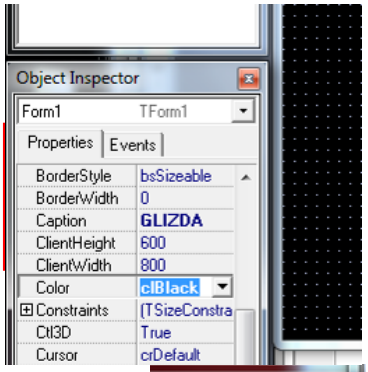
**GLIZDA**

FORMATKA

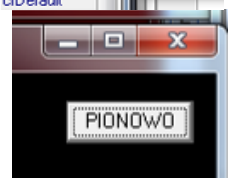



- Uruchom Delphi
- Wybierz z menu File / Save All
- Wybierz Pulpit i załóż tam folder GLIZDA-*Nazwisko Imię*
- Zapisz w tym folderze plik UNIT z nazwą GLIZDA_P.PAS
- Zapisz w tym folderze plik PROJECT z nazwą GLIZDA_D.DPR

*Nie zapomnij o „wejściu" do folderu, o rozszerzeniach pików i o częstym zapisywaniu pliku źródłowego i pliku projektu SHIFT+CTRL+S*

- Ustawiamy początkowe własności formatki w okienku **Object Inspector** na zakładce **Properties,** jak podano na obrazku

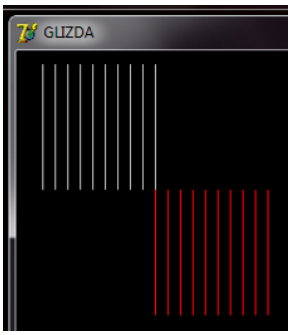
(1) Pokaż wynik działania programu - szara formatka




PIONOWO

- Umieść na formatce przycisk z prawej strony
- Nazwij przycisk „PIONOWO" - własność Caption
- Kliknij podwójnie w przycisk (tworzysz procedurę **Button1Click**)
- Za pomocą pętli FOR (lub innej) narysuj 10 pionowych odcinków (schemat obok)

*długość 100, odstęp10, początek 10,10, kolor clSilver*




- Zadeklaruj niezbędne zmienne

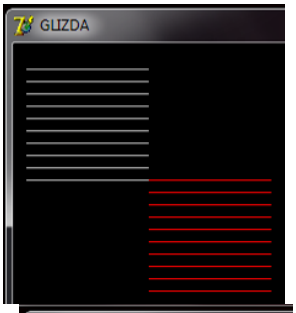
(2) Pokaż wynik działania programu - 10 pionowych szarych linii

PIONOWO-PROCEDURA

- Napisz procedurę o nazwie PIONOWO, która narysuje N pionowych odcinków o długości D, odstęp pomiędzy O, początek (X,Y), kolor K - nagłówek procedury: **procedure TForm1.PIONOWO(x,y,n,d,o,k:integer);**
- W przycisku „PIONOWE" wpisz instrukcję: **PIONOWO(110,110,10,100,10,clRed);**
- Zadeklaruj zmienną sterującą pętlą

(3) Pokaż wynik działania programu - 10 pionowych czerwonych linii

POZIOMO



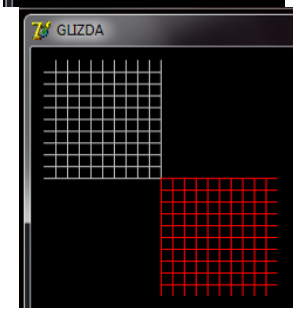

- Umieść na formatce przycisk z prawej strony (pod przyciskiem PIONOWO)
- Nazwij przycisk „POZIOMO" - własność Caption
- Kliknij podwójnie w przycisk (tworzysz procedurę **Button2Click**)
- Za pomocą pętli FOR (lub innej) narysuj 10 poziomych odcinków (schemat obok)

*długość 100, odstęp10, początek 10,10, kolor clSilver*

- Zadeklaruj niezbędne zmienne

(4) Pokaż wynik działania programu - 10 poziomych szarych linii

POZIOMO-PROCEDURA




- Napisz procedurę o nazwie POZIOMO, która narysuje N poziomych odcinków o długości D, odstęp pomiędzy O, początek (X,Y), kolor K - nagłówek procedury: **procedure TForm1.POZIOMO(x,y,n,d,o,k:integer);**
- W przycisku „PIONOWE" wpisz instrukcję: **POZIOMO(110,110,10,100,10,clRed);**
- Zadeklaruj zmienną sterującą pętlą

(5) Pokaż wynik działania programu - 10 poziomych czerwonych linii

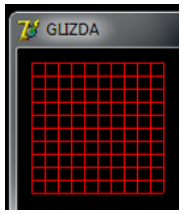
KRATKI

*Zauważ że na rysunku obok szare odcinki nie tworzą szachownicy. Potrzebne są jeszcze: jeden pionowy i jeden poziomy odcinek na początku. Jeżeli masz otrzymałeś podobny wynik popraw instrukcje w przyciskach - np. rozpocznij odliczanie zmiennej sterującej od zera*


- W procedurze **Button1Click** i **Button2Click** popraw instrukcje

(6) Pokaż wynik działania programu - szachownica z 11 szarych linii

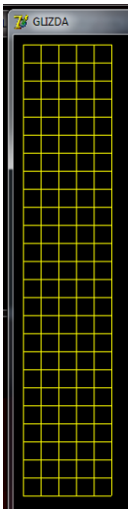
*Sprawdź, czy identyczny wynik uzyskasz rysując kratki za pomocą procedur PIONOWO i POZIOMO*




- W procedurze **Button1Click** i **Button2Click** popraw instrukcje PIONOWO i POZIOMO, aby zamalować szare odcinki z zadania +3

(7) Pokaż wynik działania programu - szachownica z 11 czerwonych linii

*Za pomocą procedur PIONOWO i POZIOMO narysujemy dowolny pokratkowany obszar. Niezbędne będą następujące parametry: lewy górny róg - X,Y, ilość kratek poziomo - KX i pionowo - KY, wielkość kratki - O i kolor odcinków - K*




- Napisz nową procedurę o nazwie **KRATKI** z parametrami: **x,y,kx,ky,o,k**
- W procedurze wykonaj następujące wyliczenia:
  - ilość odcinków pionowych według wzoru: **Ipio = ilość kratek poziomo + 1**
  - ilość odcinków poziomych według wzoru: **Ipoz = ilość kratek pionowo + 1**
  - długość linii pionowej według wzoru **Dpio = ilość kratek pionowo * odstęp**
  - długość linii poziomej według wzoru **Dpoz = ilość kratek poziomo * odstęp**
  - Wywołaj procedurę PIONOWO z parametrami **x,y,Ipio,Dpio,o,k**
  - Wywołaj procedurę POZIOMO z parametrami **x,y,Ipoz,Dpoz,o,k**
- Umieść na formatce przycisk i nazwij go **KRATKI**
- Do procedury **Button3Click** tego przycisku wpisz instrukcję: **KRATKI(10,10,5,25,20,clYellow);**

(8) Pokaż wynik działania programu - szachownica z żółtych linii

## SUWAKI

*Ilości kratek w pionie i poziomie oraz wielkość kratek będziemy ustawiać za pomocą suwaków. Aby były „widoczne" w całym programie (każda procedura mogła z nich skorzystać) zadeklarujemy zmienne globalne.*


- Zadeklaruj zmienne globalne: **KRx, KRy, Kro** typu integer
- Na formatce ustaw 3 **SUWAKI** i 3 **ETYKIETY** (obrazek)
- Ustaw biały kolor czcionki w etykietach (własność **FONT**)
- Do własności Caption etykiet wpisz początkowe wartości: np. „10"
- Początkowych ustawień suwaków dokonano w procedurze **FormActivate** formatki (patrz ramka)

```
KRx:=30;
KRy:=20;
KRo:=20;
Scrollbar1.Min:=1;
Scrollbar1.Max:=100;
Scrollbar1.Position:=30;
Label1.Caption:=IntToStr(KRx);
Scrollbar2.Min:=1;
Scrollbar2.Max:=100;
Scrollbar2.Position:=20;
Label2.Caption:=IntToStr(KRy);
Scrollbar3.Min:=1;
Scrollbar3.Max:=100;
Scrollbar3.Position:=20;
Label3.Caption:=IntToStr(KRo);
```

(9) Pokaż wynik działania programu - suwaki zostaną ustawione

*Zmieniamy położenia suwaków - szachownica powinna się za każdym razem przerysować. Stosownych zmian dokonamy w zdarzeniach OnChange (procedura ScrollBar_Change dla każdego suwaka)*

- Kliknij podwójnie w pierwszy suwak
- Wpisz instrukcję **Form1.Repaint** *(wymazanie ekranu)*
- Do zmiennej KRx wstaw położenie suwaka
- Wywołaj procedurę rysowania kratek z następującymi parametrami: **10,10,KRx,KRy,KRo,clSilver**
- Do etykiety wstaw pozycję suwaka
- W podobny sposób ustaw suwak 2 (Kry) i suwak 3 (KRo)

(10) Pokaż wynik działania programu - suwaki rysują kratki

## KROPKI

- Utwórz nową procedurę o nazwie **KROPKA** i parametrach **X, Y, S, K** typu całkowitego

*X, Y - położenie kropki, S-średnica kropki, K - kolor kropki*

- W procedurze KROPKA:
  - Ustaw szerokość linii za pomocą instrukcji: **Canvas.Pen.Width:=S;**
  - Ustaw kolor rysowanie na K
  - Ustaw kursor graficzny w punkcie X, Y
  - Narysuj linię do punktu X,Y *(linia o długości 1 piksela - powstanie kropka)*
- Utwórz przycisk o nazwie **KROPKI**
- W procedurze OnClick przycisku, za pomocą procedury KROPKI narysuj kropki, jak pokazano na rysunku


(11) Pokaż wynik działania programu - kropki w kratkach

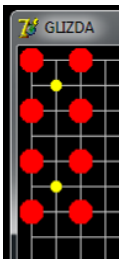
*Ponieważ zmieniliśmy szerokość rysowanych linii (Pen.Width), dlatego należy w procedurach, gdzie rysujemy linie (PIONOWO i POZIOMO) ustawić szerokość na 1*

- W procedurach PIONOWO i POZIOMO ustaw szerokość rysowanych linii na 1 piksel

(12) Pokaż wynik działania programu - zmień na suwakach rozmiar szachownicy

*Aby narysować kropkę musimy podawać współrzędne punktu na formatce. Wygodniej byłoby, gdybyśmy podawali współrzędne kratownicy. Na przykład (1,1) - lewy górny róg. (2,2) - położenie żółtej kropki z poprzedniego przykładu itd.*

- Napisz procedurę o nazwie **KKROPKI** i parametrach **Kx, Ky, S, K** typu integer




*procedura narysuje kropkę, ale podajemy współrzędne na kratownicy: Kx i Ky. Lewy górny róg (1,1)*

- W procedurze wylicz położenie piksela (środek punktu) na formatce
  - współrzędna X = od podanej ilości kratek w poziomie (KRx) odejmij jeden, wynik pomnóż przez szerokość kratek (Kro) i do wyniku dodaj 10 (początek kratownicy)
  - w podobny sposób wylicz współrzędną Y
  - wywołaj procedurę **KROPKI** z parametrami **X, Y, S, K**
- W procedurze **KROPKI**, za pomocą procedury **KKROPKI** narysuj kolejnych 5 kropek

(13) Pokaż wynik działania programu - kolorowe kropki

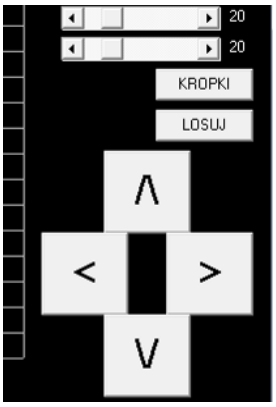
*Losowo, po całej kratownicy rysujemy kropki. Granice rysowania wyznaczają KRx+1 i KRy+1. Aby losowanie za każdym razem przebiegało inaczej należy zainicjować generator liczb „pseudolosowych"*

- Utwórz procedurę o nazwie **LOSUJ** z parametrem **ILE** typu integer
- W procedurze, za pomocą dowolnej pętli wykonaj następujące instrukcje **ILE** razy:
  - Wylosuj Kx (numer kratki w poziomie) w przedziale od 1 do Kx+1
  - Wylosuj Ky (numer kratki w pionie) w przedziale od 1 do Ky+1
  - Wykonaj procedurę KKROPKI z parametrami Kx, Ky, Kro, clRed
- Utwórz przycisk o nazwie LOSUJ
- W procedurze OnClick przycisku wpisz LOSUJ(10) - losowanie 10 jabłek
- W procedurze FormActivate dopisz instrukcję **Randomize** - inicjacja generatora

(14) Pokaż wynik działania programu - kilka razy wykonaj losowanie

## PRZYCISKI DO PRZESUWANIA GŁOWY

*Umieszczamy na ekranie kropkę (głowę naszej glizdy) w losowym miejscu i przesuwamy ją za pomocą czterech przycisków po jednej kratce w każdą stronę*




- Umieść na formatce 4 kwadratowe przyciski, jak pokazano na obrazku
- Opis przycisków złożony ze znaków „>", „<", „\", „/", czcionka 24

(15) Pokaż wynik działania programu - żółta kropka rysowana w różnych miejscach

*żółta kropka pojawia się w różnych miejscach na kratownicy*

- Zadeklaruj dwie zmienne **GLx** i **GLy** typu integer - położenie głowy glizdy
- W procedurze **FormActivate**
  - wylosuj wartości do tych dwóch zmiennych tak, aby mieściły się na kratownicy (1..KRx+1, 1..KRy+1) - podobnie jak w procedurze LOSUJ
  - za pomocą procedury **KKROPKI** narysuj żółtą kropkę o szerokości Kro w położeniu GLx i GLy

(16) Pokaż wynik działania programu - żółta kropka rysowana w różnych miejscach

*Klikamy w przycisk „W prawo" - żółta kropka przesuwa się w prawo. Poprzednie swoje miejsce zamazuje na czarno*

- Kliknij podwójnie w przycisk „>" (ruch w prawo). W procedurze:
- Narysuj czarną kropkę w położeniu GLx, GLy
- Zwiększ GLx o 1 (ruch w prawo)
- Narysuj żółtą kropkę w położeniu GLx, GLy

(17) Pokaż wynik działania programu - żółta kropka przesuwa się w prawo

*Żółta kropka przesuwa się we wszystkich kierunkach*

- W podobny sposób zaprogramuj pozostałe 3 klawisze

(18) Pokaż wynik działania programu - żółta kropka przesuwa się w dowolnym kierunku

*Żółta kropka nie wymazuje na czarno poprzedniego swojego położenia*

- Dopisz instrukcję rysowania kratownicy przed instrukcją rysowania głowy glizdy w nowym położeniu w czterech przyciskach

(19) Pokaż wynik działania programu - żółta kropka nie wymazuje kratownicy

- Jeżeli kropka wychodzi poza kratownicę, to pojawia się po przeciwnej stronie. Dopisz instrukcje warunkowe w czterech przyciskach, po instrukcji obliczenia nowego położenia

*Dla ruchu w prawo lub w dół: jeżeli współrzędna głowy glizdy jest większa od ilości kratek +1, to współrzędną głowy jest równa 1, Dla ruchu w lewo lub w górę: jeżeli współrzędna głowy glizdy jest mniejsza od 1, to współrzędna głowa glizdy jest równa ilość kratek+1*

(20) Pokaż wynik działania programu - żółta kropka pojawia się po przeciwnej stronie

*Cztery razy powtarzamy te same instrukcje podczas przesuwania głowy glizdy (a być może za chwilę będzie ich więcej). Praktyczniej będzie, jeśli wszystkie operacje związane z przerysowywaniem planszy wstawimy do jednej procedury*

- Utwórz nową procedurę o nazwie **PRZERYSUJ** bez parametrów.
- W procedurze wpisz
  - Form1.Repaint - wyczyszczenie formatki
  - rysowanie kratek
  - rysowanie głowy glizdy
- Umieścić nową procedurę w suwakach i przyciskach sterujących
- Usuń z tych przycisków niepotrzebne instrukcje
- Usuń rysowanie kropki z procedury LOSUJ

(21) Pokaż wynik działania programu - po zmianie wymiarów kratownicy widać kropkę

## ANIMACJA

*Co zrobić, żeby głowa glizdy poruszała się sama, a przyciski służyły jedynie do zmiany kierunku? Prosty sposób został pokazany na lekcjach (zwyczajna pętla). Profesjonalnie wykonamy to za pomocą TIMERa. Raz na konkretnie określony czas (np. co 1 sekundę) wykonywać będziemy instrukcje.*

- Umieszczamy na formatce obiekt **TIMER** (zakładka System), w dowolnym miejscu
- Kliknij podwójnie w obiekt i wpisz instrukcje: **GLx:=GLx+dx;** oraz **GLy:=GLy+dy;**

*Własność Interval Timera ustawiona jest na 1000, to znaczy, że co 1 sekundę będą wyliczane nowe współrzędne głowy glizdy na podstawie kierunków dx i dy*

- Zadeklaruj zmienne globalne DX i DY w globalnych
- W FormActivate ustaw DX i DY na zero *(na początku głowa się nie rusza)*

*W przyciskach należy przedefiniować sposób zmiany kierunku. Od tej potu już nie kliknięcie w przycisk będzie wyliczać nowe położenie, a jedynie zmieniać kierunek (DX lub DY). Warunki brzegowe, też będzie można wstawić w jedno miejsce - do Timera.*

- Warunki brzegowe z przycisków przestaw do procedury **Timer1Timer**
- Instrukcję PRZERYSUJ przestaw do **Timer1Timer**
- Zamiast instrukcji GLx:=GLx+1 wpisz DX:=1; DY:=0 (ruch w prawo) i w podobny sposób przedefiniuj pozostałe 3 przyciski
- Własność **Interval** w Timer1 ustaw na 200

(22) Pokaż wynik działania programu - kliknij w przycisk ruchu - głowa porusza się sama

*Kolejny suwak posłuży do zmiany szybkości poruszania się głowy glizdy*

```
ScrollBar4.Min:=0;
ScrollBar4.Max:=1000;
ScrollBar4.Position:=200;
ScrollBar4.LargeChange:=100;
```

- Na formatce umieść **SUWAK** i **ETYKIETĘ**
- Ustaw biały kolor czcionki w etykietach (własność **FONT**)
- Do własności **CAPTION** etykiety wpisz „200”
- Własności suwaka w ramce - wstaw je do procedury **FormActivate**
- Kliknij podwójnie w suwak i wpisz następujące instrukcje:
  - Do etykiety wstaw własność **POSITION** suwaka (zamień ją na tekst)
  - Do własności **INTERVAL** zegara wstaw pozycję suwaka

(23) Pokaż wynik działania programu - suwakiem możemy zmieniać szybkość ruchu

## KLAWIATURA

*Dużo wygodniejszym sposobem zmiany kierunku ruchu glizdy jest klikanie w klawisze kursorów i skorzystanie z własności OnKeyDown Formatki, która podaje kod wciskanego klawisza. Pojawia się jednak problem - kursory służą również do zmiany tzw. Focusa, czyli wyboru aktywnego elementu formatki. „Oduczenie” formatki tej własności kursorów jest dość skomplikowane, dlatego kierunek ruchu głowy glizdy będziemy zmieniać za pomocą zwykłych klawiszy.*

- Odszukaj właściwość **ONKEYDOWN** formatki i kliknij podwójnie - pojawi się procedura

*Zmiana kierunku może być wykonana przez ponowne wpisanie instrukcji DX:=... i DY:=... lub przez wywołanie procedury kliknięcia w ten przycisk, np. ruch w prawo: **Button_Click(Sender)**. W miejsce znaku „_” trzeba wstawić odpowiedni numer przycisku.*

- Jeżeli wciśnięto klawisz o kodzie 76 (klawisz L) zmień kierunek na „w prawo”
- W podobny sposób dla kodu 74 (J) - w lewo, 73 (I) - w górę, 75 (K) - w dół

(24) Pokaż wynik działania programu - klawisze IJKL służą do zmiany kierunku

- Klawisz **ESC** o kodzie 27 kończy działanie programu - instrukcja **Application.Terminate;**
- Klawisz **P** o kodzie 80 zatrzymuje lub wznawia ruch glizdy - własność Enabled Timera1 staje się nieprawda że Timer1.Enabled (zastosuj funkcję NOT)

(25) Pokaż wynik działania programu - klawisze ESC - koniec i P - pauza

## TABLICA

*Nasza glizda ma „zjadać" jabłka - będziemy zliczać punkty, za każde zjedzone jabłko. Umiemy losować jabłka lecz ich położenie nie jest nigdzie zapisane. Najwygodniejszym sposobem jest zastosowanie tablicy dwuwymiarowej, tak dużej, ile może być kratek w poziomie i pionie. W każdej komórce tablicy będą zera - brak jabłek lub 1 - jabłko po wylosowaniu. Tablica musi być wyzerowana na początku. Wylosowane położenie jabłek zapiszemy w tablicy i za każdym razem będziemy przerysowywać jabłka w tablicy*

- Zadeklaruj tablicę o nazwie JAB, dwuwymiarową o komórkach typu całkowitego, ponumerowanych w obie strony od 1 do 100, jako zmienną globalną
- W procedurze LOSUJ, po narysowaniu jabłka wpisz instrukcję: **JAB[Kx,Ky]:=1;**
- Utwórz nową procedurę o nazwie **ZERUJ_JAB** bez parametrów, w której do wszystkich komórek tablicy JAB wpiszesz zera (podwójna pętla)
- Umieść wywołanie tej procedury w FormActivate
- Utwórz nową procedurę o nazwie **RYSUJ_JAB** bez parametrów, która narysuje wszystkie jabłka znajdujące się w tablicy.
- W procedurze (najlepiej za pomocą podwójnej pętli) wpisz warunek logiczny: jeżeli komórka tablicy JAB jest równa 1 to rysuj czerwoną kropkę (położenie podają zmienne pętli) o wielkości Kro
- Umieść wywołanie procedury RYSUJ_JAB w procedurze PRZERYSUJ (przed rysowaniem głowy glizdy)

(26) Pokaż wynik działania programu - wylosuj jabłka i puść glizdę w ruch - widoczne jabłka i nie wymazuje

- Umieść na formatce przycisk, napis START
- Wstaw do niego instrukcję losowania jabłek i instrukcję ruchu w prawo
- Klawisz S (83) działa, jak wciśnięcie przycisku START

(27) Pokaż wynik działania programu - wciśnij START lub klawisz S - losowanie i ruch

## PUNKTYACJA

*Jeżeli głowa glizdy wejdzie na jabłko, to zwiększamy ilość punktów. Sprawdzamy zawsze po wyliczeniu nowego położenia głowy, przed przerysowaniem.*

- Zadeklaruj zmienną globalna **PUNKTY** typu integer
- W procedurze **Timer1Timer** *(po wyliczeniu nowych współrzędnych, przed rysowaniem)*
  - jeżeli komórka tablicy JAB o współrzędnych (GLx,GLy) jest równa 1 (znajduje się na jabłku) to
    - zerowanie tej komórki tablicy (nie ma jabłka)
    - zwiększ punkty o 1
    - ustaw etykietę **Label6.Caption:='PUNKTY:'+IntToStr(PUNKTY);** *(sprawdź numer etykiety)*
- Wstaw na formatkę etykietę, napis „PUNKTY: 0"
- Wyzeruj PUNKTY w FormActivate

(28) Pokaż wynik działania programu - glizda „zjada" jabłka i zwiększa punkty

## JABŁKA

*Jabłko czerwone jest dobre i smaczne (za jeden punkt). Dodatkowo na tablicę wylosujemy jeszcze jabłka brązowe - zgnite (-2 punkty) i zielone - niedojrzałe (-1 punkt), czyli te, które należy omijać. I w podobny sposób jak poprzednio, będziemy sprawdzać, czy głowa glizdy zjada to jabłko i odejmować punkty*

- Popraw procedurę **LOSUJ**
  - W nagłówku trzy parametry: C, B, Z oznaczające ilości losowanych jabłek każdego koloru *(będziesz musiał poprawić nagłówek w górnej części i dwa wywołania procedury LOSUJ zastosowane do tej pory)*
  - W procedurze jeszcze dwie pętle losujące jabłka za -2 i -1
- Popraw instrukcje losowania jabłek w programie: zastosuj wszędzie **LOSUJ(10,10,10)**
- Popraw procedurę **RYSUJ_JAB**
  - Dwa nowe warunki logiczne rysujące jabłka *(będziesz musiał zastosować instrukcję złożoną: Begin…end Kolor brązowy - clMaroon, kolor zielony - clLime)*
- Popraw procedurę **Timer1Timer**
  - Wpisz jeszcze dwa podobne warunki logiczne sprawdzające jabłka

  *lub od razu obsłużyć wszystkie rodzaje jabłek*
  - do zmiennej J wstaw zawartość komórki tablicy JAB o współrzędnych GLx i GLy

- Warunek logiczny: jeżeli J różna od zera to
  - Wyzeruj komórkę tablicy
  - Zwiększ PUNKTY o J
  - Wstaw punkty do etykiety

(29) Pokaż wynik działania programu - różne jabłka na ekranie „zjada" glizda

## OGON

*Po zjedzeniu dobrego jabłka rośnie gliździe ogon o jeden człon. Po zjedzeniu złego jabłka - ogon się skraca. Położenie poszczególnych elementów ogona trzeba gdzieś zapamiętać - oczywiście najlepszym sposobem jest tablica, w której zapiszemy współrzędne kropek ogona. Podczas przerysowywania uwzględnimy ogon glizdy*

- Zadeklaruj tablicę, jako zmienną globalną **OGON: array[1..2,1..100] of integer;**

*w wierszu numer 1 zapisane współrzędne X, a w wierszu 2 - współrzędne Y kropek ogona. Na wszelki wypadek w tablicy będzie można zapamiętać położenie kropek ogona o długości 100. Zmienna PUNKTY pamiętać długość ogona - ile ma kropek*

- Utwórz procedurę **ZERUJ_OGON** bez parametrów, która wpisze do komórek tablicy OGON zera
- Wstaw do procedurę instrukcję ZERUJ_OGON do FormActivate
- Utwórz procedurę **RYSUJ_OGON** bez parametrów, która rysuje ogon w następujący sposób
- W pętli **FOR** od 1 do PUNKTY wykonujemy instrukcje
  - wstaw do zmiennej X zawartość komórki **OGON[1,i]** *(„i" jest zmienną sterującą pętlą)*
  - wstaw do zmiennej Y zawartość komórki **OGON[2,i]**
  - rysuj kropkę (**KRKOPKA**) ze współrzędnymi X i Y, wielkość kropki Kro, kolor clWhite
- Wstawić procedurę RYSUJ_OGON do PRZERYSUJ *(po rysowaniu głowy)*

Aby sprawdzić, czy rysowanie ogona działa poprawnie

- W procedurze FormActivate wpisz: **OGON[1,1]:=1; OGON[2,1]:=1;**
- Uruchom program, zjedz czerwoną kropkę - biała kropka powinna pojawić się w punkcie (1,1) tablicy

(30) Pokaż wynik działania programu - biała kropka po zjedzeniu jabłka

*po sprawdzeniu usuń obie instrukcje z FormActivate*

*Poszczególne elementy ogona powinny poruszać się w identyczny sposób, jak głowa glizdy. Za każdym razem, gdy glizda się poruszy przepiszemy wszystkie komórki tablicy OGON o jeden w prawo (na następny) a do pierwszego wpiszemy poprzednie położenie głowy. Przeliczenie ogona należy wykonać przed wyliczeniem nowej pozycji głowy. Wyliczeń dokonujemy w oparciu o PUNKTY jeśli ujemne, to ogon tak długo nie urośnie, aż zjemy odpowiednią ilość dobrych jabłek.*

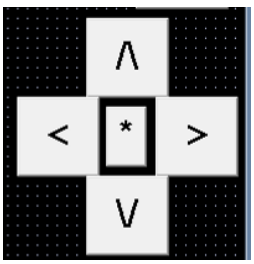
- Utwórz procedurę RUSZ_OGON bez parametrów.
- W procedurze wpisz instrukcje
  - W pętli FOR od PUNKTY aż do 2 (z odliczaniem wstecz)
    - do komórki tablicy OGON o współrzędnych [1, i] wstaw komórkę OGON[1, i-1]
    - w podobny sposób przestaw komórkę z drugiego wiersza *(zmienna „i" steruje pętlą FOR)*
  - Do komórki o współrzędnych [1,1] wstaw GLx
  - Do komórki o współrzędnych [2,1] wstaw GLy
- Umieść procedurę RUSZ_OGON w Timer1Timer przed instrukcjami wyliczającymi nowe położenie głowy

(31) Pokaż wynik działania programu - po zjedzeniu jabłka ogon glizdy się zwiększa lub zmniejsza i przesuwa się za głową

## STRZAŁ

*Po naciśnięciu spacji lub wciśnięciu przycisku na formatce głowa glizdy oddaje strzał (jeśli ma czym strzelać, tzn. ma ogon). Ponieważ jest to kolejna animacja (osobna) dlatego wykorzystamy kolejny obiekt Timer. Ustalimy, że glizda strzela na odległość 20 kratek, a sam pocisk (dwa razy mniejszy) porusza się 5 razy szybciej od samej glizdy. Jeśli strzał trafi po drodze na jabłko to je „zjada"*

- Zadeklaruj zmienne globalne STx, STy, Sx, Sy, STd typu integer (parametry strzału)
- Wstaw na formatkę nowy obiekt **TIMER**
- Własność **ENABLED** zegara ustaw na FALSE
- Utwórz na formatce nowy przycisk z napisem „*" **(**jak na obrazku)
- Kliknij w niego podwójnie i wpisz instrukcje
  - Jeżeli PUNKTY większe od zera to wykonaj instrukcje
    - Zmniejsz PUNKTY o jeden
    - Włącz drugi zegar instrukcją: **Timer2.Enabled:=True;**
    - Do zmiennych STx i STy wstaw zmienne położenia głowy glizdy
    - Do zmiennych Sx i Sy wstaw odpowiednie kierunki ruchu glizdy *(strzał w tą samą stronę)*
    - Do zmiennej STd wstaw 20 *(zasięg strzału)*



    - Ustaw szybkość zegara: Timer2.Interval:=Timer1.Interval div 5; *(5 razy szybciej leci pocisk)*
- Kliknij podwójnie w **TIMER2**
- Do procedury **Timer2Timer** wpisz instrukcje animacji pocisku
    - Zwiększ współrzędne STx i STy o odpowiednio Sx i Sy
    - Zmniejsz zasięg strzału o 1 (zmienna STd)
    - Jeżeli zasięk jest równy zero to wyłącz zegar **Timer2.Enabled:=false;**
    - instrukcja PRZERYSUJ;
- Do procedury **PRZERYSUJ** dołożyć instrukcję warunkową:
    - Jeżeli zegar 2 jest włączony (własność Enabled=True) to rysuj kropkę o współrzędnych (STx,STy), w kolorze białym, dwa razy mniejszą od Kro (Kro div 2)
- Do procedury FormKeyDown dołożyć
- obsługę klawisza **SPACJA** (kod 32) - wywołać procedurę kliknięcia w przycisk STRZAŁ
- instrukcję **key:=0;** *(tylko jeden strzał)*

(32) Pokaż wynik działania programu - można strzelać klawiszem SPACJA lub przyciskiem

*Żeby pocisk nie wychodził poza kratki*

- W procedurze **Timer2Timer** dopisz instrukcje warunkowe do obliczania położenia pocisku
- Jeżeli współrzędna pocisku jest większa od 1 i jest mniejsza niż ilość kratek+1, to oblicz nowe położenie

(33) Pokaż wynik działania programu - pocisk nie wychodzi poza kratki

*Pocisk sprawdza, czy trafił na jabłko w tablicy JAB w podobny sposób, jak robi to głowa glizdy*

- W procedurze Timer2Timer, przed przerysowaniem ekranu wpisz instrukcje
- Do zmiennej J wstaw zawartość komórki tablicy JAB o współrzędnych pocisku
    - Jeżeli w zmiennej J jest liczba różna od zera to wykonaj
        - zwiększ PUNKTY o J
        - wpisz PUNKTY do etykiety z punktacją
        - wyzeruj komórkę tablicy JKB o współrzędnych pocisku
        - wyłącz zegar 2 instrukcją: **Timer2.Enabled:=false;**

(34) Pokaż wynik działania programu - pocisk „zjada” jabłka

ILE JABŁEK

*Po zjedzeniu wszystkich dziesięciu zdrowych jabłek, losujemy kolejne dziesięć. Jak sprawdzić, ile jabłek jeszcze zostało do zjedzenia? Napiszemy funkcję ILE_JAB bez parametrów, której wynikiem będzie ilość zdrowych jabłek w tablicy, Funkcję wstawiamy w dwóch miejscach: gdy jabłko zjada głowa glizdy lub pocisk wystrzelony przez glizdę. Gdy funkcja da wynik 0 - nastąpi nowe losowanie.*

- Nagłówek funkcji **Function TForm1.ILE_JAB:integer;**
- Zmienną zawierającą sumę jabłek wyzeruj
- Podwójna pętla - zmienne sterujące **I** i **J** od **1** do **KRX+1** i **KRy+1**
    - Jeżeli komórka tablicy JAB[I, J]=1 to sumę jabłek zwiększ o 1
- Zmienną z sumą jabłek przypisz do nazwy funkcji
- Zadeklaruj zmienne sterujące i zmienną z sumą jabłek
- W procedurach **Timer1Timer** oraz **Timer2Timer**, w częściach programu „gdy zjedzone jabłko” umieść instrukcję
    - jeżeli wynik funkcji ILE_JABLEK jest równy zero to wykonaj instrukcję **LOSUJ(10,0,0);**

(34) Pokaż wynik działania programu - zjedz 10 jabłek - pojawią się nowe

DRUGI GRACZ (pytania na 4-5)

*Na ekranie pojawi się druga glizda - sterowana przez komputer. Komputerowa glizda może wykonywać dokładnie te same operacje, co glizda sterowana przez użytkownika. komputerowa glizda musi jednak szukać jabłek i je zjadać. Będzie również można sterować drugą glizdą ręcznie i zagrać w grę z inną osobą.*
*W tym celu program zostanie zmodyfikowany. Kratownica będzie miała konkretne wymiary 40x30 i nie można będzie zmieniać żadnych parametrów za pomocą suwaków. Z poprzedniej wersji programu usunięto wszystkie niepotrzebne przyciski i suwaki i procedury. Można też będzie sterować kursorami.*
*Obsługa kumpla będzie polegała na obsłudze nowych klawiszy oraz obsłudze dwóch nowych timerów. Możliwość grania z drugą osobą pojawi się po wciśnięciu klawisza.*

- Usuń wszystkie **przyciski, suwaki, etykiety i procedury** z nimi związane
- Ustaw **kratownicę 40x30** i **formatkę 620x820**
- Umieść na formatce dwa obiekty **Timer**
- **Zadeklaruj** zmienne niezbędne dla drugiego gracza (ramka)
- Właściwość **Timer3.Interval** ustaw na **200** milisekund
- Właściwość **Timer3.Enabled** ustaw na **False** *(druga glizda nie chodzi)*
- Właściwość **Timer4.Enabled** ustaw na **False** *(druga glizda nie strzela)*

```
GLx1,GLy1,dx1,dy1:integer;
OGON1:array[1..2,1..100] of integer;
PUNKTY1:integer;
STx1,STy1,Sx1,Sy1,STd1:integer;
```

- W procedurze **FormActivate**
  - **wyzeruj** zmienne
  - **wylosuj** położenie drugiej glizdy
- W procedurze **PRZERYSUJ**
  - narysuj kropkę o współrzędnych **GLx1, GLY1** w kolorze **clLime** jeżeli właściwość **ENABLED Timera3** jest równa **TRUE** *if Timer3.Enabled=true then KKROPKA(GLx1,GLy1,KRo,clAqua);*
  - Obsłuż strzał drugiego gracza: jeżeli właściwość Timer4.Enabled jest prawdziwa, to rysuj kropkę ze współrzędnymi STx1, STy1, połowę mniejszą w kolorze białym
  - Punktacja wpisywana do pola caption formatki:
    **Form1.Caption:='GLIZDA 1: '+IntToStr(PUNKTY)+' 2: '+IntToStr(PUNKTY1);**
- W procedurze **Timer3Timer** (kliknij podwójnie w trzeci zegarek)
  - Skopiuj wszystko z procedury Timer1Timer
  - Zamiast RUSZ_OGON wpisz **RUSZ_OGON1**
  - Do wszystkich zmiennych związanych z glizdą dopisz jedynkę (GLx, GLy, dx, dy, PUNKTY)
- W procedurze **FormKeyDown**

*Drugi gracz (Q - strzał)* *Pierwszy gracz (U, DEL, ZEROnum SPACJA- strzał)* *Uruchomienie graczy* *Koniec Pauza*

| Q | W | | U | I | | DEL | /\ | O | 1 | 2 | ESC | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 81 | 87 | | 85 | 73 | | 46 | 38 | 96 | 49 | 50 | 27 | 80 |
| A | S | D | J | K | L | < | \/ | > | | | | |
| 65 | 83 | 68 | 74 | 75 | 76 | 37 | 40 | 39 | | | | |

  - Dokonaj modyfikacji reakcji na klawisze zgodnie z podanymi kodami
  - **Pierwszy gracz**: **uruchomienie** - kod 49, **strzał** - kod 85 lub 46 lub 96
  - **Pauza**: Zatrzymaj lub uruchom w podobny sposób zegar 3
  - **Drugi gracz**: **uruchomienie** - kod 50 uruchomienie zegara 3 (właściwość Enabled=true)i nadanie kierunku, zmienne z jedynkami, **poruszanie** - zmień kody i do zmiennych dopisz jedynki, **strzał** - kod 81 - wpisz uruchomienie procedury STRZEL1 (za chwilę ją napiszemy)
- Utwórz nową procedurę **STRZEL1**
  - Skopiuj całą instrukcję warunkową pod spód i popraw zmienne (PUNKTY, TIMER, STx, STy, GLx, GLy, sx, sy, gx, gy, SRd1)
- W procedurze **ZERUJ_OGON**
  - W pętlach dołożyć zerowanie tablicy OGON1
- W procedurze **RYSUJ_OGON**
  - Skopiować całą pętlę i pozamieniać zmienne
- Napisać nową procedurę **RUSZ_OGON1** *(musi być nowa, bo inaczej dwa zegary przeszkadzają sobie)*
  - Skopiować wszystko z RUSZ_OGON i pozamieniać zmienne
  - Umieść nagłówek w górnej części
- W procedurze **Timer4Timer**
  - Kliknąć podwójnie w zegarek 4
  - Skopiować wszystko z procedury Timer2.Timer
  - pozamieniać zmienne

(35) Pokaż wynik działania programu - druga glizda uruchomiona klawiszem 2

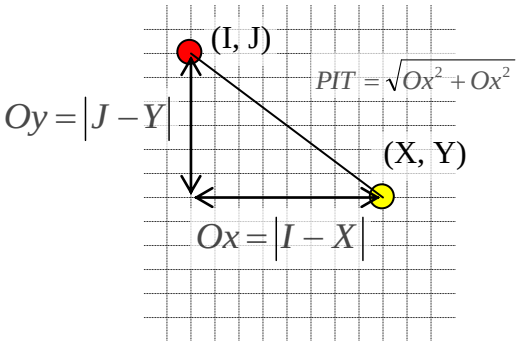
**AUTOMATYCZNA GLIZDA** (pytania na 5-6)

*„Mądra" komputerowa glizda będzie przed każdym swoim ruchem „rozglądać się" wokół siebie, analizować sytuację (gdzie znajduje się najbliższe jabłko) i jak najkrótszą drogą podążać do niego. Najkrótszą drogę najlepiej wyliczyć analizując wszystkie odległości od głowy glizdy do każdego jabłka i wybierając tę najkrótszą. Dodatkowo, jeśli na swojej drodze glizda trafi na „złe" jabłko będzie potrafiła je omijać.*

*Funkcja szukająca najbliższego jabłka sprawdza każdą komórkę tablicy JAB. Jeśli znajdzie komórkę zawierającą liczbę 1, wylicza odległość (Pitagoras), a następnie porównuje z obliczoną poprzednio najkrótszą odległością (początkowo duża liczba). Jest to prosta realizacja algorytmu znajdowania minimalnej wartości.*
*Z funkcjami jest jednak podstawowy problem - w wyniku podają tylko jedną zmienną, a nam potrzebne jest współrzędna X i Y najbliższego jabłka (dwie wartości). Jak to niedogodność ominąć? Zadeklarujemy nowy typ danych o nazwie PUNKT, który będzie zawierał w sobie te dwie wartości, a funkcja będzie podawać wynik w punktach.*

```
TPunkt=record
  X, Y:integer
end;
```

- Zadeklaruj nowy typ danych o nazwie TPunkt typu RECORD zawierający dwie zmienne X i Y typu całkowitego. Umieść nowy typ zaraz po słowie TYPE (przed deklaracją typu TFORM1) w górnej części programu
- Zadeklaruj jedną zmienną o nazwie PUNKT typu TPunkt w zmiennych globalnych
- Utwórz funkcję o nazwie SZUKAJ_JABLKO z parametrami X i Y (podajemy przez nie położenie głowy glizdy), której wynikiem działania jest wartość typu TPunkt: **function TForm1.SZUKAJ_JABLKO(X, Y:integer):TPUNKT;**
- podwójna pętla od 1 do KR_+1
- jeżeli komórka JAB[I,J]=1 to
  - wyliczamy różnicę współrzędnych I i X z wartością bezwzględną (odległość w poziomie), a wynik zapamiętujemy w zmiennej Ox: **Ox:=abs(i-x);**
  - W podobny sposób wyliczamy odległość w pionie, jako różnicę J i Y
  - W zmiennej PIT (od słowa Pitagoras) wyliczamy pierwiastek (SQRT) z sumy kwadratów (pomnóż przez siebie) obliczonych odległości. Ponieważ wynikiem jest liczba rzeczywista zamieniamy ją na liczbę całkowitą za pomocą funkcji ROUND.
  - Jeżeli obliczona odległość w PIT jest mniejsza od poprzednio najmniejszej w zmiennej NAJ to
    - do zmiennej NAJ zapisz PIT
    - Zapamiętaj położenie chwilowo najbliższego jabłka: **PUNKT.X:=I; PUNKT.Y:=J;**




- Na końcu funkcja do swojej nazwy musi przypisać ostatnio znaleziony punkt (minimalna odległość): **SZUKAJ.JABLKO:=PUNKT;**

*Aby sprawdzić działanie funkcji wpisz w procedurze **FormKeyDown** polecenia z ramki Gdy naciśniemy klawisz „zero" pokaże się okienko z komunikatem o najbliższym jabłku. Ponowne uruchomienie: ENTER i klawisz „P"*

(36) Pokaż wynik działania programu - wciskamy klawisz „zero" pojawi się komunikat

```
if key=48 then
  begin
    Timer1.Enabled:=false;
    PUNKT:=SZUKAJ_JABLKO(GLx,GLy);
    showmessage(IntToStr(PUNKT.x)+' '+IntToStr(PUNKT.y));
  end;
```

*Jak wyznaczyć drogę komputerowej glizdy? Zna położenie najbliższego jabłka, więc możemy się umówić, że najpierw idzie w prawo lub w lewo, aż zrówna się w pionie, a potem w górę lub w dół. Za każdym razem będzie jednak analizować swoje położenie i jeśli pojawi się jabłko leżące bliżej, komputerowa glizda podąży do nowego celu. Aby glizda wiedziała, w którą stronę się udać, wyliczymy dla niej w funkcjach kierunki Dx1 i Dy1. W jednej funkcji oba kierunki, a skoro funkcja ma podać dwie wartości, to kolejna deklaracja typu TPUNKT. Zakładamy również, że komputerowa glizda najpierw będzie poruszała w pionie lub w poziomie w zależności od krótszej odległości (w pionie lub w poziomie). Funkcja musi uwzględniać specyfikę ruchu glizdy: albo w pionie, albo w poziomie, nie może po skosie (oba kierunki różne od zera).*
*Przy okazji zauważmy, że wszystkie poprzednie wyliczenia położeń glizd, wszystkie współrzędne tablicowe i wykonywane obliczenia można by realizować nie na osobnych współrzędnych, ale na wartościach typu TPUNKT*

- Zadeklaruj nową zmienną o nazwie **KIER** typu **TPUNKT** w zmiennych globalnych.
- Utwórz nową funkcję o nazwie **SZUKAJ_KIER**, której parametrami są: **X, Y** typu całkowitego (położenie głowy glizdy) i **J** typu **TPunkt** (położenie jabłka), a wynik działania funkcji jest typu **TPunkt** (kierunek, w którym ma poruszać się glizda.
- Zadeklaruj w procedurze dwie zmienne pomocnicze typu **TPUNKT** o nazwach **G** i **O**

- Wyzeruj elementy zmiennej G: **G.x:=0; G.y:=0;**
- Wylicz odległości w pionie i poziomie pomiędzy jabłkiem a głową glizdy *(do zmiennej O)*
- **JEŻELI** odległość w poziomie **O.x** jest mniejsza od odległości w pionie **O.y** *(najpierw ruch w pionie, w górę jeśli jabłko wyżej a w dół jeśli jabłko niżej od głowy)*
  - **TO JEŻELI** różnica pomiędzy położeniem głowy glizdy w pionie **Y** i położeniem jabłka w pionie **J.Y** jest większa od zera **TO** kierunek ruchu glizdy w górę **G.y:=-1 W PRZECIWNYM RAZIE** kierunek w dół **G.Y:=1**
  - **W PRZECIWNYM RAZIE** *(pierwszy warunek - gdy najpierw ruch w poziomie)* **JEŻELI** różnica pomiędzy położeniem głowy glizdy w poziomie **X** i położeniem jabłka w poziomie **J.X** jest większa od zera **TO** kierunek ruchu glizdy w lewo **G.x:=-1 W PRZECIWNYM RAZIE** kierunek w prawo **G.X:=1**
- Do nazwy funkcji przypisujemy zmienną **G** *(w której mamy kierunki ruchu glizdy)*

*Schematyczna konstrukcja tej złożonej instrukcji warunkowej w ramce. Zwróć uwagę na tylko jeden średnik na końcu.*

```
IF O.x < O.y
  THEN IF Y - J.Y THEN … ELSE …
  ELSE  IF X - J.X THEN … ELSE …;
```

*Aby sprawdzić działanie funkcji wpisz w procedurze **FormKeyDown** pierwszą instrukcję z ramki: KIER…, a drugą popraw. Gdy naciśniemy klawisz „zero" pokaże się okienko z komunikatem o najbliższym jabłku i kierunku do niego. Ponowne uruchomienie: ENTER i klawisz „P"*

```
KIER:=SZUKAJ_KIER(GLx,GLy,PUNKT);
showmessage(IntToStr(PUNKT.x)+' '+IntToStr(PUNKT.y)+
      chr(13)+IntToStr(KIER.x)+' '+IntToStr(KIER.y));
```

(37) Pokaż wynik działania programu - wciskamy klawisz „zero" pojawi się komunikat o kierunku ruchu

*Wszystko gotowe do zasymulowania komputerowej glizdy*

- W procedurze **Timer3Timer** przed ręcznym wyliczaniem nowego położenia glizdy wstawiamy polenienia z ramki

```
PUNKT:=SZUKAJ_JABLKO(GLx1,GLy1);
KIER:=SZUKAJ_KIER(GLx1,GLy1,PUNKT);
dx1:=KIER.x;
dy1:=KIER.y;
```

(38) Pokaż wynik działania programu - uruchamiamy obu użytkowników i na wyścigi zbieramy glizdy

## AUTOMAT-DRUGI GRACZ

*Aby można było przełączać się w trakcie działania programu pomiędzy automatem a normalnym przeciwnikiem wykonaj poniższe czynności.*

- Zadeklaruj zmienną globalną o nazwie **AUTOMAT** typu logicznego (**boolean**);
- W procedurze **FormActivate** zmienne AUTOMAT przypisz wartość FALSE (glizda ręcznie sterowana)
- W procedurze obsługi klawiszy **FormKeyDown** dla klawisza „zero" wpisz instrukcję **AUTOMAT:-NOT(AUTOMAT);**
- Usuń z obsługi klawisza „zero" komunikaty o jabłku i kierunkach
- W procedurze **Timer3Timer**
- Jeżeli **AUTOMAT=True** to wykonuj obliczenia automatyczne w przeciwnym wypadku wykonuj obliczenia ręczne

(39) Pokaż wynik działania programu - klawiszem „ZERO" można przełączać tryby pracy drugiego gracza

## GLIZDA STRZELA

*Podczas pracy automatycznej drugi użytkownik ma możliwość strzelania klawiszem „Q". Najlepiej by było gdyby glizda automatyczna też sama strzelała i szybciej zbierała punkty. Jak glizda sama strzeli? Jeśli na kierunku ruchu glizdy znajduje się jabłko i odległość jest mniejsza od 21 a większa od np. 5 (na krótsze odległości nie opłaca się strzelać), to glizda sama wykona strzał. Jeśli glizda wykonuje pod koniec swoje „wahadłowe" ruchy to i tak nie strzeli. Co to znaczy, że na kierunku ruchu glizdy znajduje się jabłko? **Jeśli glizda przesuwa się do góry (dy1=-1) to przeszukamy wszystkie komórki powyżej głowy i jeśli jest tam komórka=1 a odległość pomiędzy nimi jest mniejsza niż 21 a większa niż 5 ((O.y>5) and (O.y<21)), to strzelamy.** Gdy glizda przesuwa się w dół oraz gdy ruch w poziomie obliczenia wykonujemy w podobny sposób. Są to cztery prawie identyczne serie operacji i można je połączyć w jedno, co przedstawia poniższy algorytm.*

- Utwórz nową procedurą o nazwie **CZY_STRZELAC**, bez parametrów, której wynikiem jest typ logiczny

*Można podać do funkcji niezbędne dane (pozycja jabłka, pozycja głowy, kierunki, ale te wszystkie parametry trzymamy w zmiennych globalnych i możemy w każdej chwili z nich skorzystać)*

- Zadeklaruj zmienną sterującą pętlą **I** typu całkowitego, zmienną **CS** typu logicznego i zmienną **O** typu TPUNKT
- Wpisz do **CS** wartość fałsz *(nie strzelać)*
- Wpisz do zmiennej **I** liczbę 1 *(pierwsza kratka za głową glizdy do analizy)*
- **DOPÓKI** zmienna sterująca I jest mniejsza od 21 wykonuj poniższe polecenia
  - **JEŻELI JAB[GLx1+i*dx1,GLy1+i*dy1]=1** *(na kierunku ruchu jest zdrowe jabłko)* **TO**

        - oblicz odległość od jabłka:
          **O.x:=abs(GLx1+i*dx1-GLx1);**
          **O.y:=abs(GLy1+i*dy1-GLy1);**
        - **JEŻELI (O.x+O.y) in [5..20]** *(odległość mieści się za kresie 5 do 20)* **TO**
          - do zmiennej sterującej *I* wpisz 21 *(żeby pętla się skończyła)*
          - do zmiennej **CS** wpisz wartość prawda *(można strzelać)*
    - **JEŻELI JAB[GLx1+i*dx1,GLy1+i*dy1]<1** *(na kierunku ruchu jest chore jabłko)* **TO**
      - do zmiennej sterującej **I** wpisz 21 *(żeby pętla się skończyła)*
    - zwiększ wartość zmiennej sterującej **I** o 1

*koniec pętli DOPÓKI*

- Do nazwy funkcji przypisz zawartość zmiennej **CS**
- W procedurze **Timer3Timer**, po wszystkich obliczeniach, przed przerysowaniem
- wpisz warunek logiczny: JEŻELI wynik funkcji **CZY_STRZELAC** jest prawdziwy i zmienna AUTOMAT jest prawdziwa **TO STRZEL1**

(40) Pokaż wynik działania programu - glizda sama strzela, gdy na kierunku znajduje się zdrowe jabłko

*Glizda komputerowa ma jeszcze jeden problem. Gdy „zobaczy" dobre jabłko, „zamyka oczy" i mknie ku niemu najbliższą drogą nie patrząc jakie jabłka zjada po drodze (z reguły te zatrute). Glizda musi więc sprawdzać dodatkowo, czy następny krok nie wypadnie na takim jabłko i nieco zboczyć z wyznaczonej trasy. Co można zrobić? Jeśli na drodze glizdy stanie złe jabłko to niech wykona jeszcze jeden ruch w tą samą stronę co poprzednio.*

- W procedurze **Timer3.Timer** wpisz instrukcję warunkową
- **JEŻELI JAB[GLx1+KIER.x,GLy1+KIER.y]>=0** (kolejne pole jest puste lub z dobrym jabłkiem) **TO**
    - wpisz do zmiennych dx1 i dy1 obliczony kierunek *(to już tam jest)*

*W przeciwnym wypadku podąży poprzednio obranym kierunkiem. Glizda jest tylko troszkę mądrzejsza. Ale jeśli podąża na wprost, to nie zmieni kierunku i wpadnie. Najlepiej by było gdyby zmieniała kierunek do pozytywnego skutku)*

(41) Pokaż wynik działania programu - glizda omija przeszkody w postaci złych jabłek - niektóre

*Strzelanie do ogona glizdy i obcinanie go. Obcięty ogon zamienia się na jabłka. Kiedy strzelać, żeby to miało sens? Gdy na kierunku ruchu komputerowej glizdy pojawi się głowa wtedy strzał.*

- Uzupełniamy funkcję **CZY_STRZELAC**
- Dokładamy trzeci warunek logiczny sprawdzany w pętli dopóki:
  **JEŻELI (GLx=GLx1+i*dx1) and (GLy=GLy1+i*dy1) TO** wykonaj **i:=21** oraz **CS:=TRUE;**

(42) Pokaż wynik działania programu - glizda strzela, gdy na jej kierunku ruchu jest głowa gracza

*A jak obciąć ogon? Pocisk musi mieć takie same współrzędne jak jedna z kropek ogona. Wtedy wszystkie kropki od tej trafionej do końca zamieniamy na jabłka.*

- W procedurze **Timer4Timer** (leci pocisk) przed poleceniem **PRZERYSUJ**
- w pętli **FOR** gdzie sterująca **I** zmienia się od **1** aż do **PUNKTY** *(PUNKTY - taki długi ogon gracza)*
- **JEŻELI (STx1=OGON[1,i]) and (STy1=OGON[2,i])** *(pocisk pokrywa się z badanym elementem ogona)* **TO**
    - w pętli **FOR** gdzie sterującą **J** *(już zadeklarowana wcześniej)* zmienia się od **I** aż do **PUNKTY**
      - wykonujemy **JAB[OGON[1,j],OGON[2,j]]:=1;** *(wszystkie elementy ogona zamieniamy na jabłka)*
    - Zmniejszamy liczbę punktów gracza do poziomu **I-1** *(bo trafiony segment też zamienia się na jabłko)*

(43) Pokaż wynik działania programu - glizda strzela, i obcina ogon gracza, który zamienia się na jabłka

- W procedurze Timer2Timer wykonaj podobne operacje - tym razem gracz obcina ogon komputerowi

**Co można poprawić?**

Końcowy etap dochodzenia glizdy do jabłka, gdy odległość w pionie jest równa odległości w pionie glizda pokonuje „zakrętasami". Jak ją tego oduczyć? Jak nauczyć glizdę , że ma zejść do „zera" a dopiero potem zmienić kierunek?

Glizda nie umie przechodzić sama na drugą stronę planszy - nie umie obliczać odległości związanych z przejściem na drugą stronę

Glizda umie strzelać tylko do głowy glizdy gracza, gdy jest na wprost. Gdyby tak umiała wyliczyć, że za chwilę fragment ogona (lub głowa) będzie się znajdować w konkretnym miejscu (jeśli gracz nie zmieni kierunku) i wystrzelić wcześniej.

Pojedynczy klawisz uruchamia program w trybie gracz z komputerem