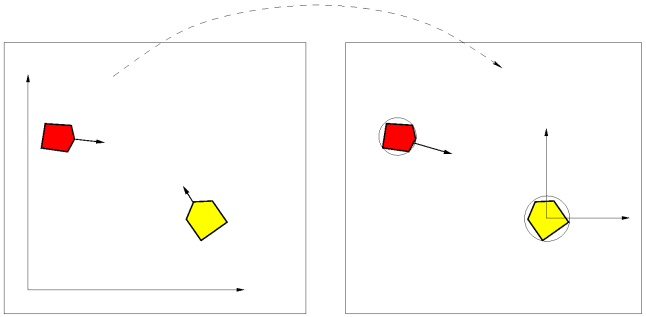
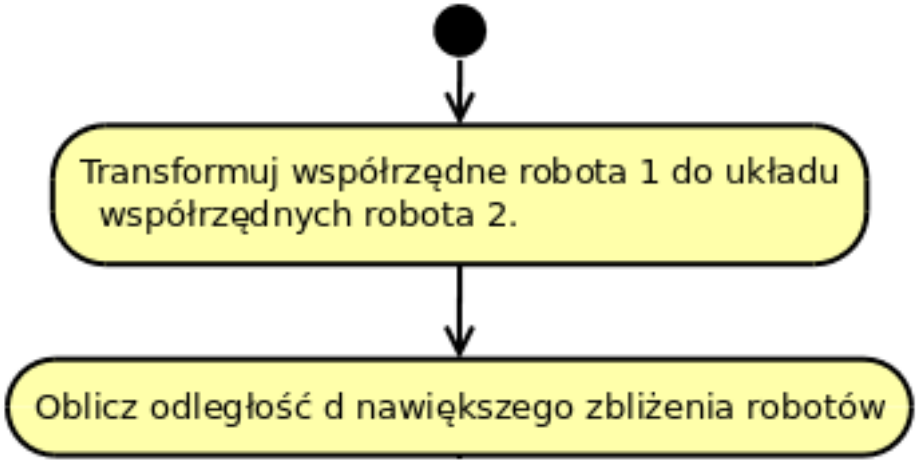


Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.



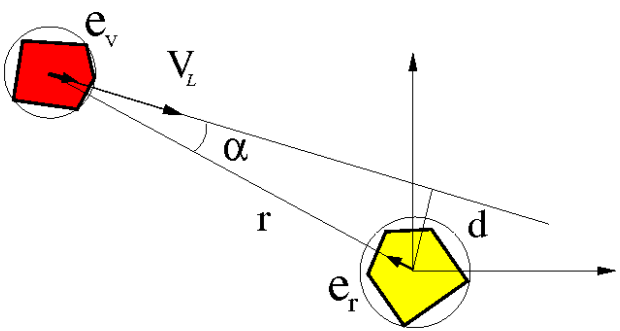
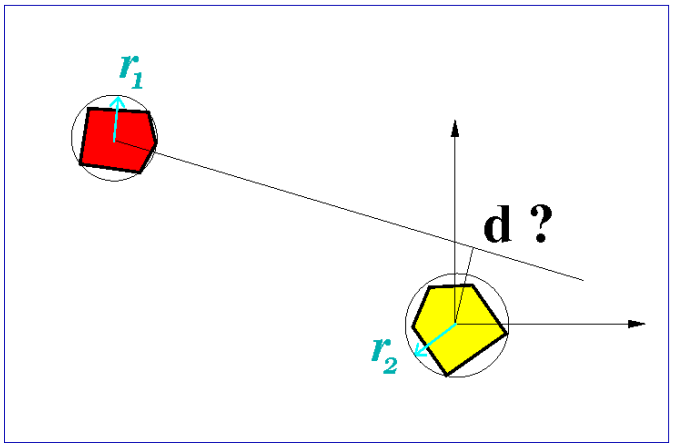


Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.



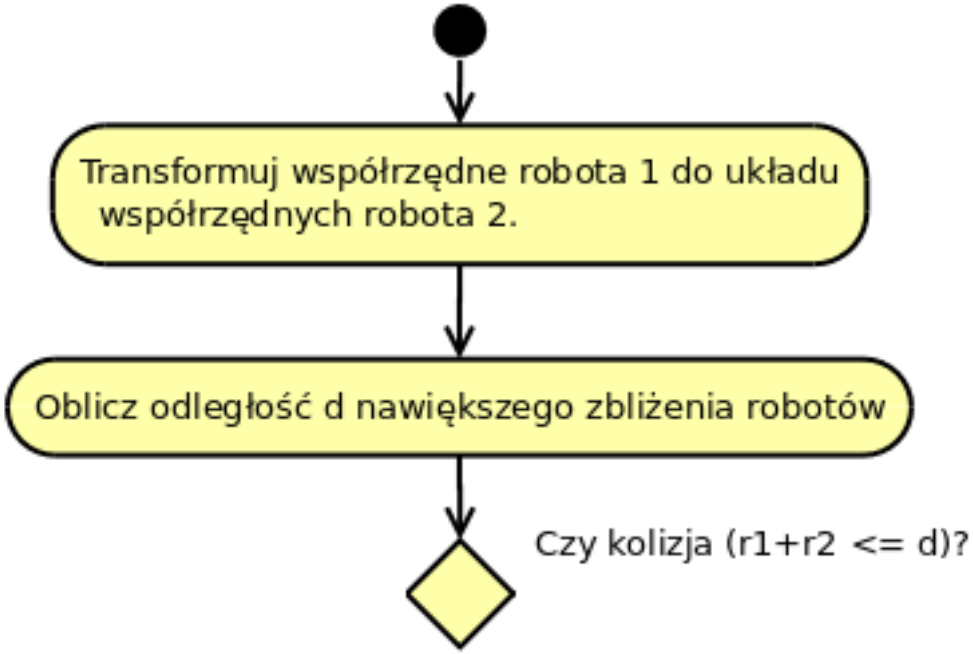


Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.

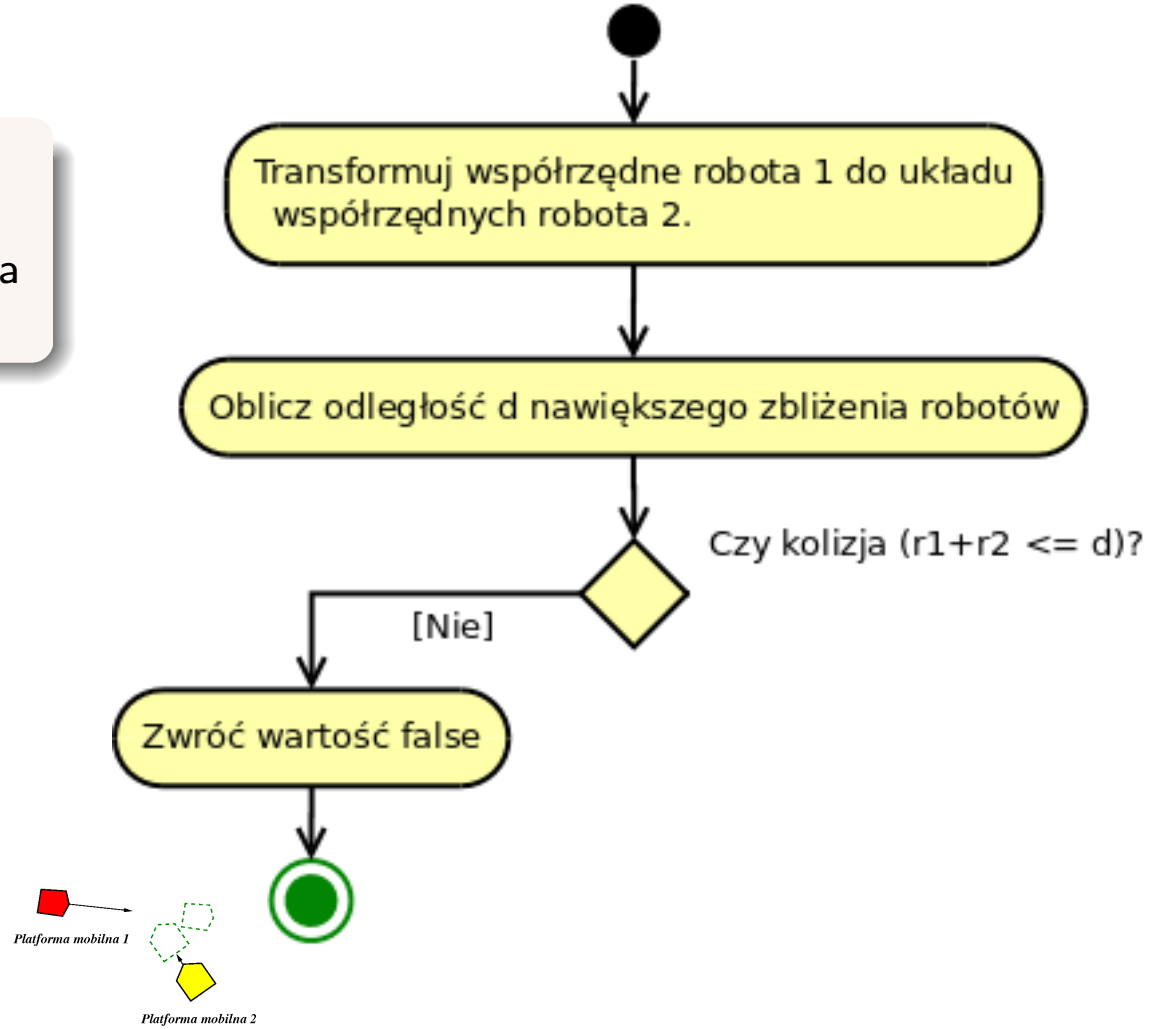


Diagram czynności

Diagram czynności modeluje dynamiczne aspekty systemu. Demonstruje przepływ sterowania od operacji do operacji.

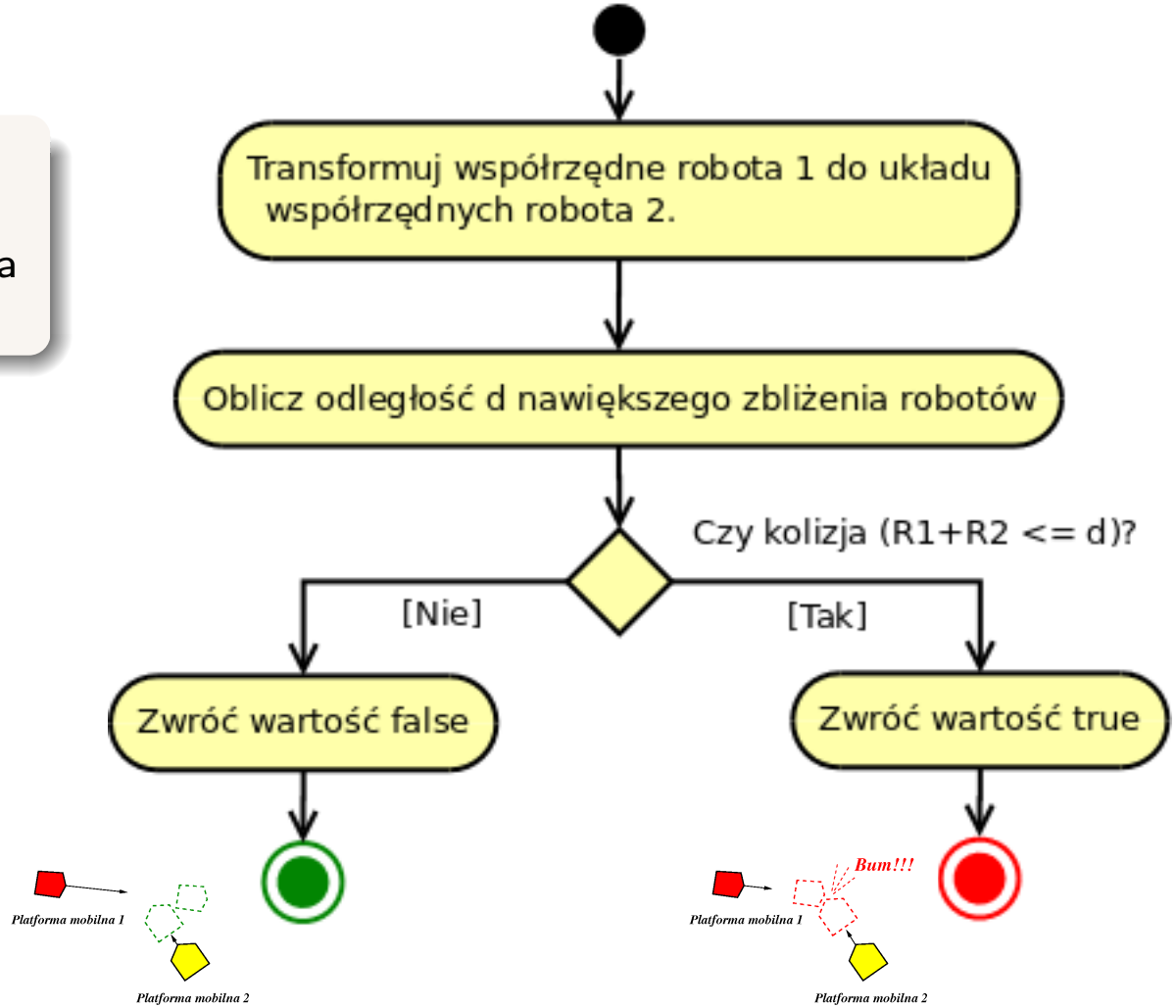


Diagram klas

Diagram klas służy do obrazowania statycznych aspektów systemu. Bierze się w nim pod uwagę wymagania funkcjonalne (usługi), jakie system powinien udostępniać swoim użytkownikom.

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych* poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**
• platforma (mobilna)		

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych* poruszających się wzdłuż prostej ruchem jednostajnym zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**
• platforma (mobilna)	• ruch • położenie • rozmiar	

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych* *poruszających się* wzdłuż prostej *ruchem jednostajnym* zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**
• platforma (mobilna)	• ruch • położenie • rozmiar	

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie okręgu opisanego na rzucie pionowym korpusu platformy.

Kluczowe rzeczowniki:

- platforma (mobilna)

Własności:

- ruch
- położenie
- rozmiar

Operacje:

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych* *poruszających się* *wzdłuż prostej* *ruchem jednostajnym* zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**
• platforma (mobilna)	• ruch • położenie • rozmiar	

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi kolizja lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:	**Własności:**	**Operacje:**
• platforma (mobilna)	• ruch • położenie • rozmiar	

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się odległości.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

(Po rozwiązaniu analitycznym)

Dodatkowe operacje:

- odejmowanie wektorów
- iloczyn skalarny
- iloczyn wektorowy

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

(Po rozwiązaniu analitycznym)

Dodatkowe operacje:

- odejmowanie wektorów
- iloczyn skalarny
- iloczyn wektorowy

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

(Po rozwiązaniu analitycznym)

Dodatkowe operacje:

- odejmowanie wektorów
- iloczyn skalarny
- iloczyn wektorowy

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

(Po rozwiązaniu analitycznym)

Dodatkowe operacje:

- odejmowanie wektorów
- iloczyn skalarny
- iloczyn wektorowy

Diagram klas

Należy zaimplementować procedurę obliczeniową umożliwiającą stwierdzenie, czy dla dwóch *platform mobilnych poruszających się wzdłuż prostej ruchem jednostajnym* zachodzi *kolizja* lub w jakiej miną się *odległości*.

Zakładamy, że kolizja zachodzi jeśli jakaś przeszkoda znajdzie się w obrębie *okręgu opisanego na rzucie pionowym korpusu platformy*.

Kluczowe rzeczowniki:

- platforma (mobilna)

(Po rozwiązaniu analitcznym)

Dodatkowe rzeczowniki:

- wektor

Własności:

- ruch
- położenie
- rozmiar

Operacje:

- sprawdzenie kolizyjności
- wyznaczanie odległości

(Po rozwiązaniu analitycznym)

Dodatkowe operacje:

- odejmowanie wektorów
- iloczyn skalarny
- iloczyn wektorowy

Diagram klas

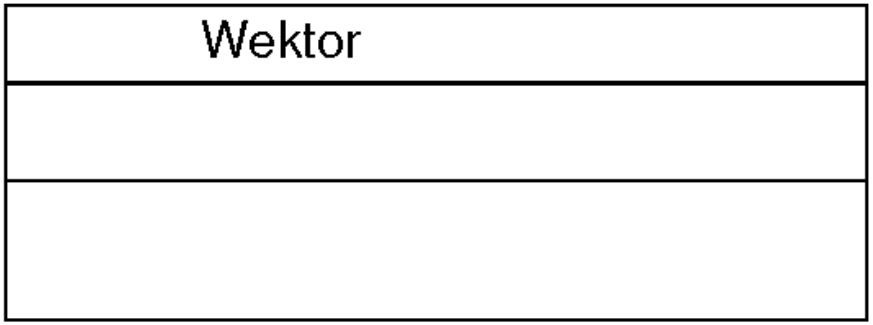


Diagram klas

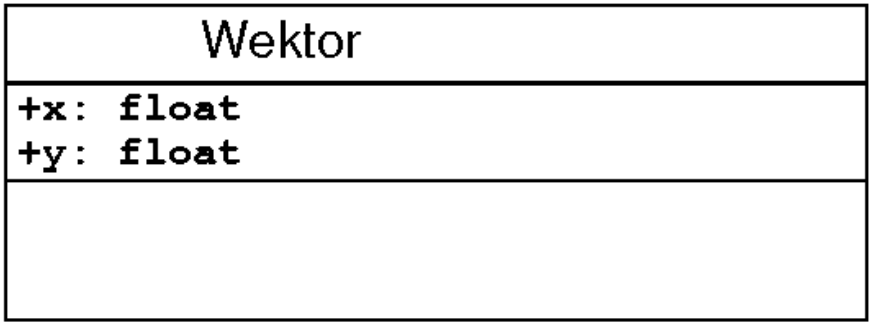
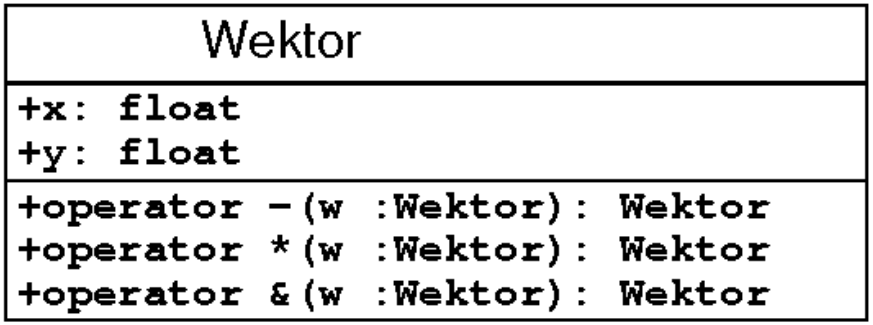


Diagram klas

Wektor
+x: float +y: float
+operator -(w :Wektor): Wektor +operator *(w :Wektor): Wektor +operator &(w :Wektor): Wektor

Diagram klas



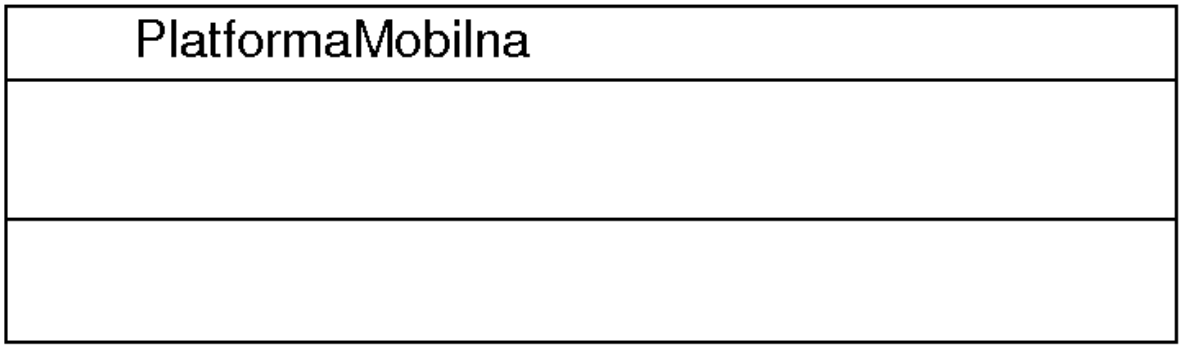
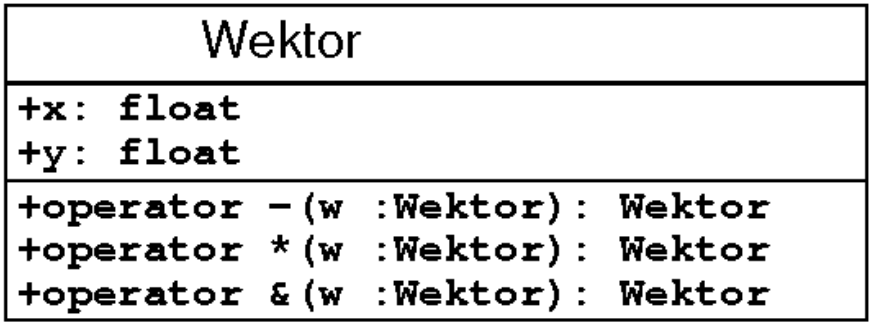


Diagram klas



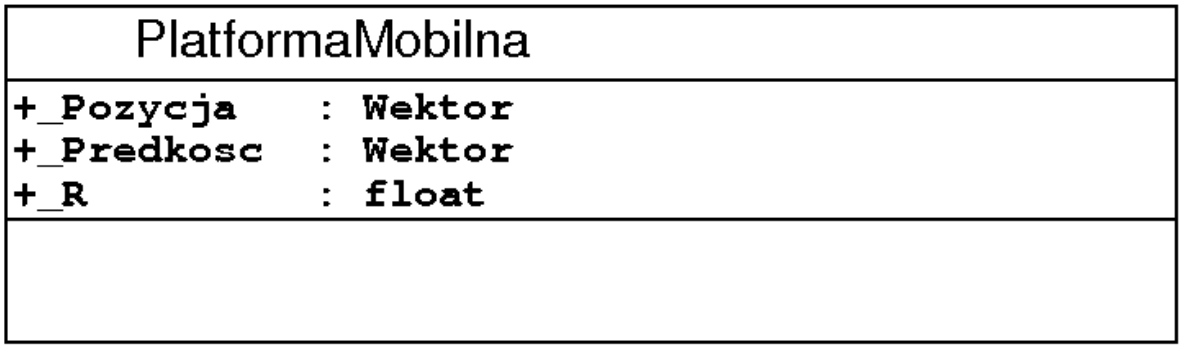


Diagram klas

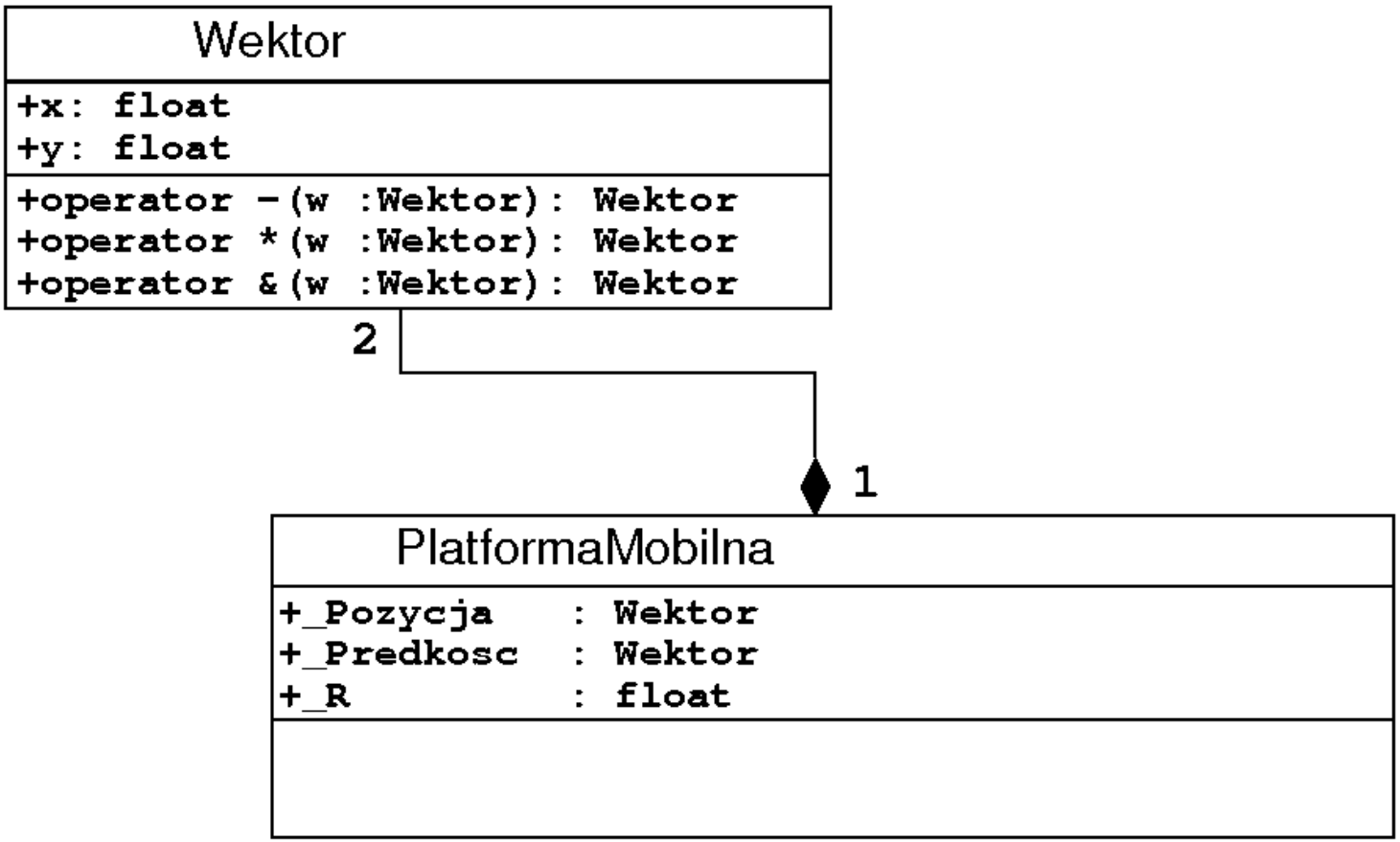


Diagram klas

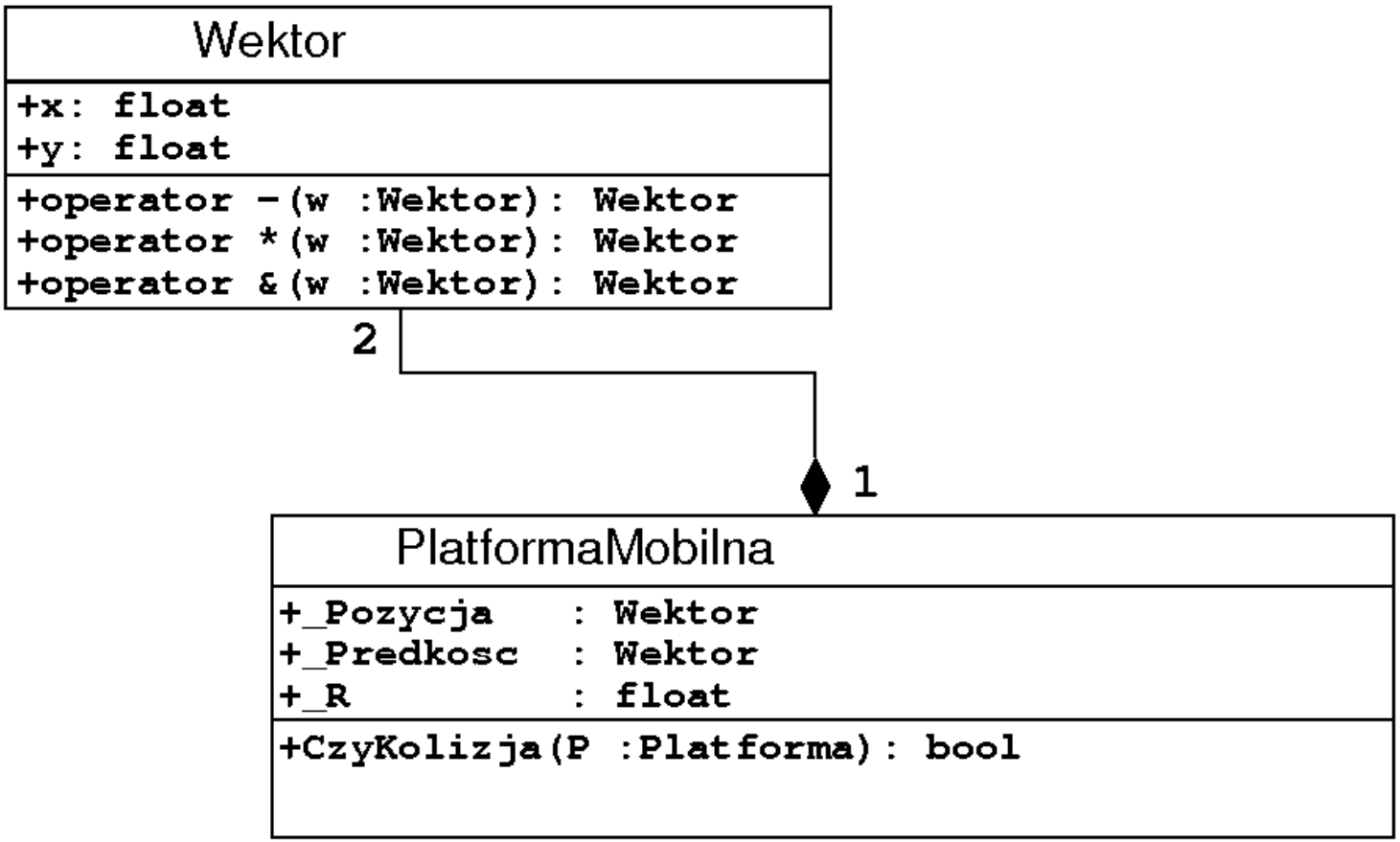
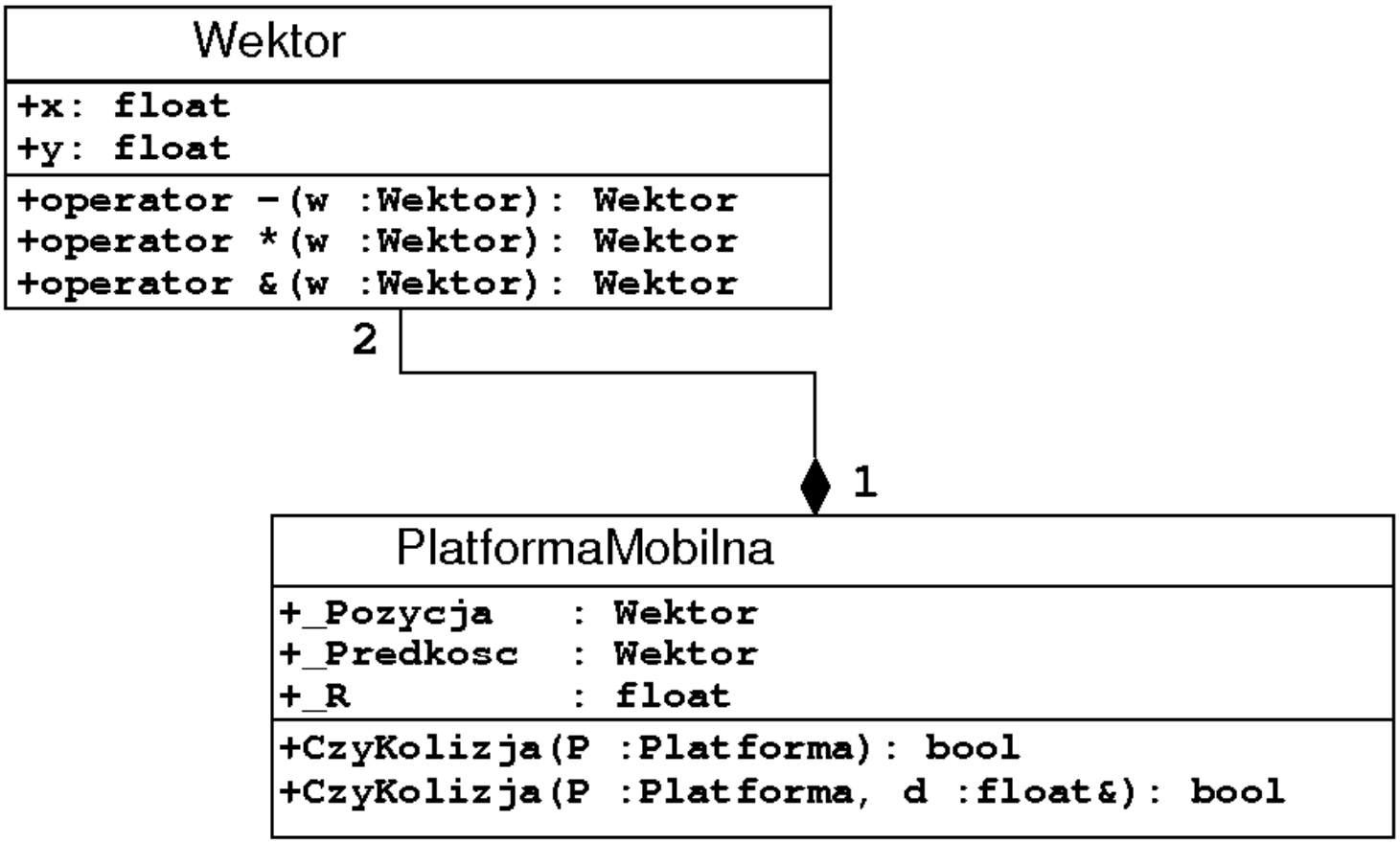


Diagram klas



Etapy osiągania rozwiązania

- **Analiza**
- **Projektowanie**
- **Konstrukcja** – jest odwzorowaniem modelu implementacji na działający system.

Plan prezentacji

Definicje klasy *Wektor*

```
class Wektor { // ..........................................................
 public:
  float x, y;

  Wektor operator – ( Wektor ) const ;
  float operator ∗ ( const Wektor &V ) const    { return x∗V.y − y∗V.x; }
  float operator & ( const Wektor &V ) const    { return x∗V.x + y∗V.y; }
}; // ..........................................................

Wektor Wektor::operator – ( Wektor W ) const
{
  W.x = x – W.x;     W.y = y – W.y;
  return W;
}
```

Definicje klasy *Wektor*

```
class Wektor { // ..........................................................
 public:
  float x, y;

  Wektor operator – ( Wektor ) const ;
  float operator * ( const Wektor &V ) const    { return x*V.y – y*V.x; }
  float operator & ( const Wektor &V ) const    { return x*V.x + y*V.y; }
}; // ..........................................................................
```

```
Wektor Wektor::operator – ( Wektor W ) const
{
  W.x = x – W.x;     W.y = y – W.y;
  return W;
}
```

Definicje klasy *Wektor*

```
class Wektor { // .....................................................
 public:
  float  x, y;

  Wektor operator – ( Wektor ) const ;
  float  operator * ( const Wektor &V ) const   { return x*V.y – y*V.x; }
  float  operator & ( const Wektor &V ) const   { return x*V.x + y*V.y; }
}; // ....................................................................
```

```
Wektor  Wektor::operator – ( Wektor W ) const
{
  W.x = x – W.x;      W.y = y – W.y;
  return  W;
}
```

Plan prezentacji

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) const ;
      ...
}; // .....

bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) const ;
    ...
}; // .....

bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\ (\mathbf{r} \times \mathbf{V}_L)_z\ |}{||\ \mathbf{V}_L\ ||}$$

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) const ;
    ...
}; // .....

bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\ (\mathbf{r} \times \mathbf{V}_L)_z\ |}{||\ \mathbf{V}_L\ ||}$$

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) const ;
    ...
}; // .....

bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\ (\mathbf{r} \times \mathbf{V}_L)_z\ |}{||\ \mathbf{V}_L\ ||}$$

Definicja klasy *PlatformaMobilna*

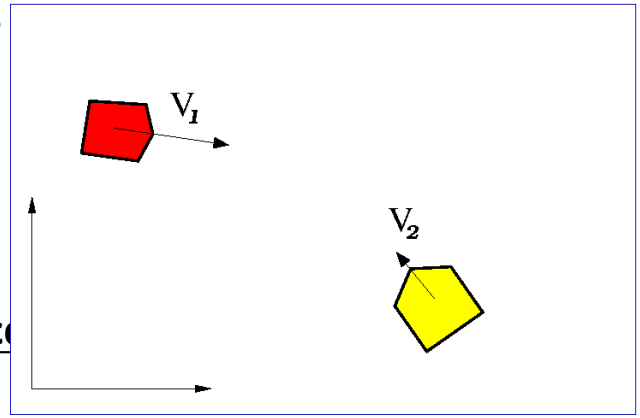
```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) const ;
        ...
}; // ..........................................................
```

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\ (\mathbf{r} \times \mathbf{V}_L)_z\ |}{||\ \mathbf{V}_L\ ||}$$

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) c
       ...
}; // ..........................................................
```

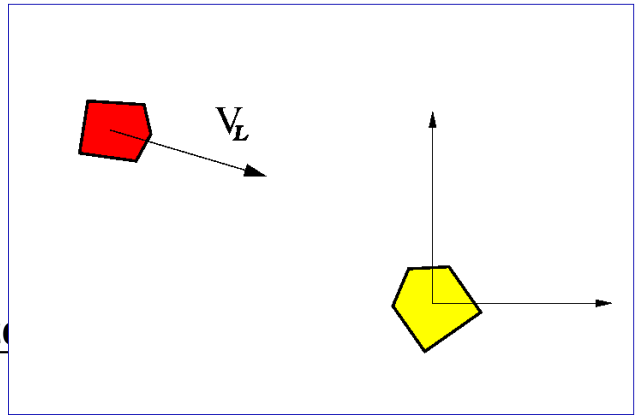



```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// ..........
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) c
       ...
}; // ..........
```



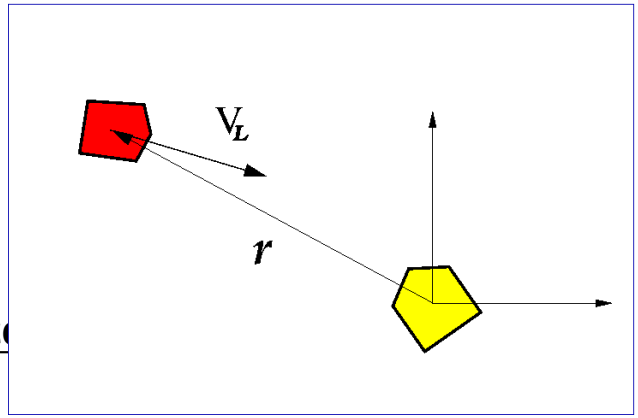

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc – this–>_Predkosc;
  Wektor r = PM._Pozycja – this–>_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this–>_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Wyliczenie prędkości platformy 1 w lokalnym układzie współrzędnych platformy nr 2.

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) c
       ...
}; // .....
```



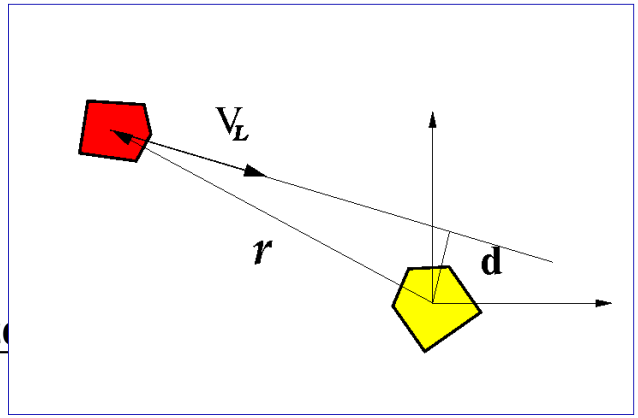

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc – this->_Predkosc;
  Wektor r = PM._Pozycja – this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Wyliczenie współrzędnych wektora poprowadzone z platformy nr 2 do platformy nr 1.

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// ..........
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) c
       ...
}; // ..........
```



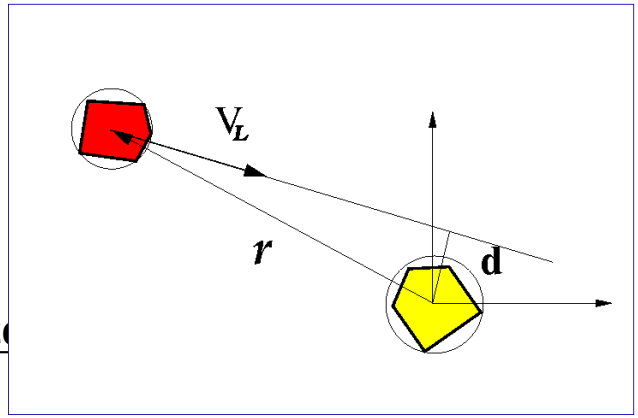

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc – this->_Predkosc;
  Wektor r = PM._Pozycja – this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Wyliczenie najmniejszej odległości między środkami platform.

Definicja klasy *PlatformaMobilna*

```
class PlatformaMobilna {// .....
  Wektor _Pozycja;
  Wektor _Predkosc;
  float  _R;
 public:
  bool CzyKolizja( const PlatformaMobilna&, float& ) c
       ...
}; // ..........................................................
```




```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna &PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return d < PM._R+this->_R;
}
```

$$d = \frac{|\,(\mathbf{r} \times \mathbf{V}_L)_z\,|}{||\,\mathbf{V}_L\,||}$$

Porównanie rozmiarów sumy obrysów obu platform z odległością ich największego zbliżenia.

Prównanie – metody wersus operatory

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna& PM, float &d) const
{
  Wektor V_L = PM._Predkosc.Odejmij(this->_Predkosc);
  Wektor r = PM._Pozycja.Odejmij(this->_Pozycja);
  d = fabs(r.IloczynWektZ(V_L))/sqrt(V_L.IloczynSkal(V_L));
  return   d < PM._R + this->_R;
}
```

```
bool PlatformaMobilna::CzyKolizja( const PlatformaMobilna& PM, float &d) const
{
  Wektor V_L = PM._Predkosc - this->_Predkosc;
  Wektor r = PM._Pozycja - this->_Pozycja;
  d = fabs(r * V_L)/sqrt(V_L & V_L);
  return   d < PM._R+this->_R;
}
```

Użycie przeciążeń operatorów daje bardziej zwarty, przejrzysty i bardziej intuicyjny zapis działań.

Plan prezentacji

Konstruktory i destruktory

```
struct PustaKlasa { //.....

        PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
        ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....


int  main( )
{
     cout <<  "Witajcie !!! :-)" <<  endl;
     PustaKlasa  Ob;
     cout <<  "Zegnajcie !!! :-(" <<  endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
Zegnajcie !!! :-(
--PustaKlasa
```

W przypadku zmiennych lokalnych (automatycznych) konstruktor wywoływany jest w chwili tworzenia obiektu, tj. w miejscu jego definicji. Destruktor natomiast jest wywoływany w miejscu końca zakresu ważności definicji obiektu.

Plan prezentacji

Obiekty tworzone dynamicznie

```
struct PustaKlasa { //.....
    PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
    ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....
```

```
int main( )
{
    cout << "Witajcie !!! :-)" << endl;
    PustaKlasa *wOb = new PustaKlasa;
    cout << "Zegnajcie !!! :-(" << endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
Zegnajcie !!! :-(
```

W przypadku obiektów tworzonych w sposób dynamiczny, to programista decyduje kiedy je zniszczyć. Zakończenie programu bez wykonania tej operacji nie spowoduje automatycznego uruchomienia destruktorów tych obiektów.

Obiekty tworzone dynamicznie

```
struct PustaKlasa { //.....
    PustaKlasa( ) { cout << "++PustaKlasa" << endl; }
    ~PustaKlasa( ) { cout << "--PustaKlasa" << endl; }
}; //.....
```

```
int main( )
{
    cout << "Witajcie !!! :-)" << endl;
    PustaKlasa *wOb = new PustaKlasa;
    delete wOb;
    cout << "Zegnajcie !!! :-(" << endl;
}
```

Wynik działania

```
Witajcie !!! :-)
++PustaKlasa
--PustaKlasa
Zegnajcie !!! :-(
```

Operator **delete** można zastosować tylko do zmiennej wskaźnikowej w dowolnym miejscu programu. Jedynym warunkiem poprawności jego wywołania jest to, aby pod danym adresem istniał obiekt utworzony dynamicznie z wykorzystaniem operatora **new**, albo też zmienna wskaźnikowa miała wartość **nullptr**.

Tablice obiektów tworzone dynamicznie

```
struct PustaKlasa { //.....................................................
    PustaKlasa( ) { }
    ~PustaKlasa( ) { }
}; //.....................................................................


int main( )
{
    int          *wTab = new int[ 30 ];
    PustaKlasa   *wOb  = new PustaKlasa[ 20 ];

    delete [ ] wTab;
    delete [ ] wOb;
}
```

W przypadku tworzenia w sposób dynamiczny tablic obiektów stosując operator **delete** należy jawnie wskazać, że usuwana jest tablica obiektów, a nie pojedynczy obiekt. Rozmiar tablicy nie jest istotny.

Plan prezentacji

Konstruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( ) { cout << "++KolaPrzednie" << endl; }
}; //.....

struct KolaTylnie { //.....
    KolaTylnie( ) { cout << "++KolaTylnie" << endl; }
}; //.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie  _KolaPrz;
    KolaTylnie    _KolaTyl;
    Pojazd4Kolowy( ) { cout << "++Pojazd4Kolowy" << endl; }
}; //.....

int main( )
{
    Pojazd4Kolowy  Ob;
}
```

Konstruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( ) { cout << "++KolaPrzednie" << endl; }
}; //.....

struct KolaTylnie { //.....
    KolaTylnie( ) { cout << "++KolaTylnie" << endl; }
}; //.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie  _KolaPrz;
    KolaTylnie    _KolaTyl;
    Pojazd4Kolowy( ) { cout << "++Pojazd4Kolowy" << endl; }
}; //.....

int main( )
{
    Pojazd4Kolowy  Ob;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
```

Pola inicjalizowane są zgodnie z kolejnością ich deklaracji w klasie.

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; //~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; //~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; //~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
```

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;
    Pojazd4Kolowy( )  { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
"Witajcie i zegnajcie !!!"
```

Konstruktory i destruktory w klasach złożonych

```
struct KolaPrzednie { //.....
    KolaPrzednie( )   { cout << "++KolaPrzednie" << endl; }
}; // ~KolaPrzednie( ) { cout << "--KolaPrzednie" << endl; }.....

struct KolaTylnie { //.....
    KolaTylnie( )   { cout << "++KolaTylnie" << endl; }
}; // ~KolaTylnie( ) { cout << "--KolaTylnie" << endl; }.....

struct Pojazd4Kolowy { //.....
    KolaPrzednie      _KolaPrz;
    KolaTylnie        _KolaTyl;

    Pojazd4Kolowy( )   { cout << "++Pojazd4Kolowy" << endl; }
}; // ~Pojazd4Kolowy( ) { cout << "--Pojazd4Kolowy" << endl; }.....

int main( )
{
    Pojazd4Kolowy  Ob;
    cout << "Witajcie i zegnajcie !!!" << endl;
}
```

Wynik działania

```
++KolaPrzednie
++KolaTylnie
++Pojazd4Kolowy
"Witajcie i zegnajcie !!!"
--Pojazd4Kolowy
--KolaTylnie
--KolaPrzednie
```

Destrukcja następuje w kolejności odwrotnej.

Plan prezentacji

Dwa sposoby inicjalizacji

```
class Kolor { // .....
        int _Kod_Koloru;

    public :
        Kolor( );
}; .....
```

Dwa sposoby inicjalizacji

```
class Kolor { //.....
        int  _Kod_Koloru;

    public :
        Kolor( );
};.....
```

```
Kolor::Kolor( )
 {
     _Kod_Koloru = 0;
 }
```

```
Kolor::Kolor( ): _Kod_Koloru(0)
 { }
```

Pole klasy może zostać zainicjalizowane na dwa sposoby, albo w ciele konstruktora, albo też poprzez listę inicjalizacyjną. Niektóre rodzaje pól mogą być zainicjalizowane tylko i wyłącznie z poziomu listy inicjalizacyjnej.

Lista inicjalizacyjna – różniece i podobieństwa

```
int   NrTeczy;
NrTeczy = 0;
```

```
Kolor::Kolor( )
 {
      _Kod_Koloru = 0;
 }
```

```
int   NrTeczy  =  0;
```

```
Kolor::Kolor( ): _Kod_Koloru(0)
 { }
```

Umieszczenie inicjalizatora w liście inicjalizacyjnej konstruktora powoduje przypisanie wartości polu w momencie jego utworzenia.

Lista inicjalizacyjna – parametryzacja

```
class Kolor { //.....
        int _Kod_Koloru;

    public :
        Kolor( int Param );
}; .....

Kolor::Kolor( int Kod ): _Kod_Koloru(Kod)
  { }
```

Do inicjalizatorów można przekazać wartość poprzez parametr wywołania konstruktora.

Lista inicjalizacyjna – parametryzacja

```
class PrzedzialLiczb { // .....................................................
        int _WartoscMini;
        int _WartoscMaks;
    public :
        PrzedzialLiczb( int Mini, int Maks );
}; // ..........................................................................

PrzedzialLiczb::PrzedzialLiczb( int Mini, int Maks ):
    _WartoscMini(Mini), _WartoscMaks(Maks)
{ }
```

Lista inicjalizacyjna może zawierać dowolną ilość inicjalizatorów. Uruchamiane są one w kolejności tworzenia pól, która jest zgodna z porządkiem ich deklaracji w klasie. Nie jest błędem podanie inicjalizatorów w innej kolejności. Jednak nie wpłynie ona na kolejność ich wywołań.
Aby nie wprowadzać nieporozumień, kolejność inicjalizatorów powinna zawsze odpowiadać faktycznej kolejności ich uruchamiania.

Wywoływanie konstruktorów w liście inicjalizacyjnej

```
struct Wektor { //.....
        float x, y;
        Wektor( ) { x = y = 0; }
        Wektor( float xx, float yy ) { x = xx; y = xy; }
}; //.....

struct Odcinek { //.....
      Wektor _Po, _Pn;
      Odcinek( );
}; //.....

Odcinek::Odcinek( ): _Po(-1,-1), _Pn(1,1) ..... Lista inicjalizująca wymusza wywołanie konstruktorów
{ }
```

Lista inicjalizująca pozwala wymusić wywołanie zadanego konstruktora dla poszczególnych składników obiektu. Kolejność składników listy powinna odpowiadać kolejności deklaracji pól klasy.

Koniec prezentacji
Dziękuję za uwagę