Biblioteka graficzna XPCE

Dominik Szponder

Języki i Systemy Sztucznej Inteligencji,
Informatyka Stosowana, IV rok
PWSZ Tarnów

Tarnów, 12.01.2009

XPCE

XPCE jest zestawem narzędzi do tworzenia graficznego interfejsu użytkownika(GUI) zarówno w prologu, jak i w innych językach programowania. XPCE jest stworzone w taki sposób, że interfejs między nim a danym językiem programowania jest mały, co czyni XPCE bardzo dobrym narzędziem do tworzenia GUI dla wielu języków, w tym dla Prologu. Do tworzenia GUI XPCE używa własnej biblioteki o nazwie VWS(Virtual Windows System), definiującej metody do tworzenia i manipulacji oknami, prostymi obiektami graficznymi takimi jak linie, koła, tekst oraz obsługi zdarzeń. Dzięki takiemu właśnie podejściu programy używające XPCE są w pełni przenośne pomiędzy różnymi platformami.

XPCE cd.

Inną konsekwencją takiego rozwiązania, moze być to, że zachowanie aplikacji stworzonych przy pomocy XPCE może się różnić od innych aplikacji graficznych w tym samym systemie. XPCE zapewnia semantykę, którą można znaleść w wielu językach obiektowych, między innymi: klasy, obiekty, dziedziczenie, warunki. Wszystkie te udogodnienia reprezentowane są przez obiekty, które można tworzyć, modyfikować, przeglądać i niszczyć. Takie podejście zapewnia możliwość rozszerzania XPCE o nowe obiekty i metody. Ponadto procedury mogą być traktowane jak obiekty, które mogą być uruchamiane przez biblioteke. Zarówno VWS jak i wbudowane w XPCE klasy napisane są w ANSI-C.

Początki

XPCE, jest projektem który został zapoczatkowany pod nazwa PCE w 1985 r. przez Anjo Anjewirdena. Jego celem było stworzenie wysokopoziomowego interfejsu użytkownika dla Prologu. Wymagania dla tego srodowiska pochodziły z projektu "Thermodynamics Coach", w którym Paul Kamsteeg potrzebował PCE/Prolog by zaimplementować GUI w swoim systemie szkoleniowym. Miał on służyć do analizy rysunków wykonywanych przez studentów na dołączonym "scratch-padzie".

Rozwój

W następnych latach PCE było przebudowywane i zostało zaimplementowane na stacji roboczej SUNa przy użyciu Quintus Prolog, a następnie SWI-Prolog przez Wielmakera w 1996 r. w projekcie Esprit. Od wersji 4.0 do 4.5 PCE przeniesiono z platformy SunView do X-windows(skąd wieła się nazwa XPCE), a od wersji 4.7 utrzymywana jest zgodność z platformą Win32. Ponadto programiści dostali możliwość korzystania z wirtualnej maszyny, dzięki czemu możliwe stało się tworzenie nowych klas XPCE.

Rozwój cd.

W wersji 5 interfejs XPCE/Prolog został przebudowany, dzięki czemu poprawiła się wydajność i możliwe stało się wykorzystanie natywnych danych w Prologu, w klasach XPCE oraz wiązanie danych Prologu z obiektami XPCE. Dodano takze kilka nowych elementów graficznych. Począwszy od wersji 5.1 XPCE dostępne jest na licencji GPL-2, natomiast od wersji 6.0 na licencji LGPL.

Podstawowe predykaty

Istnieją cztery podstawowe predykaty używane do wykorzystywania XPCE w Prologu. Predykaty te odpowiadają podstawowym funkcjom maszyny wirtualnej: tworzeniu, manipulacji, odpytywaniu oraz niszczeniu obiektów. Zostały one przedstawione kolejno, poniżej:

- new
- send
- get
- free

Podstawowe predykaty

Istnieją cztery podstawowe predykaty używane do wykorzystywania XPCE w Prologu. Predykaty te odpowiadają podstawowym funkcjom maszyny wirtualnej: tworzeniu, manipulacji, odpytywaniu oraz niszczeniu obiektów. Zostały one przedstawione kolejno, poniżej:

- new
- send
- get
- free

Predykat new

Predykat new odpowiada za tworzenie obiektów graficznych w środowisku XPCE.

Konstrukcja predykatu new

new(?Reference, +NewTerm)

Pierwszy argument zwraca referencję do nowo utworzonego obiektu, będącą unikalnym identyfikatorem, natomiast drugi argument jest rodzajem obiektu(klasą) którą chcemy utworzyć. Referencję możemy utworzyć na dwa sposoby. Pierwszym z nich jest utworzenie referencji za pomocą new, natomiast drugim sposobem jest zadanie(nazwanie) referencji, czyli utworzenie tzw. named reference.

Predykat new

Predykat new odpowiada za tworzenie obiektów graficznych w środowisku XPCE.

Konstrukcja predykatu new

new(?Reference, +NewTerm)

Pierwszy argument zwraca referencję do nowo utworzonego obiektu, będącą unikalnym identyfikatorem, natomiast drugi argument jest rodzajem obiektu(klasą) którą chcemy utworzyć. Referencję możemy utworzyć na dwa sposoby. Pierwszym z nich jest utworzenie referencji za pomocą new, natomiast drugim sposobem jest zadanie(nazwanie) referencji, czyli utworzenie tzw. named reference.

Predykat new

Predykat new odpowiada za tworzenie obiektów graficznych w środowisku XPCE.

Konstrukcja predykatu new

new(?Reference, +NewTerm)

Pierwszy argument zwraca referencję do nowo utworzonego obiektu, będącą unikalnym identyfikatorem, natomiast drugi argument jest rodzajem obiektu(klasą) którą chcemy utworzyć. Referencję możemy utworzyć na dwa sposoby. Pierwszym z nich jest utworzenie referencji za pomocą new, natomiast drugim sposobem jest zadanie(nazwanie) referencji, czyli utworzenie tzw. named reference.

Referencja

Referencja utworzona za pomocą new:

Przykład

```
new(P, point(20,30)).
```

Przykład ten tworzy instancję klasy point z argumentami 20 oraz 30. Referencja jest prezentowana w prologu przy użyciu prefiksowego operatora @. Dla referencji generowanych przez XPCE argumentem @ jest unikalna liczba całkowita.

Referencja

Referencja utworzona za pomocą new:

Przykład

```
new(P, point(20,30)).
```

Przykład ten tworzy instancję klasy point z argumentami 20 oraz 30. Referencja jest prezentowana w prologu przy użyciu prefiksowego operatora @. Dla referencji generowanych przez XPCE argumentem @ jest unikalna liczba całkowita.

Referencja

Referencja utworzona za pomocą new:

Przykład

```
new(P, point(20,30)).
```

Przykład ten tworzy instancję klasy point z argumentami 20 oraz 30. Referencja jest prezentowana w prologu przy użyciu prefiksowego operatora @. Dla referencji generowanych przez XPCE argumentem @ jest unikalna liczba całkowita.

Referencja nazwana

Przykład

```
new(@przyklad, dialog('Witaj świecie')).
```

Referencja ta tworzy obiet dialog. Obiekt ten jest oknem przeznaczonym do wyświetlania kontrolerów takich jak przyciski, pola tekstowe, itp. W tym wypadku referencja została wyspecyfikowana, i przyjmuje ona postać @przykład. Dzięki temu XPCE powiąże stworzony obiekt z tą referencją. Drugim argumentem w powyższym przykładzie jest term. Funktor ten jako argument przyjmuje nazwę klasy, której instancja zostaje stworzona. Ewentualne parametry dla tej klasy oznaczają parametry dla tworzonej instancji. Każdy nowo tworzony obiekt wypełniany jest wartościami domyślnymi oraz wartościami parametrów, jeśli te zostały podane.

Referencja nazwana

Przykład

```
new(@przyklad, dialog('Witaj świecie')).
```

Referencja ta tworzy obiet dialog. Obiekt ten jest oknem przeznaczonym do wyświetlania kontrolerów takich jak przyciski, pola tekstowe, itp. W tym wypadku referencja została wyspecyfikowana, i przyjmuje ona postać @przykład. Dzięki temu XPCE powiąże stworzony obiekt z tą referencją. Drugim argumentem w powyższym przykładzie jest term. Funktor ten jako argument przyjmuje nazwę klasy, której instancja zostaje stworzona. Ewentualne parametry dla tej klasy oznaczają parametry dla tworzonej instancji. Każdy nowo tworzony obiekt wypełniany jest wartościami domyślnymi oraz wartościami parametrów, jeśli te zostały podane.

Predykat send

Do modyfikacji utworzonych obiektów służy predykat send.

Konstrukcja send

send(+Receiver, +Selector(... Args ...)).

Pierwszym argumentem tego predykatu jest referencja do obiektu, natomiast drugi argument to term. Term ten jest funktorem, którego nazwa jest nazwą metody do wywoływania a argumenty to argumenty do operacji.

Predykat send

Do modyfikacji utworzonych obiektów służy predykat send.

Konstrukcja send

send(+Receiver, +Selector(... Args ...)).

Pierwszym argumentem tego predykatu jest referencja do obiektu, natomiast drugi argument to term. Term ten jest funktorem, którego nazwa jest nazwą metody do wywoływania a argumenty to argumenty do operacji.

Predykat send

Do modyfikacji utworzonych obiektów służy predykat send.

Konstrukcja send

send(+Receiver, +Selector(... Args ...)).

Pierwszym argumentem tego predykatu jest referencja do obiektu, natomiast drugi argument to term. Term ten jest funktorem, którego nazwa jest nazwą metody do wywoływania a argumenty to argumenty do operacji.

Użycie send

Przykład

send(@przyklad, append(text_item(name))).

Przykład ten wywołuje metodę append klasy dialog. Metoda ta dodaje komponent do okienka dialogowego. Komponent jest opisany przez term text_item(name), który to jest konwertowany na obiekt tak, jak drugi argument predykatu new. W celu obejrzenia efektów należy użyć predykatu send z argumentem open.

Przykład

send(@przyklad, open).

Dzięki temu wywołaniu zobaczymy stworzone przez nas okienko.

Użycie send

Przykład

```
send(@przyklad, append(text_item(name))).
```

Przykład ten wywołuje metodę append klasy dialog. Metoda ta dodaje komponent do okienka dialogowego. Komponent jest opisany przez term text_item(name), który to jest konwertowany na obiekt tak, jak drugi argument predykatu new. W celu obejrzenia efektów należy użyć predykatu send z argumentem open.

Przykład

```
send(@przyklad, open).
```

Dzięki temu wywołaniu zobaczymy stworzone przez nas okienko.

Użycie send

Przykład

```
send(@przyklad, append(text_item(name))).
```

Przykład ten wywołuje metodę append klasy dialog. Metoda ta dodaje komponent do okienka dialogowego. Komponent jest opisany przez term text_item(name), który to jest konwertowany na obiekt tak, jak drugi argument predykatu new. W celu obejrzenia efektów należy użyć predykatu send z argumentem open.

Przykład

```
send(@przyklad, open).
```

Dzięki temu wywołaniu zobaczymy stworzone przez nas okienko.

Użycie send

Przykład

```
send(@przyklad, append(text_item(name))).
```

Przykład ten wywołuje metodę append klasy dialog. Metoda ta dodaje komponent do okienka dialogowego. Komponent jest opisany przez term text_item(name), który to jest konwertowany na obiekt tak, jak drugi argument predykatu new. W celu obejrzenia efektów należy użyć predykatu send z argumentem open.

Przykład

```
send(@przyklad, open).
```

Dzięki temu wywołaniu zobaczymy stworzone przez nas okienko.

Predykat get

Predykatem służącym do pobierania informacji o stanie obiektów jest predykat get.

Konstrukcja get

get(+Receiver, +Selector(+Argument...), -Result))

Pierwsze dwa argumenty są identyczne jak dla send, natomiast trzeci argument unifikowany jest ze zwracaną wartością. Wartość ta jest zazwyczaj referencją na obiekt, z wyjątkiem obiektów klasy name, które to obiekty zwracane są jako atomy prologu, XPCE int, które tłumaczone są na liczby całkowite Prologu oraz obiektów XPCE real, które są konwertowane na liczby zmiennoprzecinkowe.

Predykat get

Predykatem służącym do pobierania informacji o stanie obiektów jest predykat get.

Konstrukcja get

get(+Receiver, +Selector(+Argument...), -Result))

Pierwsze dwa argumenty są identyczne jak dla send, natomiast trzeci argument unifikowany jest ze zwracaną wartością. Wartość ta jest zazwyczaj referencją na obiekt, z wyjątkiem obiektów klasy name, które to obiekty zwracane są jako atomy prologu, XPCE int, które tłumaczone są na liczby całkowite Prologu oraz obiektów XPCE real, które są konwertowane na liczby zmiennoprzecinkowe.

Predykat get

Predykatem służącym do pobierania informacji o stanie obiektów jest predykat get.

Konstrukcja get

get(+Receiver, +Selector(+Argument...), -Result))

Pierwsze dwa argumenty są identyczne jak dla send, natomiast trzeci argument unifikowany jest ze zwracaną wartością. Wartość ta jest zazwyczaj referencją na obiekt, z wyjątkiem obiektów klasy name, które to obiekty zwracane są jako atomy prologu, XPCE int, które tłumaczone są na liczby całkowite Prologu oraz obiektów XPCE real, które są konwertowane na liczby zmiennoprzecinkowe.

Użycie get

Przykład

```
get(@przyklad, display, D).
```

Powyższy przykład zwraca obiekt display, na którym @przyklad jest wyswietlany. Jest to referencja na obiekt klasy display, która reprezentuje ekran.

Użycie get

Przykład

```
get(@przyklad, display, D).
```

Powyższy przykład zwraca obiekt display, na którym @przyklad jest wyswietlany. Jest to referencja na obiekt klasy display, która reprezentuje ekran.

Użycie get cd.

Poniżej zaprezentowano przykład odpowiadający za pobranie, oraz wyświetlenie za pomocą predykatu get rozdzielczości ekranu.

Przykład

```
get(@display, size, Size),
get(Size, width, Szerokosc),
get(Size, height, Wysokosc).
```

Użycie get cd.

Poniżej zaprezentowano przykład odpowiadający za pobranie, oraz wyświetlenie za pomocą predykatu get rozdzielczości ekranu.

Przykład

```
get(@display, size, Size),
get(Size, width, Szerokosc),
get(Size, height, Wysokosc).
```

Predykat free

Za niszczenie utworzonych przez nas obiektów odpowiada predykat free.

Konstrukcja free

free(+Reference)

Jego argumentem jest referencja na obiekt, którą uzyskujemy przy pomocy predykatów new, get. Dzięki temu predykatowi obiekt który usuwamy znika ze świata XPCE.

Predykat free

Za niszczenie utworzonych przez nas obiektów odpowiada predykat free.

Konstrukcja free

free(+Reference)

Jego argumentem jest referencja na obiekt, którą uzyskujemy przy pomocy predykatów new, get. Dzięki temu predykatowi obiekt który usuwamy znika ze świata XPCE.

Predykat free

Za niszczenie utworzonych przez nas obiektów odpowiada predykat free.

Konstrukcja free

free(+Reference)

Jego argumentem jest referencja na obiekt, którą uzyskujemy przy pomocy predykatów new, get. Dzięki temu predykatowi obiekt który usuwamy znika ze świata XPCE.

Użycie free

Przykład

```
free(@przyklad).
```

Przykład ten usuwa zarówno okno dialogowe, jak i przypisane okno z ekranu.

Przykład

```
free(@display).
```

Natomiast powyższy przykład pokazuje iż prolog chroni niektóre obiekty przed usunięciem.

Użycie free

Przykład

```
free(@przyklad).
```

Przykład ten usuwa zarówno okno dialogowe, jak i przypisane okno z ekranu.

Przykład

```
free(@display).
```

Natomiast powyższy przykład pokazuje iż prolog chroni niektóre obiekty przed usunięciem.

Użycie free

Przykład

```
free(@przyklad).
```

Przykład ten usuwa zarówno okno dialogowe, jak i przypisane okno z ekranu.

Przykład

```
free(@display).
```

Natomiast powyższy przykład pokazuje iż prolog chroni niektóre obiekty przed usunięciem.

Użycie free

Przykład

```
free(@przyklad).
```

Przykład ten usuwa zarówno okno dialogowe, jak i przypisane okno z ekranu.

Przykład

```
free(@display).
```

Natomiast powyższy przykład pokazuje iż prolog chroni niektóre obiekty przed usunięciem.

Pomoc

Ponieważ biblioteka XPCE jest bardzo dużego rozmiaru, ze środowiskiem został zintegrowany rozbudowany system pomocy, który możemy uruchomić za pomocą predykatu:

Wywołanie pomocy

```
manpce.
```

System ten zawiera dużą liczbę narzędzi umożliwiającą analizowanie różnych aspektów środowiska.

Pomoc

Ponieważ biblioteka XPCE jest bardzo dużego rozmiaru, ze środowiskiem został zintegrowany rozbudowany system pomocy, który możemy uruchomić za pomocą predykatu:

Wywołanie pomocy

```
manpce.
```

System ten zawiera dużą liczbę narzędzi umożliwiającą analizowanie różnych aspektów środowiska.

Pomoc

Ponieważ biblioteka XPCE jest bardzo dużego rozmiaru, ze środowiskiem został zintegrowany rozbudowany system pomocy, który możemy uruchomić za pomocą predykatu:

Wywołanie pomocy

```
manpce.
```

System ten zawiera dużą liczbę narzędzi umożliwiającą analizowanie różnych aspektów środowiska.

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

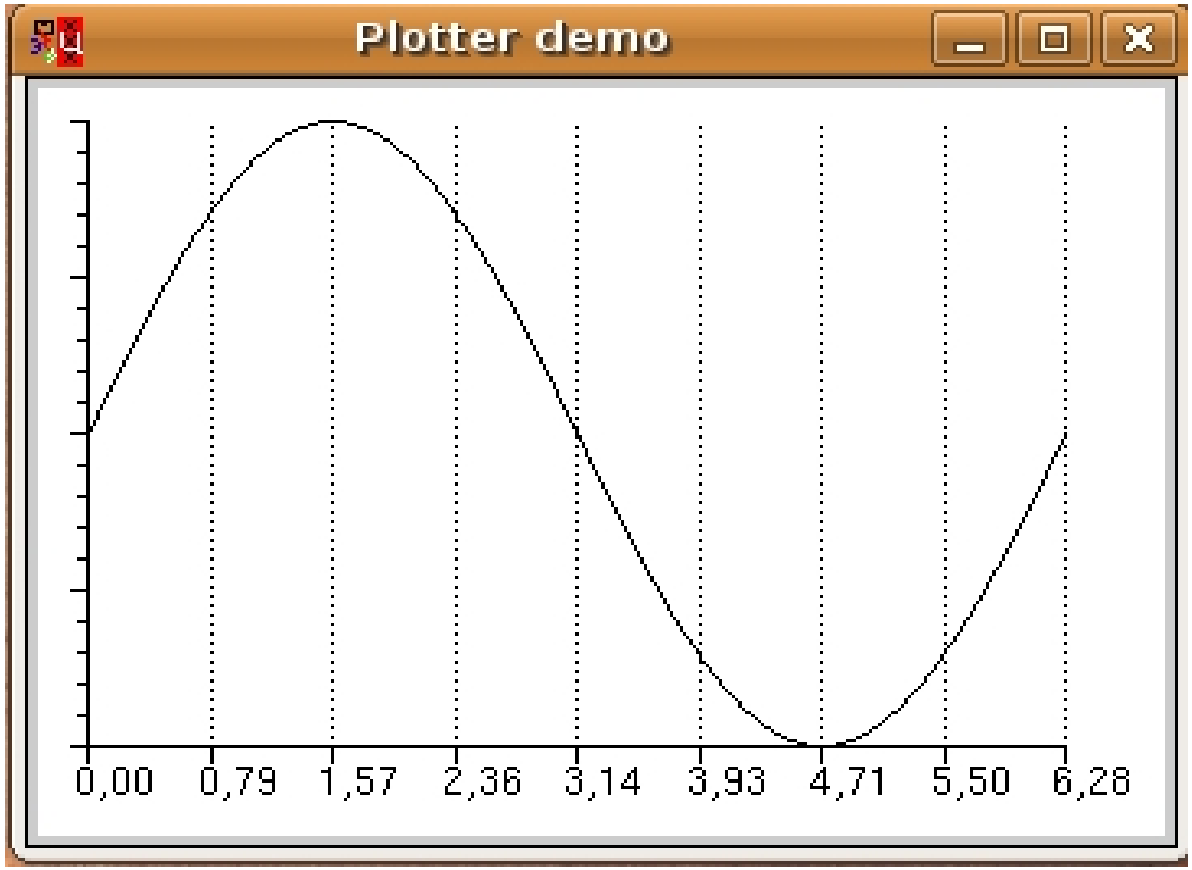
Pomoc cd.

W manpce wyróżnić możemy następujące elementy listy narzędzi dla XPCE:

- browsers/class hierarchy
- browsers/class browser
- browsers/search
- browsers/global objects
- browsers/XPCE/Prolog Predicates
- tools/visual hierarchy
- file/demo programs

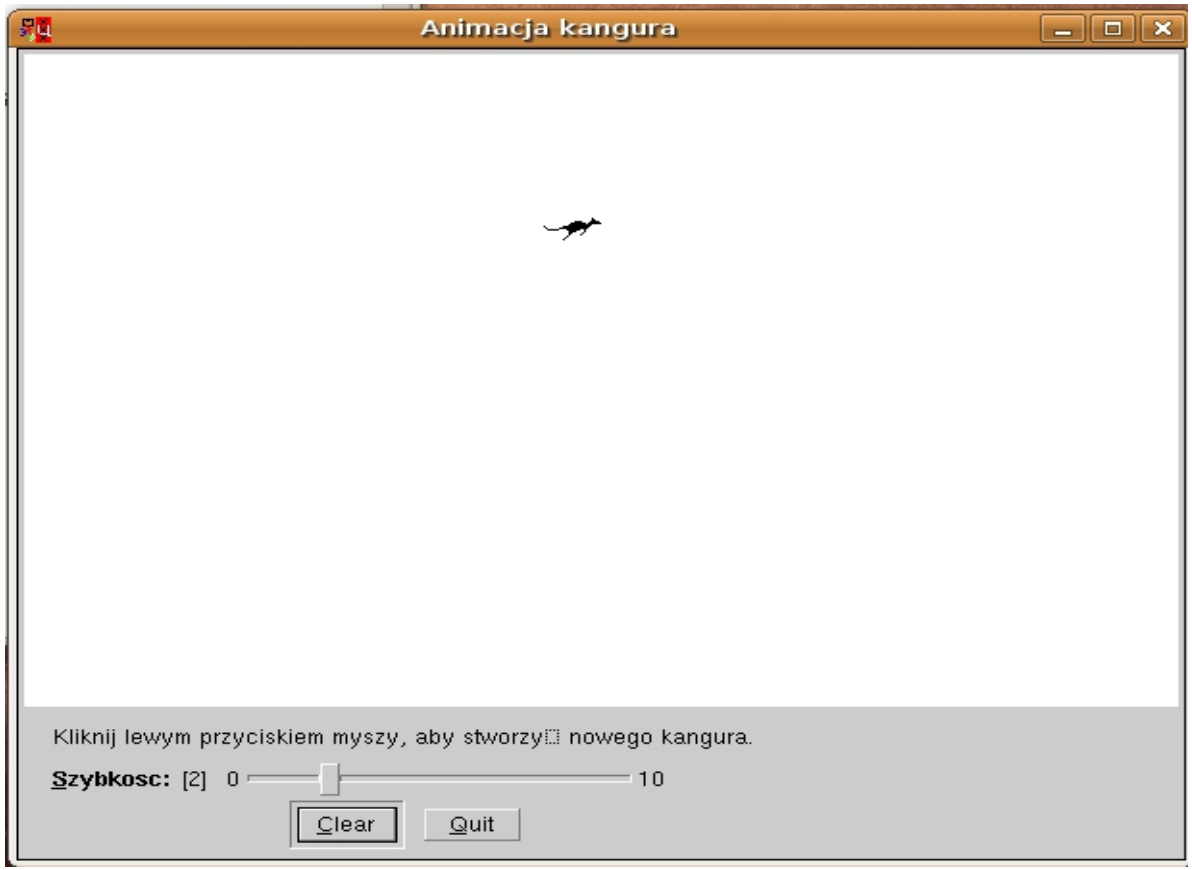
Przykład 1 - rysowanie wykresów

Wykres funkcji sinus(x).



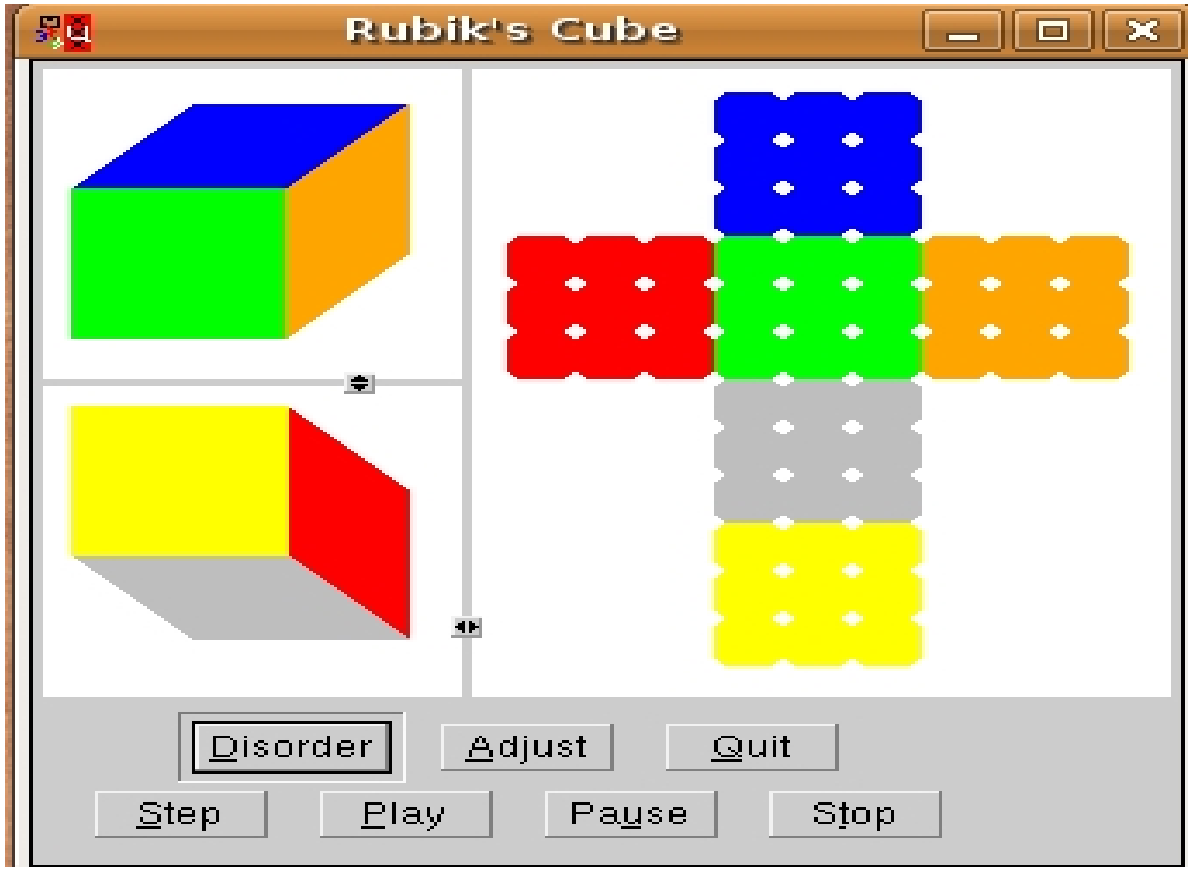
Przykład 2 - prosta animacja

Animacja Kangura.



Przykład 3 - Kostka Rubika

Kostka rubika.



Koniec

Dziękuję za uwagę.