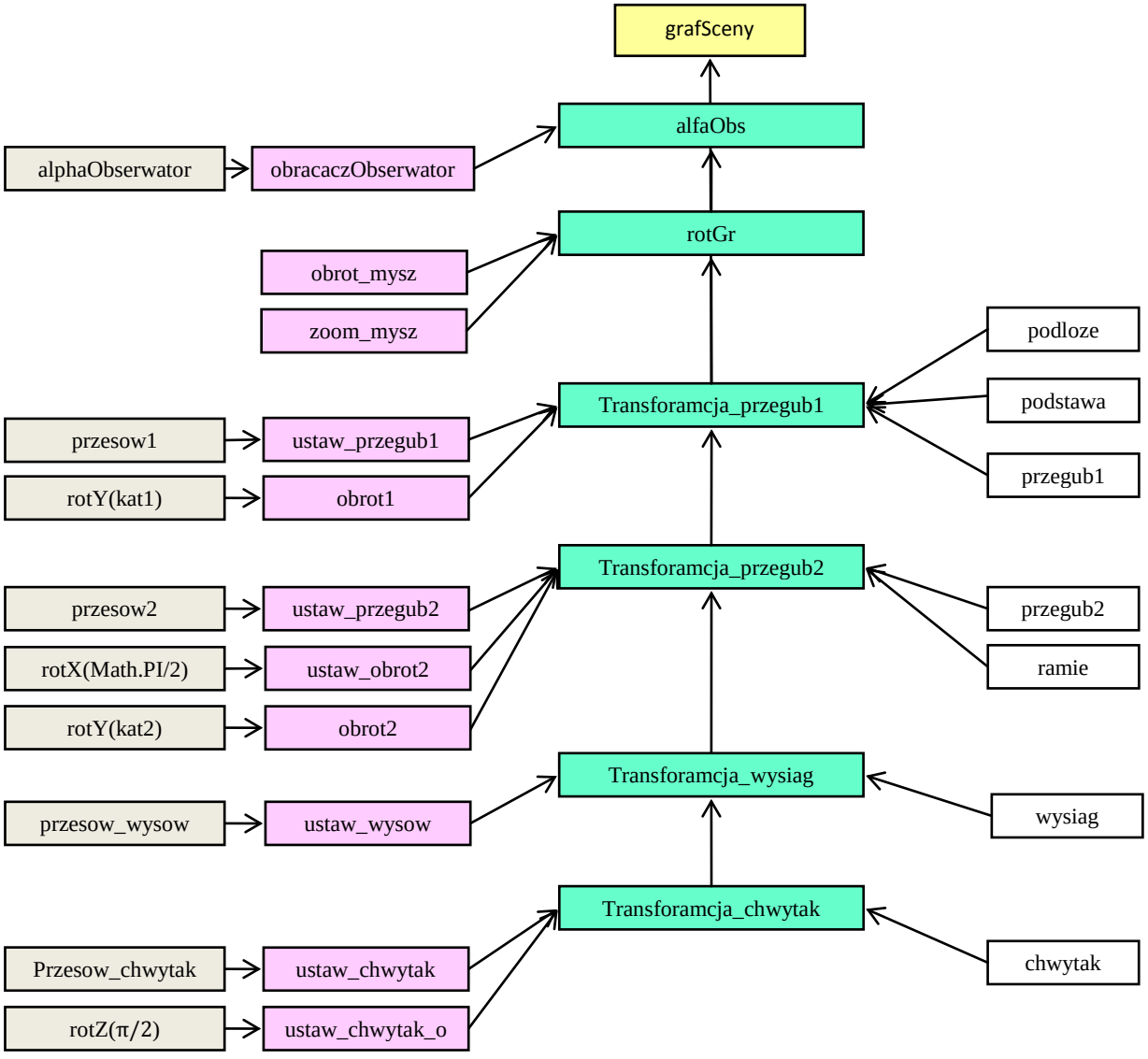
Michał Kurek 142889 16.06.14 r.
Maciej Latt 142890

Projekt z przedmiotu „Programowanie obiektowe i grafika komputerowa”

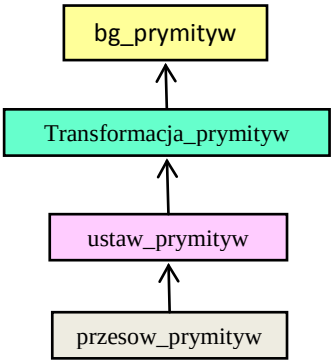
Robot typy „Polar Robot Arm”

Zgodnie z założeniami projektu utworzyliśmy animację robota typy „polar robot arm”. Obecnie zaimplementowana funkcjonalność to animacje obrotów robota wraz z wysuwającym się ramieniem, które posiada na końcu chwytak umożliwiający podnoszenie prymitywu. Nałożone zostały ograniczenia ruchu, przez co robot ma obszar roboczy zbliżony do rzeczywistego. Sterowanie obrotami pierwszego członu i przegubu jest zrealizowane za pomocą strzałek klawiatury, natomiast wysuwanie i chowanie ramienia, będącego drugim członem, odbywa się dzięki klawiszom W i S. Początkowa pozycja obserwatora została przesunięta i obrócona, aby lepiej pokazać ruchy ramienia. Robot bez chwytaka posiada trzy stopnie swobody, które są realizowane w programie według poniższego schematu:



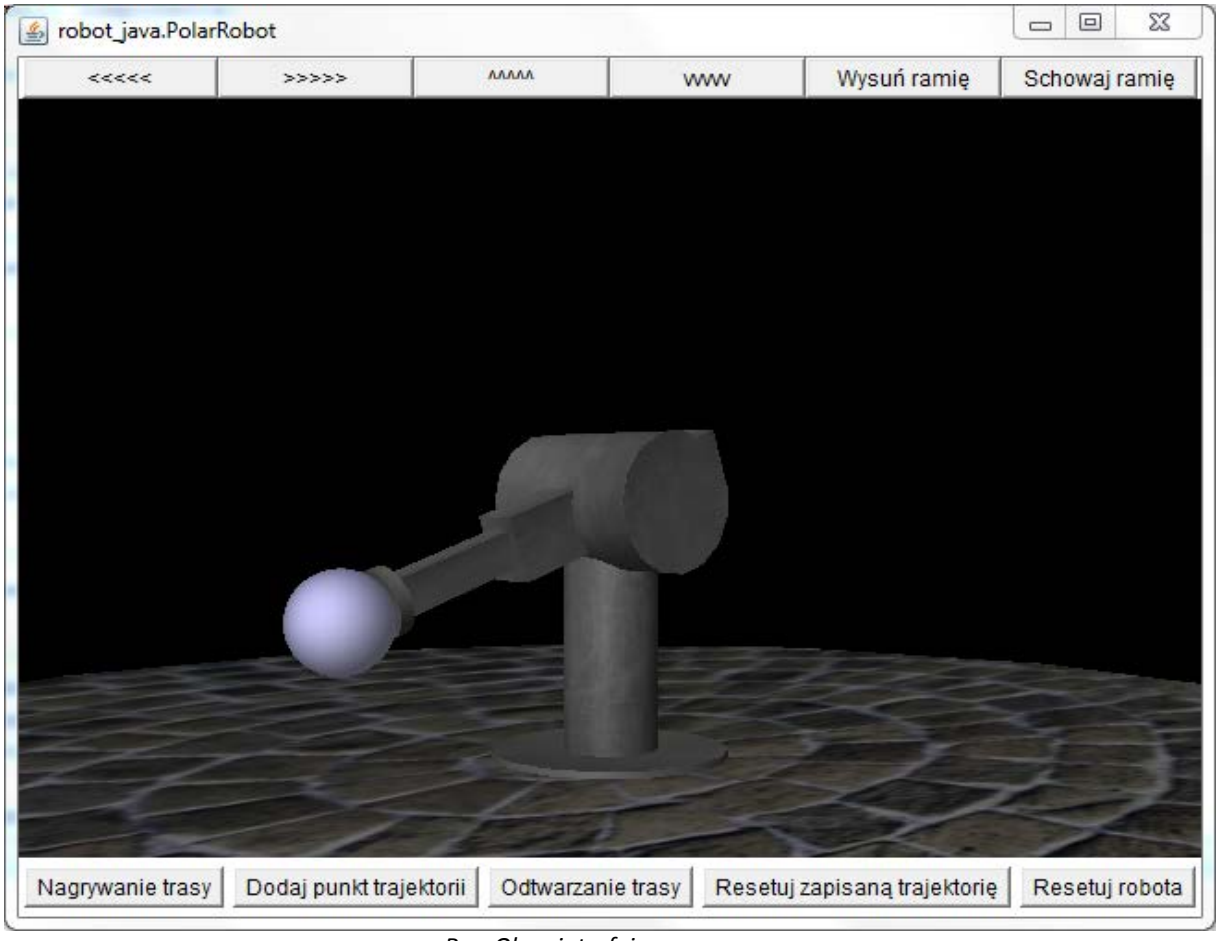
Schemat połączeń trzech przegubów robota „Polar Arm” wraz z chwytakiem

Michał Kurek 142889
Maciej Latt 142890
16.06.14 r.



Schemat połączeń obsługiwanego przez robota prymitywu

Żółty prostokąt na samej górze jest to obiekt typu BranchGroup, który zawiera w sobie odpowiednio przypisane obiekty typu TransformGroup (kolor niebieski prostokąta). Podstawa robota jest elementem nieruchomym, mającym środek identyczny z środkiem utworzonego wszechświata. Z tego powodu nie dodajemy jej do bezpośrednio do grafSceny, ale poprzez TransforGroup zawierający ruchy kamery. W programie zastosowaliśmy kaskadowe połączenie obiektów TransformGroup aby wszystkie elementy poczynając od podstawy posiadały niezbędne przesunięcia i obroty wymagane do uzyskania przez chwytak pozycji zadanej. Wszystkie elementy przesunięcia i obrotu został wyróżnione na schemacie kolorem filetowym zaś zmienne służące do translacji są przedstawione za pomocą prostokątów wypełnionych szarą barwą.



Rys. Okno interfejsu programu

Michał Kurek 142889 16.06.14 r.
Maciej Latt 142890

Uczenie robota

Robot posiada możliwość uczenia się metodą punkt po punkcie. Aby zacząć uczyć robota trzeba aktywować tą opcję poprzez kliknięcie w kontrolkę „Nagrywanie trasy” z dolnego panelu. Robot posiada możliwość zapamiętania do 5 punktów wyznaczonej trajektorii. Punkty dodajemy albo poprzez naciśnięcie przycisku „Dodaj punkt trajektorii” albo poprzez naciśnięcie klawisza „P” na klawiaturze. W momencie zapamiętywania punktu jest sczytywana wartość 3 współrzędnych: kat1, kat2, wysiąg. Odpowiadają one ustawieniom poszczególnych przegubów robota. Podczas uczenia robot nie widzi prymitywu, przenika przez niego, co umożliwia dokładne zapisanie wyznaczonej trasy. W trakcie nauki możemy zaprogramować moment przejścia przez obiekt, który chcemy chwycić oraz miejsce opuszczenia przedmioty. Do opuszczania przedmiot służy klawisz „spacja”, który jednocześnie wyznacza nowy punkt trajektorii. Po zapisaniu odpowiedniej trajektorii możemy odtworzyć zapisaną w pamięci robota trasę. Robot nie powtarza wiernie wykonanego ruchy podczas zapisywania, lecz wyłącznie zahacza o zapisane punkty. Robot sam oblicza optymalną trasę jaką musi pokonać pomiędzy poszczególnymi punktami. Robot został tak zaprogramowany aby chwycił prymityw jeżeli znajdzie się odpowiednio blisko chwytaka. Jest możliwe puszczenie przedmiotu, o ile w zapisanej trajektorii znajduje się taka informacja. Domyślną pozycją początkową i końcową robota jest pozycja „reset”. Zaimplementowany schemat jest często wykorzystywany przy zautomatyzowanych taśmach produkcyjnych, gdzie robot chwyta element, odkłada go na wyznaczone miejsce i znowu pobiera kolejny element. Schemat zapisu współrzędnych robota w danym punkcie za pomocą funkcji zapisz():

```
private void zapisz(int numer){
    if(nagrywanie_trwa == true){
        if(aktualny_zapis < 6){
        ruch_kat1[aktualny_zapis] = kat1;
        ruch_kat2[aktualny_zapis] = kat2;
        ruch_wysow[aktualny_zapis] = wysiag;
        if(numer == 1)
            ruch_puszczenie[aktualny_zapis] = 1;
        System.out.println("Zapamiętano " + aktualny_zapis + " ruch");
        aktualny_zapis += 1;
        } else System.err.println("Zapamiętana trajektoria jest pełna. Wymagany reset trajektorii.");
    }
}
```

Michał Kurek 142889 16.06.14 r.
Maciej Latt 142890

Wykrywanie kolizji

W programie została zaimplementowana obsługa kolizji za pomocą specjalnie do tego celu przeznaczonej klasy CollisionDetector. Obiektem wykrywającym kolizję jest nasz prymityw, który zostaje przyciągnięty do robota, jeżeli ten zbliży się odpowiednio blisko. Poprzez odpowiednią inicjalizację warunków wejścia oraz wyjścia ustalamy moment wykrycia przez system kolizji. Wykrycie kolizji dodatkowo zabezpiecza robota przed uderzeniem w prymityw. Jeżeli została stwierdzona kolizja to program pobiera kierunek, w którym ramię poruszało się ostatnio i blokuje dalszą możliwość wysuwu ramienia bądź jego obrotu. Dopiero po wykryciu bliskości ramienia prymityw będzie przyciągnięty przez chwytak i zostanie umożliwione przesuwnie się obiektu wraz z ruchami robota. Prymityw nabiera odpowiednich cech poprzez przełączenie go do Transformacja_chwytak z rotGr, która jest nieruchoma dla robota, za pomocą klawisza „spacja”. W grupie rotGr prymityw porusza się zgodnie z ruchami kamery. W tym momencie posiadamy zezwolenie na wszystkie ruchy robotem poprzez wyzerowanie flag odpowiadających za zezwolenia na obroty i przesuwy. W momencie puszczenia algorytm pobiera aktualną pozycję obiektu, jego wysokość oraz rzut współrzędnych na płaszczyznę XOY. Został zaimplementowany prosty algorytm grawitacji, przez co przedmioty przyspieszają po zrzuceniu z większej odległości.

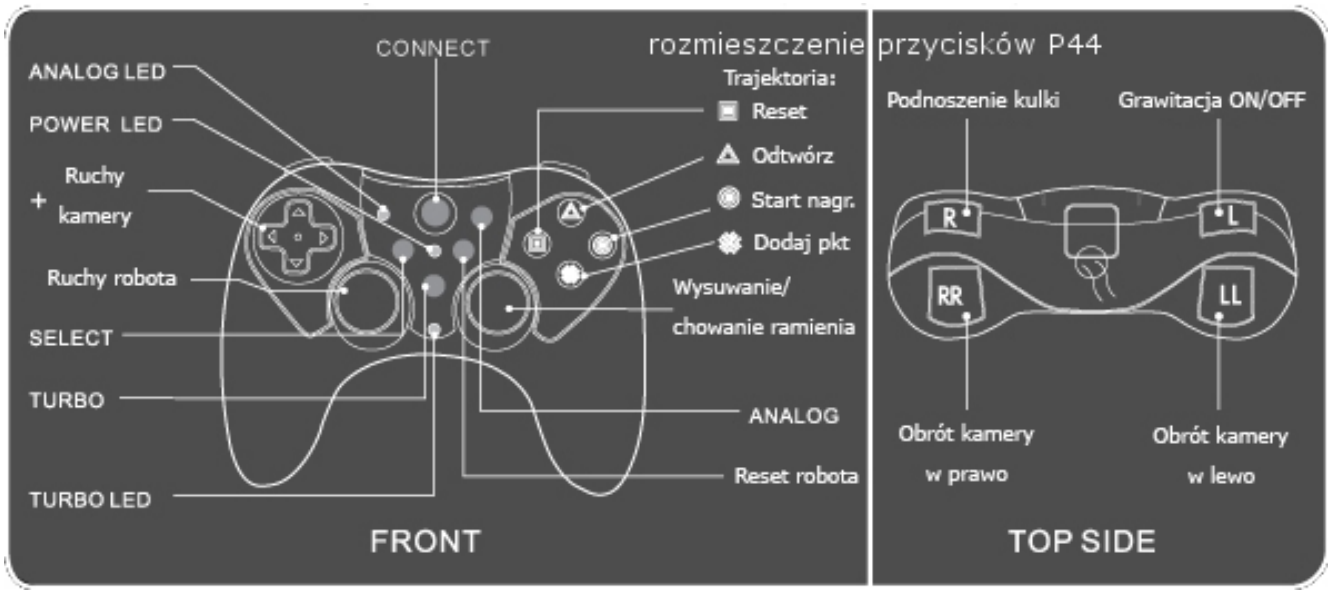
Ideowe rozwiązanie kolizji prezentuje poniższy fragment kodu:

```
if (inCollision) {
    System.out.println("Weszlo");
    wakeupOn(wExit);
    switch(PolarRobot.ostatni){
        case 1: PolarRobot.zabron_obrot1_prawo = true; break;
        case 2: PolarRobot.zabron_obrot1_lewo = true; break;
        case 3: PolarRobot.zabron_obrot2_dol = true; break;
        case 4: PolarRobot.zabron_obrot2_gora = true; break;
        case 5: PolarRobot.zabron_przesow_przod = true; break;
        case 6: PolarRobot.zabron_przesow_tyl = true; break;
    }
} else {
    System.err.println("Wyszlo");
    PolarRobot.zabron_obrot1_prawo = false;
    PolarRobot.zabron_obrot1_lewo = false;
    PolarRobot.zabron_obrot2_dol = false;
    PolarRobot.zabron_obrot2_gora = false;
    PolarRobot.zabron_przesow_przod = false;
    PolarRobot.zabron_przesow_tyl = false;
    wakeupOn(wEnter);
}
```

Michał Kurek 142889
Maciej Latt 142890 16.06.14 r.

Sterowanie

Całe sterowanie robotem oraz wykonywanie dodatkowych funkcji jest możliwa zarówno z panelu aplikacji, jak przy pomocy myszki oraz dodatkowe kontrolera, którym może być gamepad. Do obsługi programu został przystosowany gamepad Natec Genesis P44 – jego funkcjonalność opisuje poniższy schemat:



Schem. Rozkład funkcji przypisanych do klawiszy w Genesis P44

Jak widać lewa gałka odpowiada za ruchy robota, natomiast prawa umożliwia wysuwanie i chowanie ramienia. Hat Switch pozwala na ruchy kamerą na boki oraz góra/dół. Przyciskami z prawej strony kontrolera możemy wyznaczać nową trajektorię, zresetować obecną czy odtworzyć trajektorię zapisaną w pamięci robota. Przycisk R pozwala na podnoszenie prymitywu (w programie przedstawiony za pomocą kolorowej kulki). Przyciskiem L możemy natomiast wyłączyć lub włączyć grawitację – przy wyłączonej kulka upuszczona nad ziemią zawiśnie w powietrzu i zacznie spadać dopiero, gdy grawitacja zostanie włączona. Dzięki parze przycisków RR i LL możemy obracać całą sceną w osi pionowej przechodzącej przez ciało robota.

Struktura programu

Główna metoda programu jest postaci:

```
public static void main(String[] args) {
    PolarRobot robot = new PolarRobot();
    robot.addKeyListener(robot);
    MainFrame mf = new MainFrame(robot, 640, 480);
}
```

Jak widać tworzony jest w niej nowy obiekt klasy PolarRobot, do którego od razu dodawany jest KeyListener. Następnie tworzone jest okno aplikacji przy wykorzystaniu biblioteki import com.sun.j3d.utils.applet.MainFrame. Konstruktor klasy zawiera m.in. fragmenty odpowiadające za stworzenie kanwy, panelu oraz dodanie przycisków do panelu:

```
public PolarRobot(){
    setLayout(new BorderLayout());

    // stworzenie kanwy(?)
    canvas3D.setPreferredSize(new Dimension(840,680));
    add(BorderLayout.CENTER, canvas3D);
    canvas3D.addKeyListener(this);

    Panel panel = new Panel();
    add("South", panel);
    Panel sterowanie = new Panel(new GridLayout());
    add("North", sterowanie);

    panel.add(nagrywanie);
    nagrywanie.addActionListener(this);
    panel.add(dodajTraj);
    dodajTraj.addActionListener(this);
```

Dodatkowo wywołuje tworzenie grafu sceny i uruchamia timer z biblioteki java.util.Timer. Timer ten odpowiada za obsługę joysticku. Utworzone zadanie z java.util.TimerTask sprawdza co 30 ms jaki jest stan poszczególnych klawiszy w kontrolerze. Dzięki takiemu rozwiązaniu możemy jednocześnie nasłuchiwać wszystkie zdarzenia klawiatury oraz myszy i niezależnie obsługiwać rozkazy użytkownika.

```
BranchGroup scene = tworzenieGrafuSceny(u);

przesun_obserwatora.set(new Vector3f(x, y, z));
u.getViewingPlatform().getViewPlatformTransform().setTransform(przesun_obserwatora);
u.addBranchGraph(scene);

zegar.schedule(czasZad, 0, 30);
```

Funkcja tworzenieGrafuSceny(SimpleUniverse u) tworzy pełną scenę wizualizacji, tzn. ustawiane są materiały obiektów oraz nakładane tekstury:

```
/** Ustawienie wyglądu robota - nałożenie tekstury */
Appearance powloka = new Appearance();
powloka.setTexture(createTexture("img/robot_texture.png"));
TextureAttributes texAttr = new TextureAttributes();
texAttr.setTextureMode(TextureAttributes.BLEND);
texAttr.setPerspectiveCorrectionMode(TextureAttributes.NICEST);
powloka.setTextureAttributes(texAttr);

/** Ustawiamy materiał, żeby mieć cieniowanie */
Material material = new Material();
material.setSpecularColor(new Color3f(Color.WHITE));
material.setDiffuseColor(new Color3f(Color.WHITE));
powloka.setMaterial(material);
```

```
/** Wyglad prymitywu - materiał */
Material matPrym = new Material();
matPrym.setAmbientColor ( new Color3f( 0.30f, 0.30f, 0.35f ) );
matPrym.setDiffuseColor ( new Color3f( 0.30f, 0.30f, 0.50f ) );
matPrym.setSpecularColor ( new Color3f( 0.70f, 0.70f, 0.80f ) );
matPrym.setShininess( 2.4f );

/** Wyglad prymitywu - cieniowanie dla materiału */
ColoringAttributes wygPrymAtr = new ColoringAttributes();
wygPrymAtr.setShadeModel(ColoringAttributes.SHADE_GOURAUD);

/** Wyglad prymitywu - wygląd właściwy */
Appearance wygPrym = new Appearance();
wygPrym.setMaterial(matPrym);
wygPrym.setColoringAttributes(wygPrymAtr);

//Box prymityw = new Box(0.1f, 0.1f, 0.1f, powloka);
Sphere prymityw = new Sphere(0.1f, Sphere.GENERATE_NORMALS, 120, wygPrym);
ustaw_prymityw.set(przesow_prymityw);
Transformacja_prymityw = new TransformGroup(ustaw_prymityw);
Transformacja_prymityw.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
Transformacja_prymityw.addChild(prymityw);
bg_prymityw.addChild(Transformacja_prymityw);
rotGr.addChild(bg_prymityw);
```

Dodawane są także wszystkie obiekty jak widoczny powyżej prymityw oraz widoczny niżej jeden z elementów robota:

```
/**
 * Przegub robota przez, który przejdzie ramię. Przesunięty środek układu
 * współrzędnych sprawia, że można w prostu sposób obracać obiektem.
 */
Cylinder przegub2 = new Cylinder(0.15f, 0.4f,
        Cylinder.GENERATE_NORMALS | Cylinder.GENERATE_TEXTURE_COORDS,  powloka);
ustaw_przegub2.set(przesow2);
ustaw_obrot2.rotX(Math.PI/2);
ustaw_przegub2.mul(ustaw_obrot2);
Transformacja_przegub2 = new TransformGroup(ustaw_przegub2);
Transformacja_przegub2.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
Transformacja_przegub2.addChild(przegub2);
Transformacja_przegub1.addChild(Transformacja_przegub2);
```

Ważnym elementem jest tutaj przesunięcie elementu względem początku układu współrzędnych dzięki podaniu jako parametr konstruktora obiektu typu Transform3D ustaw_przegub2. Równie ważne jest zezwolenie na transformacje w funkcji setCapability(), gdyż bez tego niemożliwe byłoby sterowanie robotem.

Końcowym elementem tworzenia grafu sceny jest nałożenie obsługi kolizji, które zostały opisane wyżej w dokumentacji:

```
CollisionDetector detect = new CollisionDetector(prymityw);
detect.setSchedulingBounds(bounds);
grafSceny.addChild(detect);
```