AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca inżynierska

Zbigniew Baster

kierunek studiów: **Fizyk Medyczna**

Generowanie struktury trójwymiarowej białka dla zadanych wartości kątow dwuściennych Phi, Psi.

Opiekun: **prof. dr hab. Irena Roterman-Konieczna**

Kraków, styczeń 2012

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

Merytoryczna ocena pracy przez opiekuna

Końcowa ocena pracy przez opiekuna: . . . . .

Data: . . . . . Podpis: . . . . . .

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 - niedostateczna

Merytoryczna ocena pracy przez recenzenta

Końcowa ocena pracy przez opiekuna: . . . . .

Data: . . . . . Podpis: . . . . . .

Skala ocen: 5.0 – bardzo dobra, 4.5 – plus dobra, 4.0 – dobra, 3.5 – plus dostateczna, 3.0 – dostateczna, 2.0 - niedostateczna

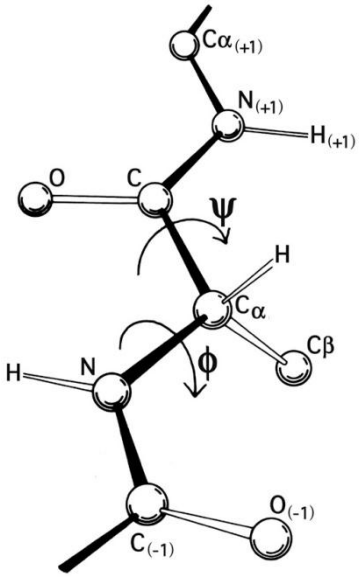
Chciałbym podziękować prof. dr hab. Irenie Roterman-Koniecznej
za pomoc przy powstawaniu niniejszej pracy,
za poświęcony czas oraz cenne wskazówki

Spis treści

1. Wstęp

1.1. Cel pracy

Celem niniejszej pracy jest opracowanie programu, którego funkcją będzie wykreowanie struktur drugo- i trzeciorzędowych białek o wcześniej zadanych parametrach w postaci kątów (wyrażonych w stopniach) φ i ψ dla każdego aminokwasu.



Ryc. 1. Położenie kątów dwuścienne w aminokwasie. [1]

Pliki wejściowe powinny zawierać informację o sekwencji aminokwasowej (kody trzy- i/lub jednoliterowe) oraz przypisane tym aminokwasom wartości kątów dwuściennych.

Pliki wynikowe programu powinny być w formacie PDB, czyli zawierają współrzędne kartezjańskie wszystkich atomów (poza atomami wodoru) wchodzących w skład danego białka w kolejności zgodnej ze standardami formatu PDB.

Program ten z założenia ma być wykorzystywany w procesie ustalania struktur drugo- i trzeciorzędowych białek natywnych.

Etap I:

Wytworzenie tzw. struktury „rozciągniętej” (kąty φ i ψ każdego z aminokwasów poza proliną, o wartości 180^{o}; w przypadku proliny kąty φ i ψ równe odpowiednio -75^{o} i 180^{o}).

Etap II:

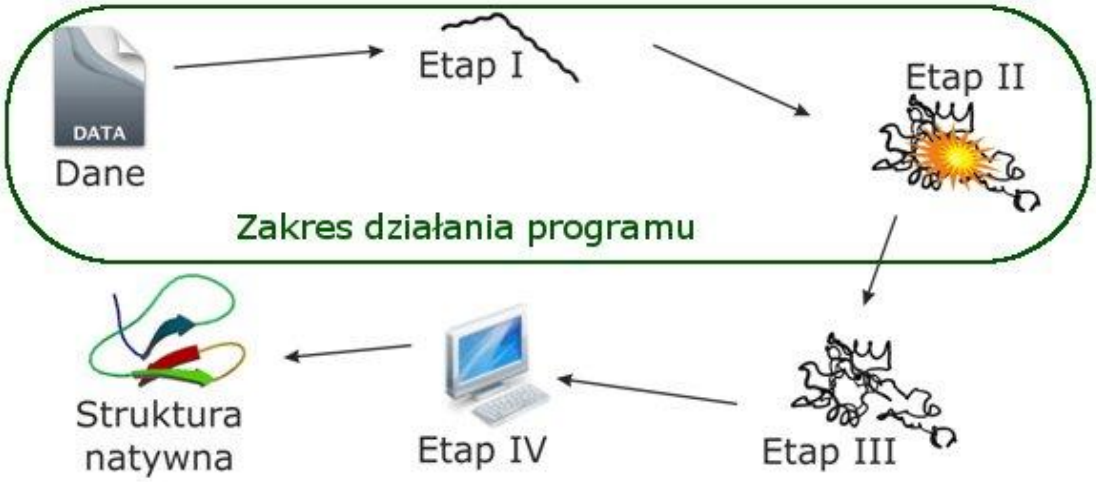
Wprowadzenie rotacji kątów wedle zadanych, kątów φ i ψ.

Etap III:

Usunięcie możliwych kolizji wynikających z nakładania się atomów bądź pozostających w zbyt małych odległościach międzyatomowych.

Etap IV:

Przeprowadzenie optymalizacji geometrii białka za pomącą przeznaczonych do tego programów.



Ryc. 2. Proces wyznaczania struktury natywnej białek. [2][3][4][5]

Program ten ma na celu realizację I i II etapu powyższego procesu.

1.2. Materiały

Program wykorzystuje dane zawierające współrzędne w układzie kartezjańskim każdego atomu w każdym aminokwasie biogennym w formie „rozciągniętej”. [6]

2. Metody

2.1. Język programowania

Niniejszy program został napisany w języku JAVA.

Wybór tego języka był spowodowany jego łatwością w przechowywaniu i obsługiwaniu dużej liczby elementów różnych typów oraz popularnością języka w środowisku programistycznym.

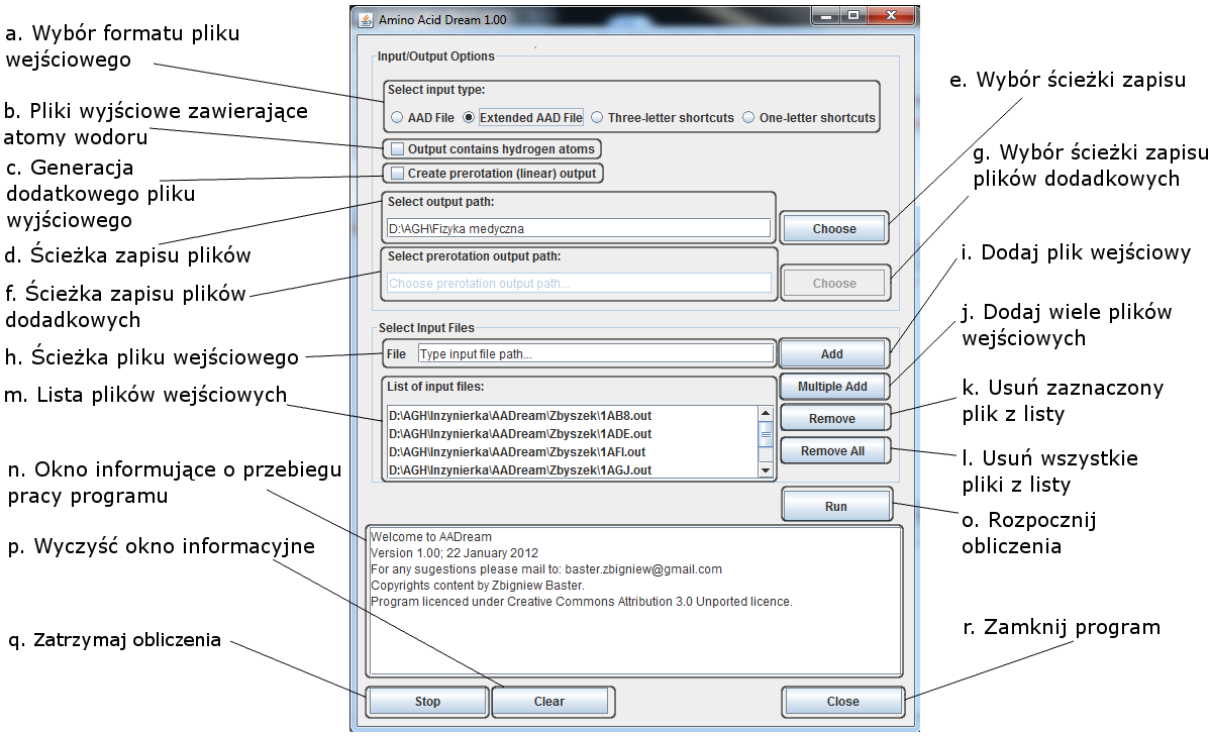
2.2. Opis działania programu

Program jest przeznaczony do systemów graficznych systemów operacyjnych. Uruchomienie programu w innym środowisku może powodować wystąpienie błędów.

Poniżej przedstawiono opis działania i funkcje programu:

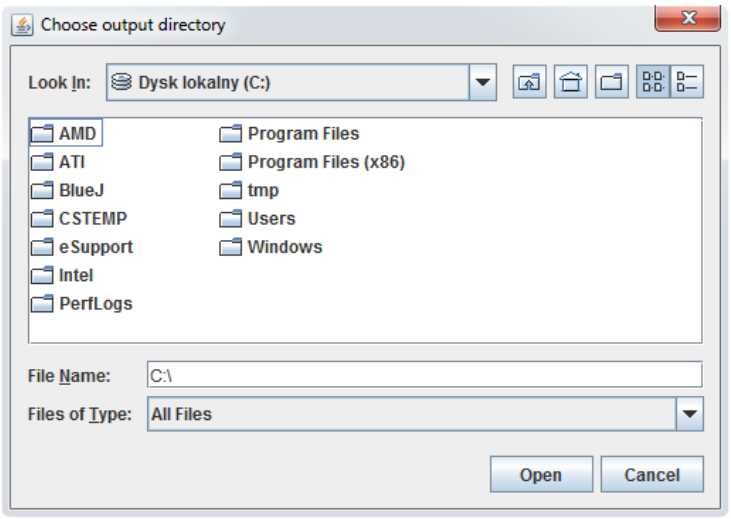
2.2.1. Funkcje programu

Po uruchomieniu programu pokazuje się główne okno przedstawione poniżej:



Rys. 1. Główne okno programu. [0]

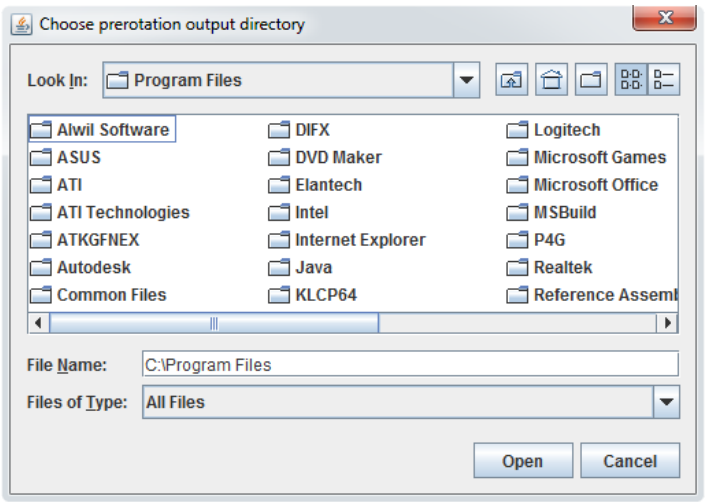
a. Wybór formatu pliku wejściowego programu. Więcej informacji na temat formatów plików wejściowych zamieszczono w punkcie 2.2.2.
b. Wybór czy pliki wyjściowe mają zawierać współrzędne atomów wodoru. Standardowo pliki formatu PDB nie zawierają atomów wodoru, jednak program przewiduje możliwość zawarcia ich w pliku wyjściowym.
c. Wybór czy program ma wytworzyć dodatkowo pliki zawierające „rozciągnięte" struktury białek. Opcja dostępna tylko dla formatów AAD i Extended AAD.
d. Wybór ścieżki zapisu plików wyjściowych, poprzez wpisanie jej z klawiatury.
e. Wybór ścieżki zapisu plików wyjściowych, poprzez wyznaczenie ścieżki za pomocą okna dialogowego (Rys.2.). W oknie dialogowy wyświetlone zostaną tylko katalogi.



Rys. 2. Okna dialogowe wyboru ścieżki zapisu pliku wyjściowego. [0]

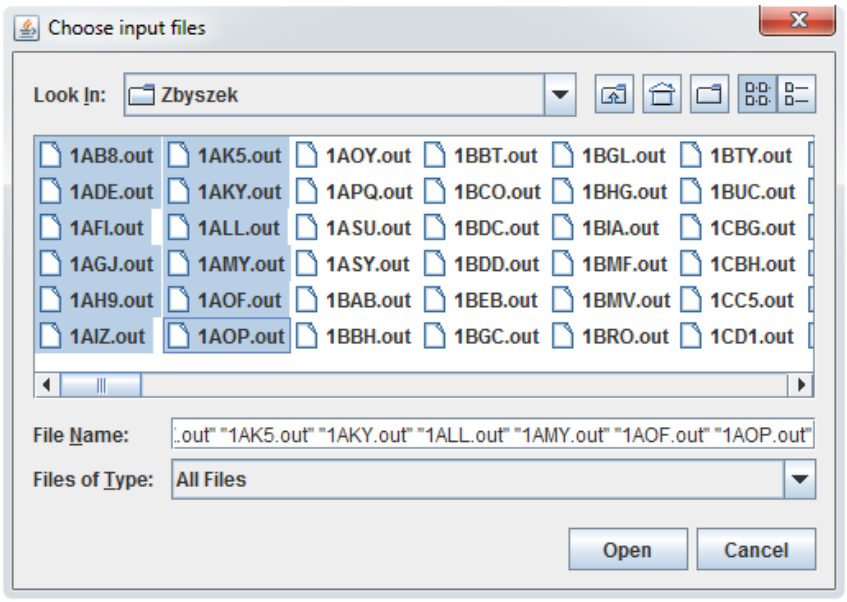
f. Wybór ścieżki zapisu plików zawierających „rozciągniętą" strukturę białka, poprzez wpisanie jej z klawiatury. Opcja dostępna tylko w przypadku wybrania opcji wytworzenia plików z zwierających strukturę „rozciągniętą".

g. Wybór ścieżki zapisu plików zawierających „rozciągniętą" strukturę białka, poprzez wyznaczenie ścieżki za pomocą okna dialogowego (Rys.3.). Opcja dostępna tylko w przypadku wybrania opcji wytworzenia plików z zwierających strukturę „rozciągniętą". W oknie dialogowy wyświetlone zostaną tylko katalogi.



Rys. 3. Okna dialogowe wyboru ścieżki zapisu pliku zawierającego strukturę „rozciągniętą". [0]

h. Pole do wpisania z klawiatury ścieżki, pod którą znajduje się plik wejściowy.
i. Przycisk dodający ścieżkę z pola **h** do listy plików, na których zostaną wykonane operacje.
j. Przycisk otwierający okno dialogowe umożliwiające wybór wielu plików wejściowych. Wybór wielu plików wejściowych w jednym katalogu dokonuje się za pomocą zaznaczenia ich z jednoczesnym przytrzymaniem klawisza Shift.

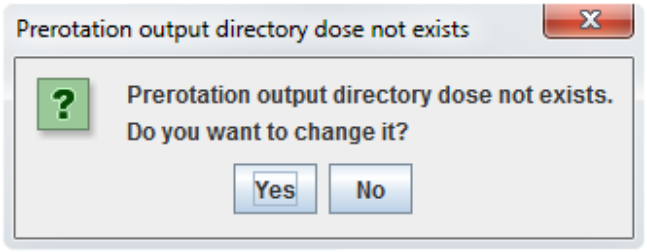


Rys. 4. Okno wyboru wielu plików wejściowych. [0]

k. Usuń zaznaczony plik z listy plików wejściowych.
l. Usuń wszystkie pliki z listy plików wejściowych.
m. Lista plików wejściowych.
n. Okno informujące o przebiegu pracy programu.
o. Przeprowadź obliczenia dla plików z listy plików wejściowych.
p. Wyczyść okno informujące o przebiegu pracy programu.
q. Zatrzymaj obliczenia.
r. Zamknij program.

W trakcie wykonywania obliczeń istnieje możliwość pojawienia się dodatkowych okien dialogowych w przypadku drobnych błędów zawartych w plikach wejściowych lub błędnie podanych parametrów pracy programu:

s. Informacja o nieistniejącej ścieżce zapisu plików wyjściowych lub ścieżce zapisu plików zawierających „rozciągniętą” strukturę białka:

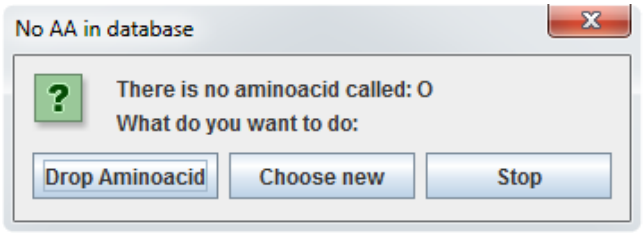


Rys. 5. Okno informacji o nieistniejącej ścieżce zapisu. [0]

Okno pojawia się w sytuacji podania nieistniejącej ścieżki zapisu lub pozostawienia pustego pola ścieżki (pola **d** lub **f**).

W przypadku wyboru opcji „*Yes”* zostanie otwarte okno wyboru ścieżki zapisu (Rys.2. lub Rys.3.). Wybranie opcji *„No”* lub zamknięcie okna spowoduje przerwanie obliczeń.

t. Informacja o nieistniejącym aminokwasie w bazie danych programu:



Rys. 6. Okno braku aminokwasu w bazie danych. [0]

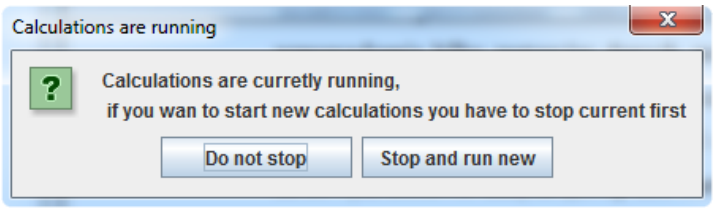
W trakcie obliczeń program może natrafić na aminokwas nienależący do bazy danych (nietypowy aminokwas) lub na błędnie wprowadzony aminokwas do pliku wejściowego. Użytkownik może ominąć dany aminokwas w dalszych obliczeniach, jeżeli aminokwas został wprowadzony przypadkowo (*„Drop aminoacid”*), zakończyć obliczenia prowadzone przez program (*„Stop”* lub zamknięcie okna) lub wybrać inny aminokwas w miejsce niestandardowego aminokwasu lub błędnie wprowadzonego (*„Choose new”*).

u. Po wybraniu opcji *„Choose new”* pojawia się okno wyboru nowego aminokwasu:

Rys. 7. Okno wyboru nowego aminokwasu. [0]

Z jego pomocą użytkownik może wybrać, który aminokwas ma zostać wstawiony lub zakończyć obliczenia zamykają powyższe okno.

v. W sytuacji, jeżeli dojdzie do próby równoległego uruchomienia obliczeń dla drugiego zestawy danych (przycisk 2.2.1.o.), program wyświetli okno informujące o zaistniałej sytuacji. Ponieważ nie przewiduje on możliwości prowadzenia obliczeń równocześnie w przypadku wprowadzenia kilku zestawów danych, przed uruchomieniem obliczeń na nowym zestawie danych należy poczekać aż program zakończy obliczenia („Do not stop”) albo przerwać obliczenia i rozpocząć prace na nowym zestawie danych („Stop and start new”).



Rys. 8. Okno ostrzegające o próbie uruchomienia drugich obliczeń. [0]

2.2.2. Rodzaje plików i opis wejściowych

Program w trakcie jednych obliczeń może pracować tylko na jednym formacie plików wejściowych. W celu wykreowanie plików wyjściowych z różnych formatów plików wejściowych, po skończonych obliczeniach dla jednego typu danych należy ponownie uruchomić obliczenia dla kolejnego typu.

Program przewiduje cztery formaty tekstowych plików wejściowych:

a. Plik zawierający trzyliterowe skróty nazw aminokwasów
b. Plik zawierający jednoliterowe skróty nazw aminokwasów
c. Plik w formacie AAD
d. Plik w formacie AADX

Poniżej zamieszczono opis przygotowania poszczególnych formatów plików oraz ich zawartości. Żaden z formatów nie uwzględnia wielkości znaków zawartych w pliku.

a. Pliki zawierające trzyliterowe skróty nazw aminokwasów:
   Plik w powyższym formacie powinien zawierać tylko i wyłącznie skróty trzyliterowe nazw aminokwasów zawartych w danym białku oddzielone białymi znakami.

   Każda inna dana spowoduje pojawienia się informacji o nieistniejącym aminokwasie w bazie danych programu (okno 2.2.1.t.).
   Poniżej zamieszczono przykład pliku wejściowego w opisywanym formacie:

   Plik.1. „example_tri.txt” [0]

```
ala cys asp glu phe gly his ile lys
leu met asn pro gln arg ser thr val
trp tyr
```

   Pliki wynikowe tego formatu zawierają wyłącznie struktury „rozciągnięte” białek.

b. Pliki zawierające jednoliterowe skróty nazw aminokwasów:
   Plik w powyższym formacie powinien zawierać tylko i wyłącznie skróty jednoliterowe nazw aminokwasów zawartych w danym białku nieoddzielone żadnymi znakami.

   Każdy biały znak powoduje zakończenie sczytywania sekwencji.

Każdy znak nierozpoznany, jako litera skrótu nazwy aminokwasu spowoduje pojawienia się informacji o nieistniejącym aminokwasie w bazie danych programu (okno 2.2.1.t.).

Poniżej zamieszczono przykład pliku wejściowego w opisywanym formacie:

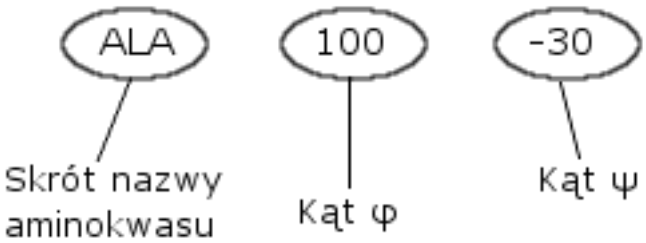
Plik.2. „example_one.txt” [0]

```
acdefghiklmnpqrstvwy
```

Pliki wynikowe tego formatu zawierają wyłącznie struktury „rozciągnięte” białek.

c. Pliki w formacie AAD

Plik w powyższym formacie powinien zawierać w jednej linii informacje o jednym, kolejnym aminokwasie: nazwę aminokwasu (w postaci skrótu jedno- lub trzyliterowego) oraz kąty φ i ψ dla danego aminokwasu.



Ryc. 3. Przykładowy opis jednego aminokwasu w pliku formatu AAD. [0]

Każda kolejna dana zawarta w linii nie będzie brana pod uwagę w trakcie obliczeń.

W przypadku, jeżeli na pierwszym miejscu w linii będzie dana nierozpoznawalna, jako skrót aminokwasu, pojawi się informacja o nieistniejącym aminokwasie w bazie danych programu (okno 2.2.1.t.).

W przypadku, jeżeli na drugim lub trzecim miejscu pojawią się dane niebędące liczbami program wyświetli informacje o niepoprawnym formacie pliku wejściowego i pominie dany plik w obliczeniach.

Poniżej zamieszczono fragment pliku będący przykładem pliku wejściowego w formacie AAD:

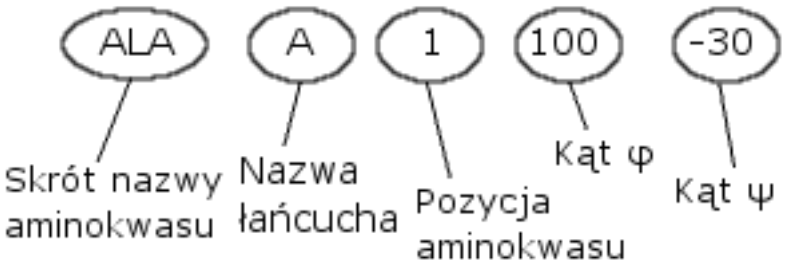
Plik.3. „example.aad” [0]

```
ala  0    -75
cys  15   -60
asp  30   -45
glu  45   -30
(…)
```

Pliki wynikowe tego formatu zawierają struktury białek wytworzone na według podanych danych. Mogą zostać dodatkowo wygenerowane struktury „rozciągnięte” białek.

d. Pliki w formacie AADX

Plik w powyższym formacie powinien zawierać w jednej linii informacje o jednym, kolejnym aminokwasie: nazwę aminokwasu (w postaci skrótu jedno- lub trzyliterowego), nazwie łańcucha białkowego, do którego aminokwas należy, pozycja aminokwasy w łańcuchu białka oraz kąty φ i ψ dla danego aminokwasu.



Ryc. 4. Przykładowy opis jednego aminokwasu w pliku formatu AADX. [0]

Każda kolejna dana zawarta w linii nie będzie brana pod uwagę w trakcie obliczeń.

W przypadku, jeżeli na pierwszym miejscu w linii będzie dana nierozpoznawalna, jako skrót aminokwasu, pojawi się informacja o nieistniejącym aminokwasie w bazie danych programu (okno 2.2.1.t.).

W przypadku, jeżeli na trzecim miejscu pojawi się dana niebędąca liczbą całkowitą lub czwartym albo piątym miejscu pojawią się dane niebędące liczbami, program wyświetli informacje o niepoprawnym formacie pliku wejściowego i pominie dany plik w obliczeniach.

Poniżej zamieszczono fragment pliku będący przykładem pliku wejściowego w formacie AADX:

Plik.4. „example.aadx" [0]

```
ala  A    1    0    -75
cys  A    2    15   -60
asp  A    3    30   -45
glu  A    4    45   -30
(…)
```

Pliki wynikowe tego formatu zawierają struktury białek wytworzone na według podanych danych. Mogą zostać dodatkowo wygenerowane struktury „rozciągnięte" białek.

Program nie rozróżnia formatu plików po ich rozszerzeniu.

2.2.3. Opis plików wyjściowych

Program na bazie pliku wejściowego może generować jeden lub dwa (w przypadku plików wejściowych w formatach AAD i AADX) pliki wyjściowe w formacie PDB lub w formacie PDB z dodatkowo zawartymi

współrzędnymi atomów wodoru, w zależności od opcji wybranej w polu 2.2.1.b.

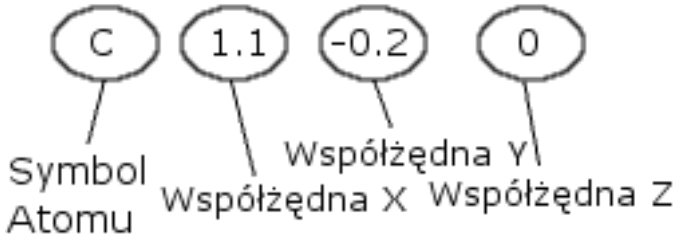
Dla każdego z formatów plików wejściowych jest generowany plik wyjściowy w katalogu o ścieżce podanej w polu 2.2.1.d. Wygenerowany plik będzie posiadać nazwę w formacie „nazwa_pliku_wejściowego.pdb”.

W przypadku plików w formacie AAD i AADX w przypadku wybrania opcji generacji struktury „rozciągniętej” (pole 2.2.1.c.), program wygeneruje plik zawierający tę strukturę w katalogu o ścieżce podanej w polu 2.2.1.f. Wygenerowany plik będzie posiadać nazwę w formacie „nazwa_pliku_wejściowego_p.pdb”.

2.2.4. Modyfikacja wewnętrznej bazy danych aminokwasów programu

Struktura programu daje możliwość wprowadzenia zmian w bazie danych aminokwasów programu. Taka właściwość jest przewidziana, na wypadek sytuacji, gdyby użytkownik programu chciał wygenerować inne izomery konformacyjne niż domyślnie przyjmuje program.

Każdy z plików zawierający informacje o atomach w aminokwasie jest edytowalnym plikiem tekstowym. Dane w każdej linii odpowiadają schematowi przedstawionemu na Ryc.5.:



Ryc. 5. Przykładowy opis jednego atomu w pliku bazy danych. [0]

Wszystkie współrzędne wyrażono w układzie kartezjańskim, w jednostkach Angstremach.

Poniżej przedstawiono przykładowy plik zawierający informacje o atomach wchodzących w skład glicyny:

Plik.5. „gly.txt”

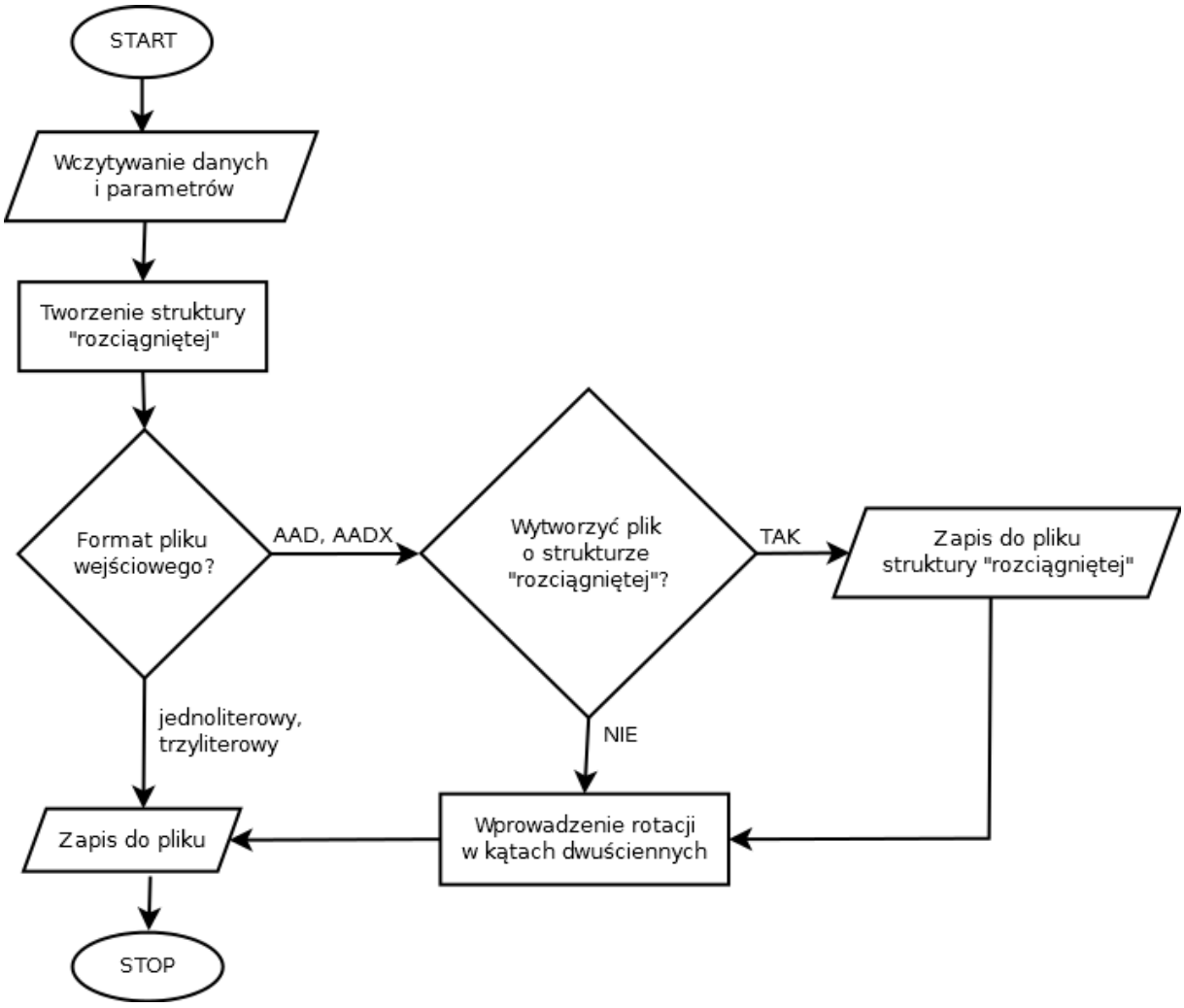
```
N   0.0       0.0       0.0
HN  -0.4226   0.9063    0.0
CA  1.4530    0.0       0.0
HA  1.8202    -0.5343   0.8762
C   2.0013    1.4284    0.0
O   1.2356    2.3910    0.0
HA  1.8202    -0.5343   -0.8762
    3.3231    1.5208    0.0
```

Ostatnia linia w pliku określa współrzędne kartezjańskie wstawienia atomu azotu kolejnego aminokwasu.

2.2.5. Opis operacji wykonywanych przez program w trakcie obliczeń.

Po ustawieniu odpowiednich parametrów i uruchomieniu obliczeń (przycisk 2.2.1.o.) program rozpocznie pracę nad przekazanymi danymi.

Pracę programu na tym etapie można przedstawić za pomocą schematu blokowego:



Schemat.1. Schemat blokowy obliczeń prowadzonych przez program.

Na samym początku zostanie sprawdzone czy w momencie uruchomienia obliczeń nie są wykonywane inne, wcześniej uruchomione. W sytuacji zaistnienia takiej sytuacji pojawi się okno 2.2.1.v.

Koleją operacją wykonaną przez program będzie inicjacja nowego obiektu klasy *MultiInput*, a następnie jej uruchomienie, jako wątek (metoda *run()*).

```
Klasa MultiInput
Jest to klasa dziedzicząca po klasie
Thread.
Konstruktor MultiInput (String[] files,
String prerotPath, String savePath, int
```

InputType, int h_atoms) - przyjmuje, jako parametry listę plików, na których mają być wykonane operacje, ścieżkę zapisu plików ze strukturą "rozciągniętą" (jeśli została wybrana taka opcja, w przeciwnym wypadku przyjmuje wartość "null")(pole 2.2.1.c.), ścieżkę zapisu plików wyjściowych, format plików wejściowych (pola 2.2.1.a.) oraz wybór czy w pliki wyjściowe mają zawierać atomy wodoru (0 - zawiera, 1 - nie zawiera) (pole 2.2.1.b.). Deklaruje i inicjuje odpowiednie zmienne deklaruje i inicjuje nową zmienną klasy Data.

Klasa Data
Jest to klasa dziedzicząca po klasie ArrayList<DataLine>.
Obiekty tej klasy służą do przechowywania informacji o atomach w białku.
Metody:
void printToFile (String savePath, int h_atoms) - metoda zapisująca zawartość obiektu klasy Data do pliku o ścieżce savePath. H_atoms informuje czy plik ma zawierać atomy wodoru.
void center() - przekształca współrzędne atomów w taki sposób by środek białka znajdował się w pobliżu początku układu współrzędnych.

Klasa DataLine
Konstruktor DataLine (String atom_name, String aa_name, String strand_letter,

```
int aa_i, double x, double y, double z)
- przyjmuje za parametry nazwę atomu,
nazwę aminokwasu, do którego atom nale-
ży, nazwę łańcucha, do którego amino-
kwas należy, numer aminokwasu, oraz
współrzędne atomu. Deklaruje i inicjuje
odpowiednie zmienne.
Klasa służy do przechowywania informa-
cji o atomie wchodzącym w skład badane-
go białka.
```

Uruchomiony wątek na samym początku sprawdza poprawność podanej ścieżki do katalogu zapisu plików wyjściowych oraz w przypadku wybrania opcji generacji dodatkowej struktury „rozciągniętej" (pole 2.2.1.c.) poprawność ścieżki do katalogu zapisu plików zawierających dodatkowe struktury. W przypadku, jeżeli zostanie wykryty błąd w podanej ścieżce nastąpi pojawienie się okna 2.2.1.s.

Następnie dla każdego przekazanego pliku program wykonuje poniższe operacje. W przypadku, gdy nie przekazano żadnego pliku w polu 2.2.1.n. zostanie wyświetlony odpowiedni komunikat.

Najpierw program sprawdza format przekazanego pliku. Ponieważ operacje wykonywane na plikach różnych formatów mają podobnego typu wyniki, ale inną metodykę, dalsza praca programu zostanie opisana w zależności od formatu pliku wejściowego.

I. Formaty skrótów jedno- i trzyliterowych:
Z pliku wejściowego następuje sczytanie jego zawartości (do zmiennej typu *String* w przypadku formatu jednoliterowego, do tablicy *Stringów* w przypadku skrótów trzyliterowych).

Następnie na bazie powyższej zmiennych generowane są kolejne pozycje w obiekcie klasy *Data*. Polega to na rozpoznaniu odpowiedniego amino-

kwasu (w przypadku nie rozpoznania pojawia się okno 2.2.1.t.), a następnie odwołaniu się do odpowiedniego pliku w bazie danych programu. Każda linia z pliku bazy danych jest wykorzystywana do utworzenia osobnej pozycji w obiekcie typu *Data*. Brakujące parametry wymagane przez konstruktor klasy *Data* są dodawane domyślnie; nazwa łańcucha - "A", numer aminokwasu - kolejna liczba naturalna przypisując pierwszemu aminokwasowi liczbę 1.

Następnie wszystkie atomy zapisane w obiekcie klasy *Data* są transformowane o wektor przeciwny do wektora utworzone na bazie ostatniej linii pliku z bazy danych, do którego odnosił się ostatni aminokwas, celem wczytania następnego pierwszego atomu azotu następnego aminokwasu w pozycji (0,0,0) w układzie współrzędnych, a za razem zachowania ciągłości białka. Po tej operacji następuje rotowanie wszystkich atomów względem osi **z** celem wytworzenia kąta 120^o pomiędzy węglem C ostatniego dodanego aminokwasu oraz atomem CA następnego aminokwasu względem punktu (0,0,0). Dodatkowo dla proliny, której ułożenie przestrzenne nie wymusza kąta torsyjnego CA-C-N_{+1}-CA_{+1+} równego 180^o lub 0^o w strukturze "rozciągniętej" następuje rotacja względem osi **x** celem osiągnięcia tego kąta. Współrzędne **y** oraz **z** każdego parzystego aminokwasu po wczytaniu są dodatkowo mnożone prze -1 celem osiągnięcia kąta torsyjnego CA_{-1}-C_{-1}-N-CA o wartości 180^o. Następnie następuje wczytanie kolejnego aminokwasu sposobem opisanym powyżej. Operacje transformacji oraz rotacji zostaną omówione w dalszej części tego podrozdziału.

Po wczytaniu wszystkich danych następuje transformacja współrzędnych atomów białka w taki sposób by jego środek znalazł się w pobliżu początku układu współrzędnych za pomocą metody *Data.center()*.

Kolejną operacją jest zapisanie zawartości obiektu klasy *Data* do pliku za pomocą metody *Data.printToFile (String savePath, int h_atoms)*, Gdzie *savePatch* oznacza ścieżkę zapisu pliku. W przypadku niezaznaczenia opcji by atomu wodoru były zawarte w pliku wyjściowym (pole 2.2.1.b.) są

one pomijane. Numery atomów w pliku są kolejnymi liczbami naturalnymi zaczynając od 1 w pierwszej linii.

Po wykonaniu powyższych program rozpoczyna obliczenia dla kolejnego pliku wejściowego.

Operacje transformacji i rotacji

Operacje są wykonywane z wykorzystaniem elementarnych macierzy transformacji:

$$Tran(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macirz.1. Elementarna macierz translacji. (a, b, c - przesunięcie wzdłuż osi X, Y oraz Z).

$$RotX(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macirz.2. Elementarna macierz rotacji względem osi X. (α – kąt rotacji).

$$RotY(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macirz.3. Elementarna macierz rotacji względem osi Y. (β – kąt rotacji).

$$RotZ(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macirz.4. Elementarna macierz rotacji względem osi Z. (γ – kąt rotacji).

Wyznaczanie nowego położenia punktów następuje za pomocą równania (1).

(1) $v' = A * v$

Gdzie:

A – macierz transformacji

v' – wektor tworzony na bazie wektora łączącego początek układu współrzędnych z nowym położeniem punktu

v – wektor tworzony na bazie wektora łączącego początek układu współrzędnych położeniem punktu przenoszonego

Aby umożliwić wykonanie mnożenia macierzowego do wektora powstałego na bazie dodaje się dodatkową pozycję:

[x , y , z] => [x , y , z , 1]

Wektor.1. Sposób kreowania wektora przeznaczonego do operacji transformacji.

Ze względu na właściwości mnożenia macierzowego metodę tę można wykorzystać do przyspieszenia transformacji punktów w przestrzeni:

(2) $v' = A * v$

(3) $v'' = B * v'$

(4) $v'' = B * A * v$

(5) $C = B * A$

(6) $v'' = C * v$

Gdzie:

A, B, C – macierze transformacji

v, v', v'' – wektory tworzone na bazie wektorów łączących początek układu współrzędnych z położeniami punktów

Dzięki tej właściwości obliczenia prowadzone dla aminokwasów będących w dalszej części białka niż aminokwas, nad którym pracuje program, mogą zostać odłożone w czasie oraz powoduje to zmniejszenie ilości operacji, skutkiem, czego jest przyspieszenie pracy programu.

II. Formaty AAD i AADX:

Podobnie jak w przypadku pozostałych dwóch formatów, w przypadku formatów AAD i AADX na początku następuje sczytanie danych z pliku wejściowego. Informacje te są zapisywane w obiekcie klasy *InputData.*

```
Klasa InputData
Jest to klasa dziedzicząca po klasie
ArrayList<InputDataLine>.
Obiekty tej klasy służą do przechowywa-
nia informacji pobranych z pliku wej-
ściowego.
Metody:
void fill (String input) - metoda sczy-
tująca dane z pliku o ścieżce input i
formacie AAD, a następnie dopisująca je
w obiekcie. Każda linia pliku wejścio-
wego odpowiada jednej pozycji w obiek-
cie. Brakujące dane wynikające z
formatu pliku są uzupełniane domyślnie;
nazwa łańcucha - "A", numer aminokwasu
- kolejna liczba rzeczywista, zaczyna-
jąc od 1 w przypadku pierwszego amino-
kwasu.
void fillX (String input) - metoda
sczytująca dane z pliku o ścieżce input
i formacie AADX, a następnie dopisująca
je w obiekcie. Każda linia pliku wej-
ściowego odpowiada jednej pozycji w
obiekcie.

Klasa InputDataLine
Konstruktor InputDataLine (String
String aa_name, String strand_letter,
int aa_i, double phi, double psi) - na-
```

```
zwę aminokwasu, do którego atom należy,
nazwę łańcucha, do którego aminokwas
należy, numer aminokwasu, oraz kąty
dwuścienne atomu. Deklaruje i inicjuje
odpowiednie zmienne.
Klasa służy do przechowywania aminokwa-
sie zadanym w pliku wejściowym w forma-
cie AAD i AADX.
```

Po ukończeniu wypełniania obiektu klasy *InputData* danymi następuje wypełnienie obiektu klasy *Data* informacjami o atomach w sposób podobny do opisanego wcześniej; program odwołuje się po kolei do obiektów klasy *InputDataLine* przechowywanych w obiekcie klasy *InputData*, pozyskując z nich informacje o nazwie aminokwasu, numeru aminokwasu oraz nazwie łańcucha.

W sytuacji, jeżeli zaznaczono opcję utworzenia pliku ze strukturą "rozciągniętą" następuje wycentrowanie białka metodą *Data.center()* a następnie zapis do pliku za pomocą metody *Data.printToFile (String savePath, int h_atoms)*, gdzie *savePath* oznacza ścieżkę zapisu pliku zawierającego strukturę "rozciągniętą". W przypadku niezaznaczenia opcji, aby atomu wodoru zostały zawarte w pliku wyjściowym (pole 2.2.1.b.) są one pomijane. Numery atomów w pliku są kolejnymi liczbami naturalnymi zaczynając od 1 w pierwszej linii.

Następnie następuje rotacja atomów w aminokwasie wedle zadanych kątów dwuściennych. Rotacja nie jest przeprowadzana dla pierwszego i ostatniego aminokwasu, gdyż brakuje w tej sytuacji punktów odniesienia.

Zostaje utworzona jednostkowa "główna macierz rotacji" (GMR) o wymiarach 4x4, wykorzystywana później w celu usprawnienia obliczeń.

Omówiony dalej przykład rotacji w kątach dwuściennych zostanie opisany dla dowolnego aminokwasu.

Na początku w obiekcie klasy *Data* zostaje wyznaczona pozycja atomu azotu N dla rotowanego aminokwasu.

Następnie za pomocą GMR następuje dołączenie atomów N i dwóch kolejnych (HN oraz CA) do wcześniej zrotowanej części białka, poprzez wymnożenie GMR z wektorem utworzonym na bazie współrzędnych atomów (w przypadku drugiego atomu następuje sytuacja mnożenia przez macierz jednostkową, więc położenie atomów się nie zmieni, natomiast w kolejnych obliczeniach jest wymagane wykonanie tej operacji, gdyż program nie przeprowadza obliczeń dla aminokwasów o liczbie wyższej niż ten, w którym obecnie wprowadza rotacje).

Następnie atomy N, HN oraz CA oraz wszystkie atomy je poprzedzające są transformowane tak, aby atom N znalazł się na początku układu współrzędnych. Każda operacja transformacji lub rotacji wymaga następującej po niej operacji aktualizacji GMR poprzez przypisanie do zmiennej przechowywującej macierz wynik mnożenia macierzy użytej do wykonania transformacji lub rotacji przez GMR.
Następnie przygotowana zostaje macierzy rotacji względem osi **z** a, po wykonaniu operacji rotacji za jej pomocą wszystkich atomów do atomu CA rotowanego aminokwasu, przygotowanie macierzy rotacji względem osi **y** i rotacji celem ustawienia atomu CA na osi **x**.

Po powyższych operacjach oś kąta φ aminokwasu leży na osi **x**, co ułatwia rotację do zadanej mu wartości. Ponieważ znany jest kąt początkowy (180^o dla wszystkich aminokwasów poza proliną) kąt, o jaki należy wprowadzić rotację jest równy wartości zadanej kąta minus 180^o. Na tej bazie po utworzeniu macierzy rotacji jest możliwa rotacja całego białka do atomu węgla CA celem wprowadzenia odpowiedniego kąta. W przypadku proliny φ rotacja w kącie psi nie jest wprowadzana, celem zachowania go równego -75^o.

W celu wprowadzenia rotacji o kat ψ, program za pomocą GMR dołącza pozostałe atomy rotowanego aminokwasu do badanego białka. Następnie w celu ustawienia osi kąta ψ wzdłuż osi **x,** transponuje białko tak, by węgiel CA znalazł się w początku układu współrzędnych, a później przemieszcza atom C by leżał na osi **x,** analogicznie jak w przypadku atomu CA dla kąta φ, z tym, że w tej sytuacji wykonuje rotacje na całym białku do aminokwasu, na którym są prowadzone obliczenia, a nie tylko do atomu CA.

Następnie w analogiczny jak powyżej przedstawiony sposób program przygotowuje macierz rotacji dla kąta ψ i wykonuje odpowiednią rotację współrzędnych atomu tlenu O.

Po wprowadzeniu rotacji w aminokwasie program przekazuje nową GMR do obliczeniach prowadzonych na kolejnym aminokwasie, którego atomy, za względu, że do tej pory nie były na nich prowadzone żadne obliczenia mają te same współrzędne, co atomy po sczytaniu danych z pliku wejściowego.

Po wykonaniu wszystkich przewidzianych operacji rotacji wedle zdefiniowanych kątów dwuściennych, następuje transformacja współrzędnych atomów kreowanego białka w taki sposób, by jego środek znalazł się w pobliżu początku układu współrzędnych za pomocą metody *Data.center()*.

Kolejną operacją jest zapisanie zawartości obiektu klasy *Data* do pliku za pomocą metody *Data.printToFile (String savePath, int h_atoms)*, Gdzie *savePatch* oznacza ścieżkę zapisu pliku. W przypadku niezaznaczenia opcji, aby atomy wodoru były zawarte w pliku wyjściowym (pole 2.2.1.b.) są one pomijane. Numery atomów w pliku są kolejnymi liczbami naturalnymi zaczynając od 1 w pierwszej linii.

Po wykonaniu powyższych program rozpoczyna obliczenia dla kolejnego pliku wejściowego.

Po skończeniu obliczeń dla ostatniego pliku uruchomiony wątek zostaje zakończony i jest możliwe ponowne uruchomienie obliczeń.

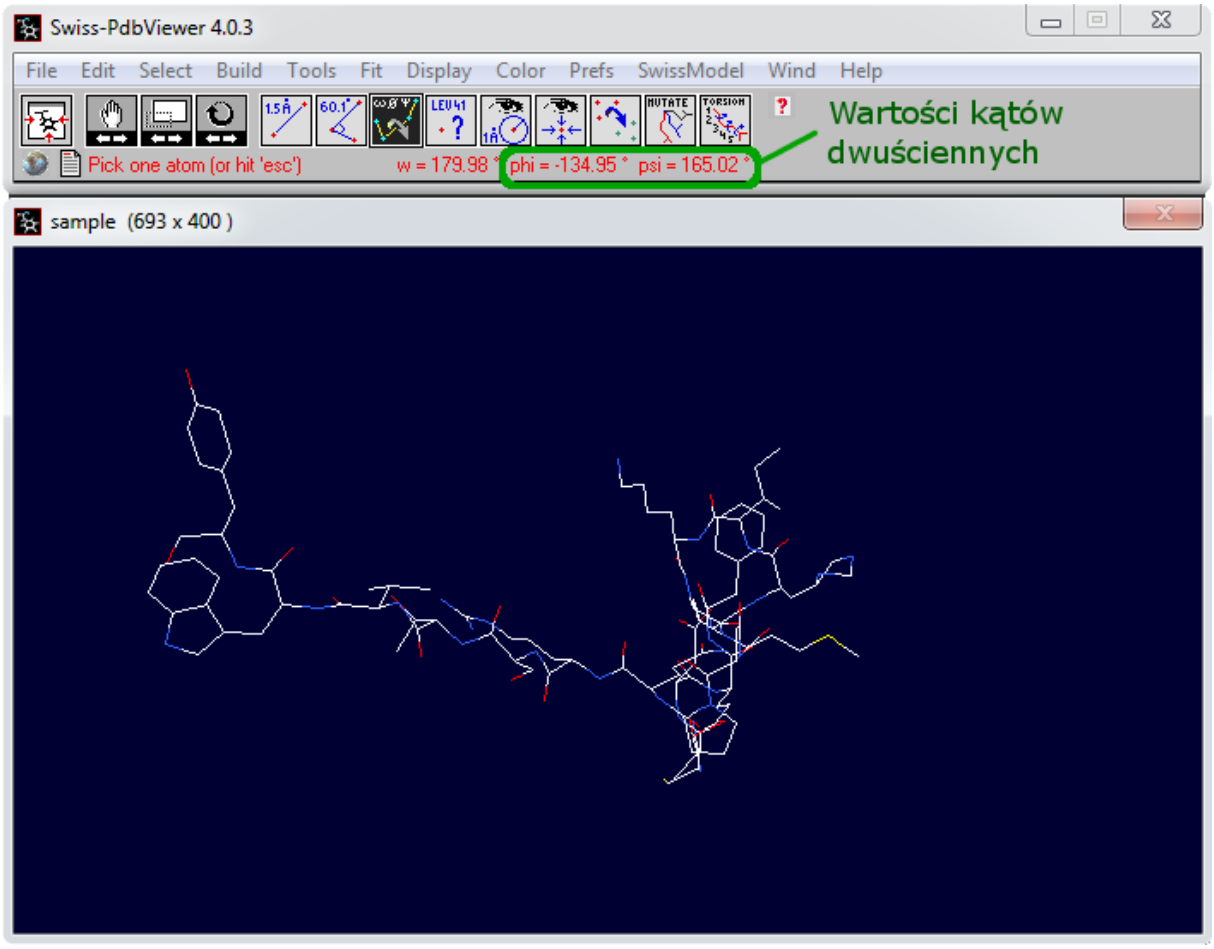
3. Wyniki i testy poprawności generowanych struktur

W celu sprawdzenia poprawności generowanych struktur przeprowadzono szereg testów:

a. Testy na plikach „example”

Pierwszymi testami przeprowadzonymi na programie były testy na plikach zawartych w katalogu *Example* znajdującym się w głównym katalogu programu.

W trakcie testów wykorzystano program Swiss - PdbViewer [7] do oceny poprawności kątów generowanych przez program.



Rys.9. Testy poprawności generowanych struktur. [7]

Na Rys.9. został przedstawiony plik wyjściowy dla pliku *example.aad*. Kąty mierzone na rycinie są zdefiniowane dla treoniny w pliku wejściowym.

Testy wyszły pozytywnie.

b. Testy z wykorzystaniem generatora liczb losowych

W kolejnym teście przygotowano trzy pliki wejściowe w formacie AAD, w których wygenerowano dziesięć losowych aminokwasów z losowym zestawem kątów dwuściennych.

Na bazie plików wygenerowano struktury białek porównano wartości kątów w nich zawartych z wartościami wygenerowanymi.

Testy wyszły pozytywnie.

c. Testy z wykorzystaniem plików z powtórzonymi aminokwasami

Na bazie plików „example" utworzono pliki, w których każdy aminokwas pojawiał się dwa razy, parami koło siebie.

Na bazie plików wygenerowano struktury białek porównano wartości kątów w nich zawartych z wartościami wygenerowanymi.

Testy wyszły pozytywnie.

d. Testy na wykrywanie błędów wprowadzania danych przez użytkownika

Testy polegały na przygotowaniu plików wejściowych z błędnymi danymi lub przekazywaniu do programu plików zawierających inny format danych niż został wybrany w polu 2.2.1.a., a także bez podania ścieżek zapisu plików wyjściowych.

W śród plików wejściowych znalazły się pliki z katalogu *Example,* na bazie, których starano się wykonać obliczenia wybierając inny format danych niż był w rzeczywistości.

Wykorzystano zmodyfikowane pliki *Example* ze zmienionymi na nieistniejące nazwami aminokwasów.

Na sam koniec próbowano uruchomić obliczenia z wykorzystaniem plików wyjściowych programu oraz dowolnych plików zawartych w komputerze.

We wszystkich przypadkach program informował o błędzie w sposób wcześniej przewidziany.

Testy wyszły pozytywnie.

e. Zewnętrzne testy poprawności

Na bazie dodatkowych plików wejściowych, otrzymanych od prof. dr hab. Ireny Roterman-Koniecznej, wygenerowano struktury białek, a następnie odesłano celem sprawdzenia poprawności plików wynikowych.

Testy wyszły pozytywnie.

Na podstawie powyższych testów stwierdzono,

że program generuje prawidłowe dane.

4. Bibliografia i źródła

[0] Źródło własne

[1] Jane Shelby Richardson, Licencja: Creative Commons Attribution 3.0 Unported, http://upload.wikimedia.org/wikipedia/commons/c/c0/Protein_backbone_PhiPsiOmega_drawing.jpg, 23.01.2012r.

[2] Adam Betts, Licencja: Free for personal use, http://files.softicons.com/download/application-icons/adobe-cs3-icons-by-adam-betts/png/512x512/Adobe_Photoshop_Lightroom_Data.png, 23.01.2012r.

[3] Wygenerowane za pomocą VMD 1.9

[4] Autor: artdesigner.lv, Licencja: Creative Commons Attribution 3.0 Unported, http://files.softicons.com/download/web-icons/webtoys-icons-by-artdesigner.lv/png/64x64/Computer.png, 23.01.2011r.

[5] PDB, Wizualizacja białka 1CBH, http://www.rcsb.org/pdb/images/1CBH_bio_r_250.jpg?bioNum=1, 23.01.2011r.

[6] Monamy FA, McGuire RF, Burgess AW, Scheraga HA. Energy parameters in polypeptides. VII. Geometric parameters, Partial Atomic charges, onbonded interaction, hydrogen bond interactions and intrinsic torsional potenmtials for the naturrally occurring amino acids. J. Phys Chem. 79, 1975, p. 2361- 2381.

[7] Swiss – PdbViewer v4.0.3

5. Informacje licencyjne programu

Program oraz kod źródłowy programu dostępne są na stronie www.zbaster.com na bazie licencji Creative Commons Attribution 3.0 Unported.