WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY, SZKOŁA NAUK ŚCISŁYCH

UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO
W WARSZAWIE

Gra
"Skoki narciarskie"

Autorzy:
Krzysztof Jura
Michał Gawroński

Warszawa 2013

Spis treści

1. Wstęp

Opisywana w niniejszym opracowaniu aplikacja jest grą - symulatorem skoków narciarskich, wykorzystującym zasady skoków narciarskich i prawa fizyki obowiązujące w rzeczywistości [1]. Została napisana obiektowo w języku C++ [2] przy użyciu środowiska programistycznego *Microsoft Visual C++ 2008 i 2010.* Do wyświetlania grafiki i odtwarzania dźwięków użyta została biblioteka *SDL* [3].

Gra należy do gatunku zręcznościowych, w których umiejętności gracza decydują o wynikach w grze. Gracz wciela się w rolę skoczka narciarskiego, który bierze udział w konkursie skoków narciarskich na jednej z dwóch dostępnych skoczni. Jego zadaniem jest oddać jak najdłuższy skok i ustać po wylądowaniu, gdyż od tego zależą jego wyniki. Obowiązuje zasada: im dłuższy ustany skok, tym uzyskany wynik jest lepszy, a miejsce w rankingu wyższe.

Podczas skoku mamy możliwość manipulacji następującymi kątami: nachylenia skoczka do podłoża (strzałki góra i dół) oraz nachylenia nart do podłoża (strzałki lewo i prawo). Aby oddać najdłuższy skok, należy optymalnie ustawić oba kąty tak, aby najlepiej wykorzystać siłę nośną i ograniczyć opór powietrza. Wynik jest dodatni tylko przy prawidłowym lądowaniu, gdy kąt nachylenia nart lub/i skoczka do stoku był nieprawidłowy – wynik równy jest zeru.

Aplikacja umożliwia grę w trybie zarówno jedno-osobowym jak i wielo-osobowym. W przypadku rozgrywki w trybie wielo-osobowym gracze będą oddawać skoki jeden po drugim. Dla zwiększenia poziomu trudności rozgrywki istnieje możliwość dodawania do konkursu graczy komputerowych o trzech różnych poziomach trudności. Domyślnie gracze ludzcy widzą skoki oddawane przez zawodników komputerowych, lecz dla skrócenia czasu gry istnieje możliwość pokazywania jedynie wyników tzn. odległości uzyskanych przez graczy komputerowych.

Po ukończonym konkursie na ekranie pokazywane są wyniki konkursu, a istotne informacje dotyczące graczy ludzkich biorących udział w grze zostają zapisane w bazie danych, z której później są pobierane do wyświetlania rankingów graczy i rekordów skoczni.

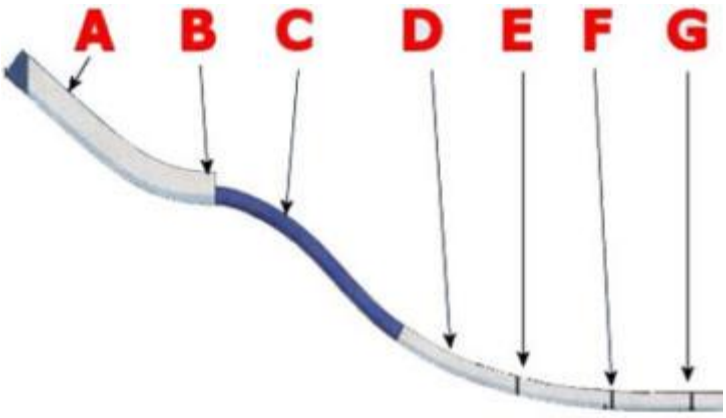
2. Opis teoretyczny

2.1 Zasady i punktacja

Zawody składają się z dwóch serii skoków. Każdy zawodnik wykonuje po jednym skoku w danej serii. W przeciwieństwie do prawdziwych konkursów - klasyfikacja zawodników odbywa się jedynie na podstawie długości skoku, o ile skoczek ustał przy lądowaniu.

2.2 Skocznia

Zawody Poniżej przedstawiono schemat skoczni narciarskiej (rys.1) podzielony na 7 części: belka startowa, próg skoczni, bula, punkt konstrukcyjny, linia bezpieczeństwa, szpaler choinek, dojazd.



Rys.1

A) *Belka startowa*

Belka startowa określa długość najazdu. Podnosząc bądź opuszczając belkę, sędziowie mogą regulować długość rozbiegu.

B) *Próg skoczni*

Próg skoczni jest miejscem, w którym zawodnik się odbija. Próg opada pod kątem ok. 10-11 stopni. Do lat 80-tych skocznie (np. Wielka Krokiew) miały progi poziome, wręcz katapultujące narciarza. Ich przebudowa przyczyniła się do skrócenia, ale i zwiększenia bezpieczeństwa skoków.

C) *Bula*

Bula to inaczej grzbiet skoczni, za nią zaczyna się zeskok - miejsce lądowania skoczków. Lądowanie tuż za bulą oznacza, że zawodnik albo zupełnie się nie odbił, albo odbił się zbyt wcześnie.

D) *Punkt "K" (konstrukcyjny)*

Punkt "K" służy do oceny długości skoku. Jeśli zawodnik wyląduje dokładnie w punkcie K, zdobywa 60 punktów. Jeśli go przeskoczy, to za każdy dodatkowy metr dostaje 1,8 punktu (na skoczni dużej) dodatkowo. Jeśli zaś nie doskoczy, to liczbę brakujących metrów mnoży się przez 1,8 i odejmuje od końcowej punktacji. Gdy skok jest bardzo krótki, może się zdarzyć, że zawodnik nie dostanie żadnych punktów za długość, ale jedynie za styl skoku.

E) *Linia bezpieczeństwa (punkt "J" - jury)*

Linia bezpieczeństwa wyznacza koniec strefy bezpiecznych skoków. Poza nią zawodnik nie powinien lądować, bo teren niebezpiecznie się spłaszcza.

F) *Szpaler choinek (linia upadku)*

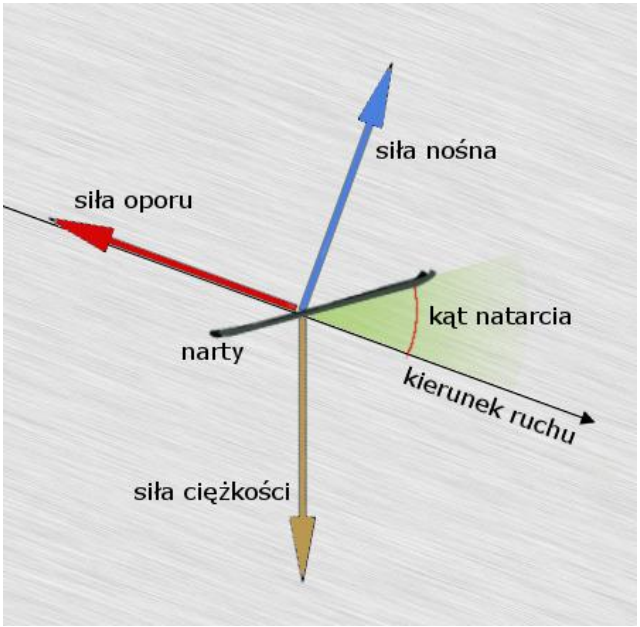
Szpaler choinek to linia, za którą zawodnik zjeżdżający po stoku zaczyna powoli hamować.

G) *Dojazd (wybieg)*

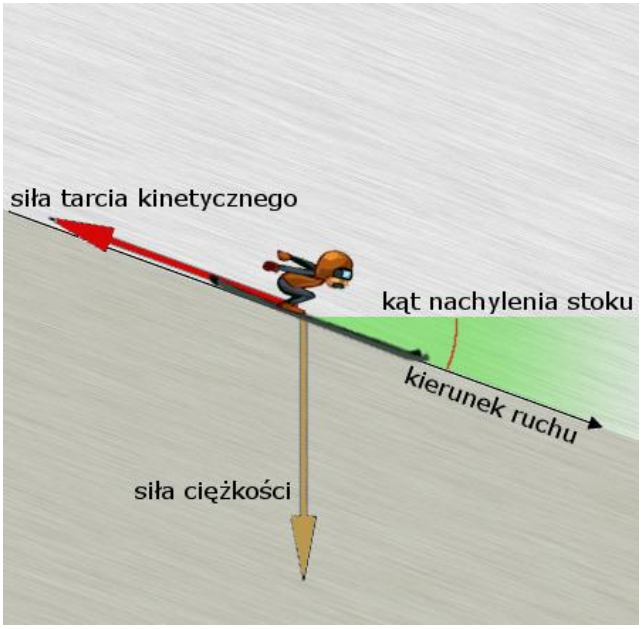
Dojazd określa obszar, gdzie zawodnicy wyhamowują do całkowitego zatrzymania.

2.3 Model fizyczny skoków narciarskich

Skok można podzielić na 3 fazy: rozbieg, lot, hamowanie po lądowaniu. W pierwszej i ostatniej fazie skoczek porusza się po wyprofilowanym stoku, gdzie działa na niego siła grawitacji oraz siła tarcia, a kierunek ruchu wymusza kąt nachylenia podłoża. W fazie lotu oprócz siły ciężkości działa na niego wiatr, opór powietrza i siła nośna.



Rys.2 Ilustracja sił działających na skoczka podczas lotu



Rys.3 Ilustracja sił działających na skoczka podczas zjazdu

Siła nośna:

Siła nośna jest składową siły aerodynamicznej powstającej przy ruchu ciała w płynie względem tego płynu, prostopadłą do kierunku ruchu. Współczynnik C_z zależy od kąta natarcia.

$$P_z = C_z \cdot \rho \cdot S \cdot \frac{V^2}{2} \tag{2.1}$$

P_z - wytworzona siła nośna
ρ - gęstość powietrza
S - pole powierzchni skoczka i nart
$\overrightarrow{v}$ - wektor prędkości ciała względem powietrza

Tarcie kinetyczne:

Siła tarcia kinetycznego to rodzaj siły tarcia suwnego (ślizgowego), gdy ciało jest w ruchu. Występuje tu na styku nart ze śniegiem, gdyż styk ten przenosi siłę nacisku skoczka na podłożę. Siła t.k. ma kierunek ruchu wzajemnego ciał, a zwrot przeciwny do zwrotu ruchu.

Siła ta jest proporcjonalna do siły dociskającej powierzchnie trące, w naszym przypadku składowej siły ciężkości prostopadłej do powierzchni podłoża.

$$F = \mu * g * cos(\alpha) * m \tag{2.2}$$

F - siła tarcia kinetycznego
α - kąt nachylenia skoczni
μ - współczynnik tarcia
g - przyspieszenie ziemskie
m - masa skoczka z nartami

Opór aerodynamiczny w locie skoczka:

$$|\vec{D}| = 0.5 * \rho * C_D * |\vec{v}|^2 * S_D \tag{2.3}$$

Zgodnie z drugą zasadą dynamiki przyspieszenie ciała jest proporcjonalne do wypadkowej siły *F* działającej na to ciało i odwrotnie proporcjonalne do masy ciała *m*. Kierunek i zwrot przyspieszenia pokrywa się z kierunkiem i zwrotem siły.

$$|\vec{a_D}| = \frac{|\vec{D}|}{m} \tag{2.4}$$

$\vec{D}$ - wektor siły oporu skierowany przeciwnie do wektora prędkości ciała względem powietrza

C_D - współczynnik siły oporu

S_D - powierzchnia rzutu skoczka i nart na płaszczyznę prostopadłą do wektora prędkości ciała względem powietrza

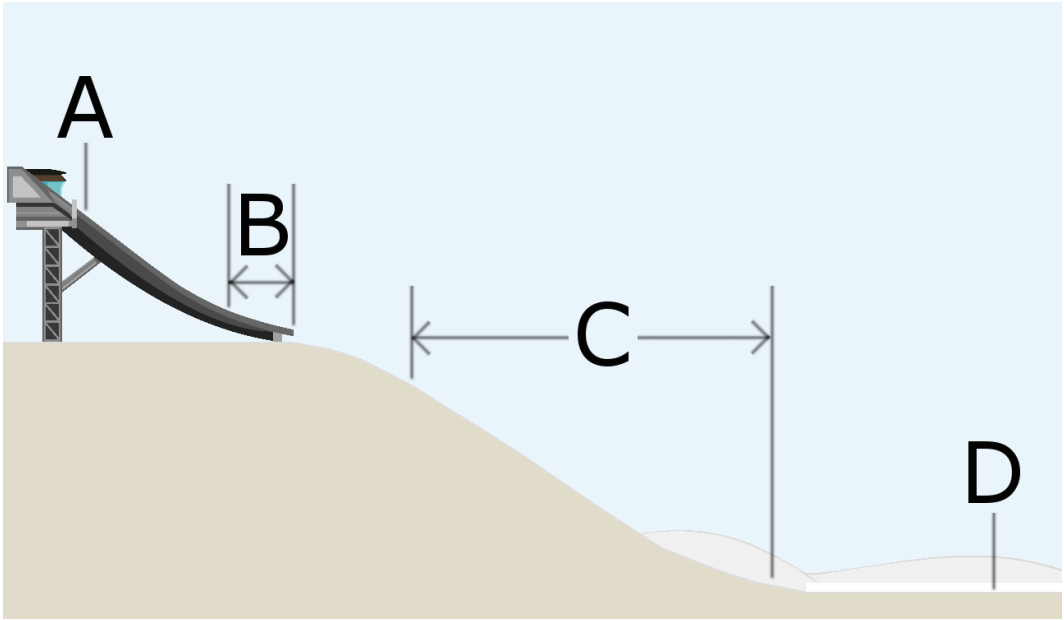
$\vec{v}$ - wektor prędkości ciała względem powietrza

ρ - gęstość powietrza

m - masa skoczka z nartami

2.4 Opis równań ruchu

Równania obu ruchów:



Rys.4 schemat skoczni z punktami charakterystycznymi

A – Belka startowa.
B – Punkt wybicia się zawodnika z progu. Zależy od momentu, w którym gracz wciśnie spację.
C – Punkt lądowania. Zależy od sylwetki zawodnika w czasie lotu.
D – Punkt zatrzymania.

Dla każdej ze skoczni, wartości punktów na osi x różnią się.

Umiejscowienie punktów na osi na skoczni Gawronowo:
$A = 352$
$1304 \leq B \leq 1419$
$1700 \leq C \leq 3500$
$D = 4720$

Umiejscowienie punktów na osi na skoczni Jurowo:
$A = 442$
$2080 \leq B \leq 2290$
$1650 \leq C \leq 4150$
$D = 6280$

Legenda

μ – współczynnik tarcia
g – przyspieszenie ziemskie
C_x – współczynnik siły nośnej, zależy od postawy narciarza
v – wektor prędkości skoczka
v_w – składowa wektora prędkości wiatru w kierunku przeciwnym do ruchu skoczka
α – kąt ruchu skoczka względem poziomu morza
β – kąt nachylenia skoczka względem poziomu morza
C_d – współczynnik siły oporu
ρ – gęstość powietrza
m – masa skoczka z nartami
S – powierzchnia całkowita skoczka i nart
t – czas

Dla $A \leq x \leq B$ (zjazd po rozbiegu) oraz dla $C \leq x \leq D$ (zjazd po stoku):

$$x(t) = cos(\alpha(t)) * (v * t + G * sin(\alpha(t)) * \frac{t^2}{2}) - (G * cos(\alpha(t)) * \mu * t^2) \quad (2.5)$$

$$y(t) = sin(\alpha(t)) * (v * t + G * sin(\alpha(t)) * \frac{t^2}{2}) - (G * cos(\alpha(t)) * \mu * t^2) \quad (2.6)$$

W punkcie B następuje wybicie.

Dla $B \leq x \leq C$ (lot):

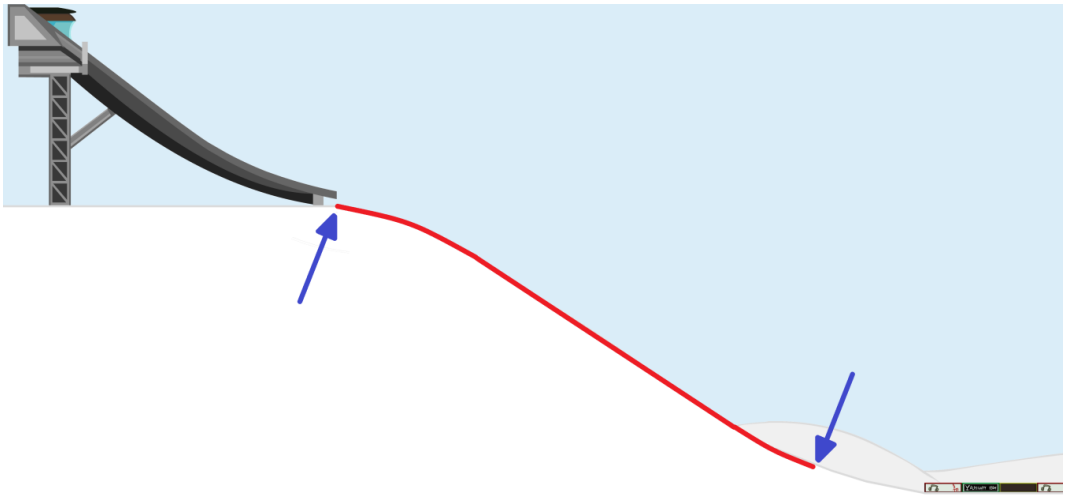
$$x(t) = cos(\alpha(t)) * (v * t + |sin(\alpha(t) - \beta(t))| * (v - v_w) * \rho$$
$$* S * C_x * \frac{1}{m} * t^2 + \rho * \frac{(v - v_w)^2}{2} * |sin(\alpha(t) - \beta(t))| * S * C_d * \frac{1}{m} * t^2) \quad (2.7)$$

$$y(t) = sin(\alpha(t)) * (v * t + \frac{g * t^2}{2} + |sin(\alpha(t) - \beta(t))| * (v - v_w) * \rho$$
$$* S * C_x * \frac{1}{m} * t^2 + \rho * \frac{(v - v_w)^2}{2} * |sin(\alpha(t) - \beta(t))| * S * C_d * \frac{1}{m} * t^2) \quad (2.8)$$

W punkcie C następuje lądowanie.
W punkcie D następuje zatrzymanie.

2.5 Sposób mierzenia długości skoku

Odległość na jaką skoczył skoczek mierzymy po długości krzywej stoku, od punktu pod progiem wybicia do miejsca styku nart z podłożem. Jak poniżej:



Rys.5 schemat skoczni z zaznaczoną na czerwono długością skoku

3. Opis programu

3.1 W czym pisany, środowiska itp.

Program został napisany obiektowo w języku C++ przy użyciu środowiska programistycznego *Microsoft Visual C++ 2008 i 2010.* Do wyświetlania grafiki i odtwarzania dźwięków użyta została biblioteka *SDL* wraz z kilkoma bibliotekami o nią opartymi: *SDL_image*, *SDL_ttf, SDL_rotozoom* oraz *SDL_mixer*.

3.2 Opisy algorytmów

Dane:

alpha – kąt określający kierunek ruchu skoczka względem horyzontu, w radianach
v – prędkość ruchu skoczka, w m/s
time – czas rozpoczęcia poprzedniej iteracji
g – przyspieszenie ziemskie w m/s
friction_factor – współczynnik tarcia nart o stok
gradients[] – tablica z kątami nachylenia stoku
MTP – przelicznik z metrów na piksele
area – pole powierzchni rzutu skoczka od przodu
Cx, Cd – współczynniki siły nośnej i oporu
wind_vx, wind_vy – wektory prędkości wiatru w osiach x i y
RO – gęstość powietrza, w kg/m^3

Wynik: nowe wartości v, alpha, x, y

Opis algorytmu ruchu po stoku:

Krok 1: obliczenie nowego odcinka czasowego

```
t := (SDL_GetTicks() - time) / 1000.0;
```

Krok 2: obliczenie przyspieszenia

```
a := g * sin(alpha);
```

Krok 3: obliczenie drogi przebytej w odcinku czasu

```
s: = v * t + a * pow(t,2.0) / 2;
```

Krok 4: obliczenie siły tarcia

```
friction := friction_factor * g * cos(alpha) * weight;
```

Krok 5:obliczenie nowej prędkości

```
v := v + a*t – friction / weight * t;
```

Krok 6: uaktualnienie kąta ruchu skoczka

```
alpha := gradients[x];
```

Krok 7: uaktualnienie pozycji skoczka

```
x := x + cos(alpha) * s * MTP;
y := y + sin(alpha) * s * MTP;
```

Krok 8: sprawdzenie czy skoczek wybił się z progu

```
if jumped then
      exit;
else
      goto Krok 1;
```

Opis algorytmu ruchu w locie:

Krok 1: obliczenie nowego odcinka czasowego

```
t := (SDL_GetTicks() - time) / 1000.0;
```

Krok 2: obliczenie przyspieszenia

```
a := g * sin(alpha);
```

Krok 3: obliczenie drogi przebytej w odcinku czasu

```
s: = v * t + a * pow(t,2.0) / 2;
```

Krok 4: rozłożenie wektora prędkości na składowe wzdłuż osi x i y

```
vx := v * cos(alpha);
vy := v * sin(alpha);
```

Krok 5: uwzględnienie wpływu siły ciężkości

```
vy := vy + g*t;
```

Krok 6: obliczenie składowej wektora prędkości wiatru wzdłuż toru lotu skoczka

```
wind_v := sqrt( pow(wind_vx, 2.0) + pow(wind_vy, 2.0) );
wind_angle := asin( (wind_vy)/wind_v);
wind_v := wind_v * abs(cos(wind_angle-alpha));
```

Krok 7: uwzględnienie wpływu siły nośnej

```
lift_F := abs(sin(alpha - beta)) * (v - wind_v) * RO * area * Cx;
vy := vy - lift_F * abs(sin(beta)) / weight * t;
vx := vx + lift_F * abs(cos(beta)) / weight * t;
```

Krok 7: uwzględnienie wpływu siły oporu aerodynamicznego

```
S := abs(sin(alpha - beta)) * area;
drag_F := 0.5 * RO * pow(v,2.0) * S * Cd;
v := v - drag_a / weight * t;
```

Krok 8: uaktualnienie pozycji skoczka

```
x := x + cos(alpha) * s * MTP;
y := y + sin(alpha) * s * MTP;
```

Krok 9: sprawdzenie czy skoczek wylądował

```
if landed then
      exit;
else
      goto Krok 1;
```

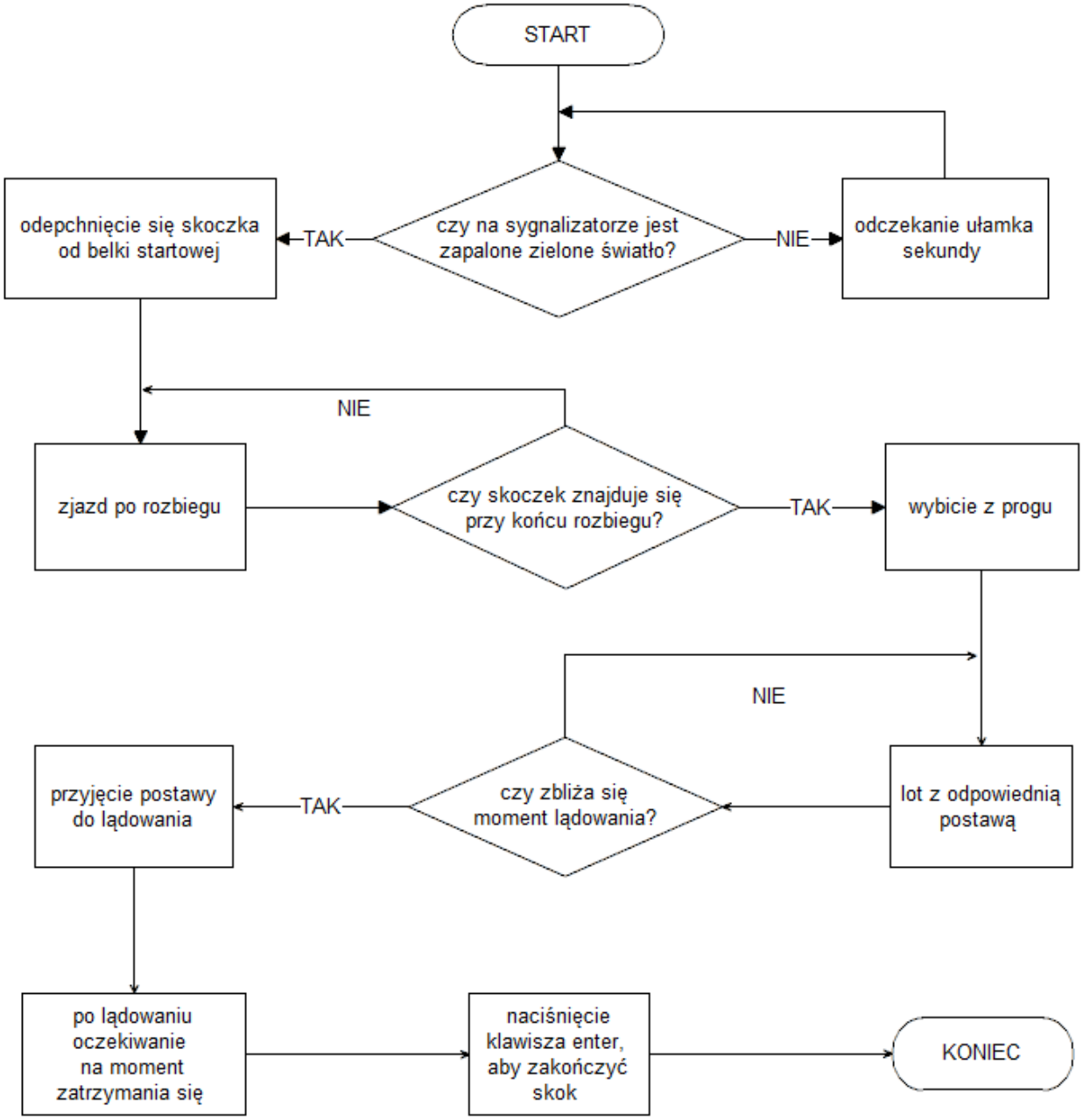
3.3 Opisy klas

- background - klasa wszystkich obiektów w tle, z którymi gracz nie może w żaden sposób oddziaływać; jedynym ich zadaniem jest wyświetlanie się na ekranie
- hill – reprezentuje i określa parametry skoczni; wyświetla budynki i konstrukcje związane ze skocznią tj. rozbieg, filary, domek na szczycie, stok do lądowania, wybieg, bandy z reklamami; zawiera informacje o kątach nachylenia ziemi w każdym jej punkcie w celu obliczania kolizji.
- button - klasa do której należą wszystkie przyciski menu, na które możemy klikać, i które wywołują jakąś akcję.
- camera - kamera definiuje miejsce na mapie, które ma zostać w danej chwili wyświetlone na ekranie; od pewnego momentu zaczyna ona podążać za zjeżdżającym po rozbiegu skoczkiem do chwili wyhamowania po lądowaniu.

- game - główna klasa, w która zajmuje się obsługą grafiki, dźwięku oraz klawiatury, służy też do ładowania ustawień początkowych, wyświetlania menu, sprawdzania kolizji etc.
- jumper - (pol. skoczek) obiektem tej klasy jest skoczek, którym steruje gracz zarówno komputerowy jak i ludzki.
- slope - (pol. stok) klasa ta służy głownie do obliczania odległości skoku i kąta nachylenia stoku w danym punkcie.

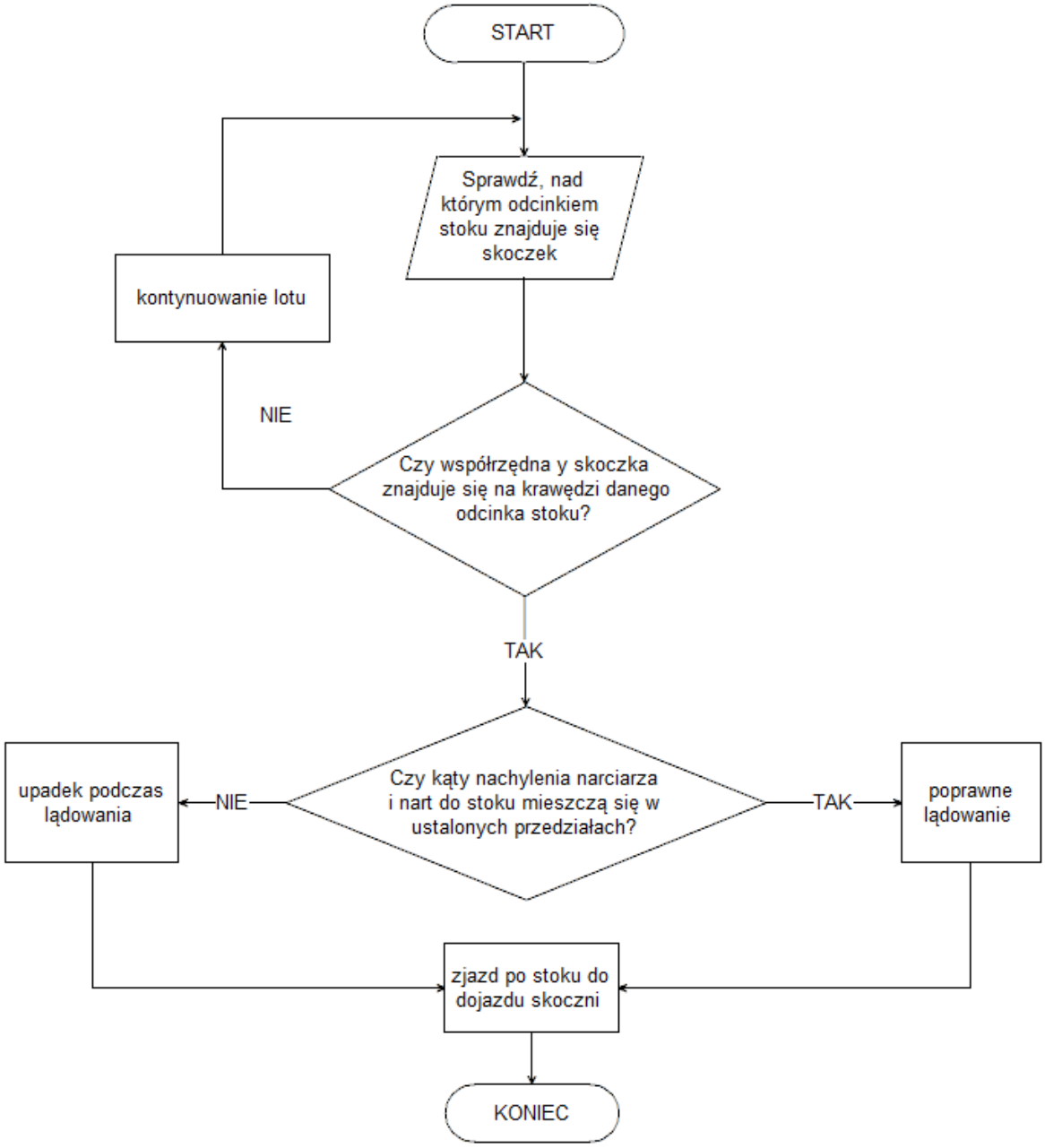
3.4 Ogólny schemat blokowy

- schemat przedstawiający wszystkie fazy skoku narciarskiego

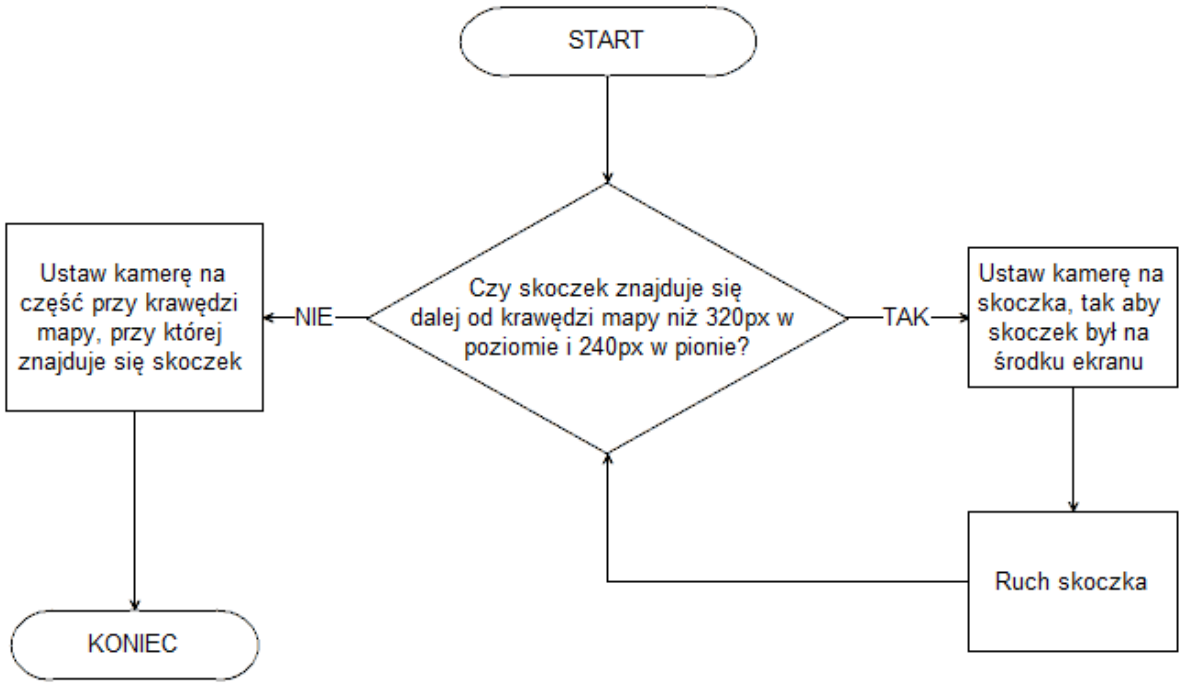


3.5 Szczegółowe schematy blokowe

- wykrywanie kolizji ze stokiem podczas lotu



- działanie kamery podczas skoku



3.6 Opis wybranych procedur

- Procedurą odpowiadającą za ruch skoczka we wszystkich fazach skoku narciarskiego jest metoda *move()* klasy **jumper**.
  Na skoczka podczas trwania całego skoku oddziałuje siła ciężkości. Podczas fazy lotu dodatkowo brany jest pod uwagę wiatr, siła nośna oraz opór aerodynamiczny, natomiast przed wyskokiem i po wylądowaniu tarcie.
  Wiatr działa na zasadzie dodania do prędkości skoczka wartości składowej wektora prędkości wiatru w kierunku przeciwnym do kierunku lotu skoczka.
  Siła nośna obliczana jest w oparciu o różnicę kąta nachylenia nart i skoczka do kąta obecnego toru lotu. Największą siłę nośną otrzymuje skoczek w przypadku nachylenia ciała i nart pod kątem 30 stopni do kierunku ruchu. Wartość siły nośnej maleje liniowo wraz ze zmniejszaniem różnicy kątów do 0 stopni lub wraz ze zwiększaniem różnicy kątów do 60 stopni.
  Opór aerodynamiczny wyhamowuje skoczka i jest największy, gdy skoczek jest ustawiony prostopadle do toru ruchu, a najmniejszy gdy równolegle.
  Po nieudanym lądowaniu skoczek się przewraca, a współczynnik tarcia ulega zmianie: zwiększa się, co powoduje powolniejszy zjazd skoczka po stoku.
  Dodatkowo po upadku odczepiają się narty, które posiadając mniejszy współczynnik tarcia, szybciej zjeżdżają po stoku.

```
void jumper::move()
{
	if(!this->on_bar)
	{
		double t = (SDL_GetTicks() - time) / 1000.0; //delta time
		time = SDL_GetTicks();
		if(t > 0.1){t = 0.00;}
```

```
double a = G * sin(this->alpha);

//odległość jaką pokona skoczek w ciągu jednej iteracji - zależna od zmiany
czasy
double s = this->v * t + a * pow(t,2.0) / 2;

//jeżeli skoczek przekroczy punkt hamowania to zaczyna hamować
if(this->position.x >= ogame.skocznia->slow_down_from){
	ogame.FRICTION_FACTOR = 2.0;
}

//jeżeli skoczek przekroczy punkt bezwzględnego zatrzymania to natychmiast
się zatrzymuje
if(this->position.x >= ogame.skocznia->stop_from)
{
	s = 0.0;
	a = 0.0;
	this->v = 0.0;
	this->x_precise = this->position.x;
	this->y_precise = this->position.y;
}

//kiedy po wylądowaniu skoczek wyhamuje, skok uznaje się za zakończony
if(!v){
	ogame.skok_zakonczony = true;
	if(ogame.id_zawodnika==ogame.lista_zawodnikow.size()) {
		if(ogame.rekord_skoczni_type==1) ogame.rekord_skoczni_type=2;
		ogame.zawody_info = true;
	}
}

//jeżeli skoczek nie wyskoczy z progu to po prostu z niego zjedzie
if(!this->flies && !this->landed && !this->jumped && this->position.x >=
ogame.skocznia->rozbieg_width-1)
{
	this->flies = true;
	this->jump_speed = this->v*3.6;
}


if(this->flies)
{
	//rozkładamy wektor prędkości v na wektory składowe vx i vy
	double vx = this->v * cos(this->alpha);
	double vy = this->v * sin(this->alpha);

	//--SIŁA CIĘŻKOŚCI--//
	double weight_F = this->weight * G;
	double weight_ay = weight_F / this->weight;
	vy += weight_ay*t;
	//--KONIEC SIŁY CIĘŻKOŚCI--//



	//--WIATR--//
	double ro = 1.2; //gęstość powietrza

	//obliczamy wind_v - skladową wektora prędkości w kierunku
	przeciwnym do kierunku lotu skoczka
	double wind_v = sqrt( pow(ogame.wind_vx+ogame.wind_vx_change, 2.0) +
	pow(ogame.wind_vy+ogame.wind_vy_change, 2.0) );
	double wind_angle = asin(
	(ogame.wind_vy+ogame.wind_vy_change)/wind_v);
	if(ogame.wind_vy < 0 && ogame.wind_vx < 0){wind_angle = -
	(180*PI/180) - wind_angle;}
	else if(ogame.wind_vy >= 0 && ogame.wind_vx < 0){wind_angle =
	(180*PI/180) - wind_angle;}

	wind_v *= abs(cos(wind_angle-this->alpha));
```

```
                if(ogame.wind_vy<0){wind_v=-wind_v;}
                //--KONIEC WIATRU--//



                //--SIŁA NOŚNA--//
                double Cx = 10.0; //wspolczynnik siły nośnej
                double lift_F = 0;
                double kat = 0;

                //w przypadku, gdy skoczek nie wyskoczy z progu, nie osiaga sily
                nosnej
                if(!this->jumped) {Cx = 0;}



                kat = abs(this->alpha - jumper_alpha);
                if(kat < PI/6) Cx *= kat*3;
                else if(kat < PI/3) Cx *= (PI - kat*3);
                else Cx=0;
                lift_F = abs(sin(this->alpha - jumper_alpha)) * (this->v - wind_v) *
                ro * jumper_area * Cx;

                kat = abs(this->alpha - skis_alpha);
                if(kat < PI/6) Cx *= kat*3;
                else if(kat < PI/3) Cx *= (PI - kat*3);
                else Cx=0;
                lift_F += abs(sin(this->alpha - skis_alpha)) * (this->v - wind_v) *
                ro * skis_area * Cx;

                double lift_ay = lift_F * abs(sin(jumper_alpha)) / this->weight;
                double lift_ax = lift_F * abs(cos(jumper_alpha)) / this->weight;

                vy -= lift_ay * t;
                vx += lift_ax * t;
                //--KONIEC SIŁY NOŚNEJ--//



                //znow skladamy wektor predkosci
                this->v = sqrt( pow(vx, 2.0) + pow(vy, 2.0) );

                this->alpha = asin(vy/v);

                //--OPÓR AERODYNAMICZNY--//
                double S = abs(sin(this->alpha - skis_alpha)) * skis_area +
                abs(sin(this->alpha -jumper_alpha)) * jumper_area;

                //Cd - współczynnik oporu
                double Cd = (3.0 - 3.0*( abs(90.0 - fmod((abs(this->alpha -
                skis_alpha)*180.0/PI),180.0))/90.0 )) *
                skis_area/(skis_area+jumper_area);

                Cd+= (3.0 - 3.0*( abs(90.0 - fmod((abs(this->alpha -
                jumper_alpha)*180.0/PI),180.0))/90.0 )) *
                jumper_area/(skis_area+jumper_area);

                double drag_F = 0.5 * ro * pow(this->v,2.0) * S * Cd;
                double drag_a = drag_F / this->weight;
                this->v -= drag_a*t;
                //--KONIEC OPORU AERODYNAMICZNEGO--//

        }
        else
        {
                //--TARCIE--//
                double tarcie = ogame.FRICTION_FACTOR * G * cos(this->alpha) * t;
                if(tarcie > v + a*t) tarcie = v + a*t;
```

```
                        this->v += a*t - tarcie;

                        this->alpha = ogame.skocznia->gradients[this->position.x];
                        //--KONIEC TARCIA--//
                }


                this->x_precise += cos(this->alpha) * s * METER_TO_PIXEL;
                this->y_precise += sin(this->alpha) * s * METER_TO_PIXEL;

                //jeżeli obiekt skoczka próbuje opuścić mapę, nie pozwalamy na to
                if( (int)this->x_precise + this->position.w > ogame.skocznia->
                level_width){this->x_precise = ogame.skocznia->level_width - this-
                >position.w;}
                else if( (int)this->x_precise < 0){this->x_precise = 0.0;}
                if( (int)this->y_precise > ogame.skocznia->level_height){this->y_precise =
                ogame.skocznia->level_height;}
                else if( (int)this->y_precise - this->position.h < 0){this->y_precise =
                this->position.h;}

                /* jeżeli tuż po wyladowaniu nastąpi kolizja ze stokiem, podnies skoczka o
                1px do góry tak, aby zjeżdżał po krawędzi stoku, a nie w jego głębi */
                if(this->flies)
                {
                        while( double slope_y = ogame.collision_check(this->x_precise, this-
                        > y_precise) )
                        {
                                if(!slope_y){this->y_precise-=1.0;}
                                else break;
                        }
                }

                this->position.x = (int)this->x_precise;
                this->position.y = (int)this->y_precise;

                //fizyka odjezdzaacych nart po upadku skoczka
                if(this->fell_down){
                        double s_nart = ogame.narty_po_upadku.v * t + a * pow(t,2.0) / 2;

                        double tarcie_nart = ogame.narty_po_upadku.FRICTION_FACTOR * G *
                        cos(ogame.narty_po_upadku.alpha) * t;
                        if(tarcie_nart > ogame.narty_po_upadku.v + a*t) tarcie_nart =
                        ogame.narty_po_upadku.v + a*t;
                        ogame.narty_po_upadku.v += a*t - tarcie_nart;

                        ogame.narty_po_upadku.alpha = ogame.skocznia->
                        gradients[ogame.narty_po_upadku.position.x];

                        ogame.narty_po_upadku.x_precise += cos(ogame.narty_po_upadku.alpha)
                        * s_nart * METER_TO_PIXEL;
                        ogame.narty_po_upadku.y_precise += sin(ogame.narty_po_upadku.alpha)
                        * s_nart * METER_TO_PIXEL;

                        if( (int)ogame.narty_po_upadku.x_precise +
                        ogame.narty_po_upadku.position.w > ogame.skocznia->
                        level_width){ogame.narty_po_upadku.x_precise = ogame.skocznia->
                        level_width - ogame.narty_po_upadku.position.w;}

                        ogame.narty_po_upadku.position.x =
                        (int)ogame.narty_po_upadku.x_precise;
                        ogame.narty_po_upadku.position.y =
                        (int)ogame.narty_po_upadku.y_precise;
                }
        }
}
```

- Procedurą odpowiadającą za wyświetlanie nart i skoczka jest metoda *render()* klasy **jumper**.
  Zajmuje się ona przede wszystkim dobraniem odpowiednich tekstur do fazy skoku, obracaniem ich o odpowiedni kąt i wyświetlaniem w odpowiednim miejscu. Wszystkie tekstury skoczka są w plikach o tym samym rozmiarze i są obracane względem tego samego stałego punktu (pięty), podobnie wszystkie narty (wiązania). Dobór obrazka i kąta obrotu zależy od fazy skoku, tj. zawodnik siedzi na belce startowej, zjeżdża po rozbiegu, leci, zjeżdża po prawidłowym wylądowaniu, zjeżdża na pośladkach po upadku do tyłu lub zjeżdża na brzuchu po upadku do przodu. Potem obliczane są współrzędne, w jakich znalazł się punkt wokół którego chcemy obrócić teksturę, po obróceniu go funkcją rotozoomSurface. Na koniec wyświetlane są tekstury skoczka i nart odpowiednio przesunięte i wyśrodkowane wg punktu obrotu.

```
void jumper::render()
{
	int x = this->position.x - cam.position.x;
	int y = this->position.y - cam.position.y;

	SDL_Surface* temp_jumper; //tekstura tułowia
	SDL_Surface* temp_ski; //tekstura nart

	double j_kat_sr_pkt = 1.9853; //kat miedzy srodkiem obrazka a pkt.obrotu
	double j_odleg_srodek_punkt = 27.313; //odleglosc srodka obrazka od pkt.obrotu
	double s_kat_sr_pkt = 3.07917; //j.w
	double s_odleg_srodek_punkt = 16.0312; //j.w

	//siedzi na belce startowej
	if(this->on_bar)
	{
		temp_jumper = this->icon_jumper_bar;
		temp_ski = this->icon_skis_run;

		s_kat_sr_pkt += this->alpha;

		temp_jumper = rotozoomSurface(temp_jumper, 0, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -this->alpha*180.0/PI, 1, true);
	}

	//zjezdza po rozbiegu lub leci bez wyskoku
	else if((!this->flies && !this->landed) || (this->flies && !this->jumped))
	{
		temp_jumper = this->icon_jumper_squat;
		temp_ski = this->icon_skis_run;

		j_kat_sr_pkt += this->alpha;
		s_kat_sr_pkt += this->alpha;

		temp_jumper = rotozoomSurface(temp_jumper, -this->alpha*180.0/PI, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -this->alpha*180.0/PI, 1, true);
	}

	//leci po wybiciu
	else if(this->flies && this->jumped)
	{
		temp_jumper = this->icon_jumper_fly;
		temp_ski = this->icon_skis_fly;

		j_kat_sr_pkt += this->jumper_alpha + PI/2;
		s_kat_sr_pkt += this->skis_alpha;

		temp_jumper = rotozoomSurface(temp_jumper, -this->jumper_alpha*180.0/PI -
		90, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -this->skis_alpha*180.0/PI, 1, true);
```

```
	}

	//wyladowal poprawnie
	else if(this->landed && !this->fell_down && this->position.x <= ogame.skocznia->
	slow_down_from )
	{
		temp_jumper = this->icon_jumper_land;
		temp_ski = this->icon_skis_run;

		j_kat_sr_pkt += this->alpha;
		s_kat_sr_pkt += this->alpha;

		temp_jumper = rotozoomSurface(temp_jumper, -this->alpha*180.0/PI, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -this->alpha*180.0/PI, 1, true);

	}

	//wyhamowuje na stojaco po poprawnym ladowaniu
	else if(this->landed && !this->fell_down && this->position.x > ogame.skocznia->
	slow_down_from )
	{
		temp_jumper = this->icon_jumper_out;
		temp_ski = this->icon_skis_run;

		s_kat_sr_pkt += this->alpha;

		temp_jumper = rotozoomSurface(temp_jumper, 0, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -this->alpha*180.0/PI, 1, true);
	}

	//po padku na plecy
	else if(this->landed && (this->fell_down==1 || this->fell_down==3) )
	{
		temp_jumper = this->icon_jumper_fall1;
		temp_ski = this->icon_skis_run;

		j_kat_sr_pkt += this->alpha;
		s_kat_sr_pkt += ogame.narty_po_upadku.alpha;

		temp_jumper = rotozoomSurface(temp_jumper, -this->alpha*180.0/PI, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -ogame.narty_po_upadku.alpha*180.0/PI,
		1, true);
	}

	//po upadku do przodu
	else if(this->landed && this->fell_down==2)
	{
		temp_jumper = this->icon_jumper_fall2;
		temp_ski = this->icon_skis_run;

		j_kat_sr_pkt += this->alpha;
		s_kat_sr_pkt += ogame.narty_po_upadku.alpha;

		temp_jumper = rotozoomSurface(temp_jumper, -this->alpha*180.0/PI, 1, 1);
		temp_ski = rotozoomSurface(temp_ski, -ogame.narty_po_upadku.alpha*180.0/PI,
		1, true);
	}

	//liczenie nowych wsporzednych, w ktorych znalazl sie pkt.obrotu po obrocie
	double x_offset = cos(j_kat_sr_pkt) * j_odleg_srodek_punkt + temp_jumper->w/2;
	double y_offset = sin(j_kat_sr_pkt) * j_odleg_srodek_punkt + temp_jumper->h/2;
	//wyswietlenie odpowiednio przesunietego skoczka
	ogame.apply_surface(x - (int)x_offset, y - (int)y_offset, temp_jumper,
	ogame.screen);

	//liczenie nowych wsporzednych, w ktorych znalazl sie pkt.obrotu po obrocie
	x_offset = cos(s_kat_sr_pkt) * s_odleg_srodek_punkt + temp_ski->w/2;
	y_offset = sin(s_kat_sr_pkt) * s_odleg_srodek_punkt + temp_ski->h/2;
```

```
        if(this->fell_down){
                x = ogame.narty_po_upadku.position.x - cam.position.x;
                y = ogame.narty_po_upadku.position.y - cam.position.y;
        }
        //wyswietlenie odpowiednio przesunietego skoczka
        ogame.apply_surface(x - (int)x_offset, y - (int)y_offset, temp_ski,ogame.screen);

        SDL_FreeSurface(temp_jumper);
        SDL_FreeSurface(temp_ski);
}
```

- Procedurą odpowiadającą za symulację skoków oddawanych przez komputer jest metoda *ai_handle()* klasy **game**.
  Imituje ona skoki narciarzy na trzech różnych poziomach zaawansowania.
  Ich skoki różnią się pod względem momentu wybicia i zachowania w locie.
  Im bliżej końca rozbiegu się wybiją, tym lepiej. Im słabszy skoczek, tym wcześniej się wybija. Punkt wybicia jest losowany z określonych dla każdego przedziałów.
  W powietrzu:
  Skoczek słaby przyjmuje stałą, nienajlepszą pozycję, ale wystarczającą aby bezpiecznie wylądować.
  Skoczek średni przyjmuje również stałą pozycję, lecz lepszą pod względem aerodynamiki.
  Skoczek najlepszy na bieżąco dostosowuje swoje nachylenie do aktualnego kierunku lotu w celu maksymalizacji siły nośnej. Podobnie jak pozostali, pod koniec lotu przyjmuje powoli pozycję właściwą do lądowania.
  Długości skoków oddawanych na poszczególnych poziomach przez komputer wahają się w poniższych granicach:
  Na skoczni "Gawronowo" –
  łatwy: 40 – 50 m
  średni: 65 – 75 m
  trudny: 85 – 95 m
  Na skoczni "Jurowo" –
  łatwy: 70 - 80 m
  średni: 90 - 100 m
  trudny: 140 - 150 m

```
void game::ai_handle()
{
        int level_zawodnika = 0;
        int i = 0;
        for (list<gracz>::iterator it=lista_zawodnikow.begin(); it !=
        lista_zawodnikow.end(); ++it){
                i++;
                if(i == ogame.id_zawodnika) level_zawodnika = (*it).level;
        }

        //losowość w momencie wybicia
        int rndm = rand() %10;

        //zroznicowane momenty wybicia, osobno dla kazdej skoczni
        if(this->green_light)
        {
                if(C.on_bar)
                {
                        C.on_bar = false;
                        C.time = SDL_GetTicks();
                }

                if(ogame.skocznia->name=="Gawronowo")
```

```
		{
			if(!C.landed && C.position.x >= (ogame.skocznia->rozbieg_width -90 -
			12 +level_zawodnika*30 +rndm) ) { C.jump(); }
		}
		else if(ogame.skocznia->name=="Jurowo")
		{
			if(!C.landed && C.position.x >= (ogame.skocznia->rozbieg_width -150
			-12 +level_zawodnika*50 +rndm) ) { C.jump(); }
		}
	}
	//miejsce, od ktorego skoczek ma zacząć przyjmowac pozycje do ladowania
	int start_landing_from=0;
	if(ogame.skocznia->name=="Gawronowo")
	{
		switch (level_zawodnika)
		{
			case 1: start_landing_from = 250; break;
			case 2: start_landing_from = 650; break;
			case 3: start_landing_from = 1000;break;
		}
	}
	else if(ogame.skocznia->name=="Jurowo")
	{
		switch (level_zawodnika)
		{
			case 1: start_landing_from = 970; break;
			case 2: start_landing_from = 1300;break;
			case 3: start_landing_from = 1700;break;
		}
	}
	//sterowanie skoczkiem przez komputer w locie
	if(C.flies && (C.x_precise < ogame.skocznia->rozbieg_width + start_landing_from)
)
	{
		//zawodnik 1 levelu leci stale tym samym początkowym kątem
		//zawodnik 2 levelu zmienia kąt po wybiciu i dalej leci tak samo
		//zawodnik 3 levelu na bieżąco dostosowuje kąt lotu
		if (level_zawodnika == 2)
		{
			C.jumper_alpha = -PI/9;
			C.skis_alpha = 0;
		}
		else if (level_zawodnika == 3)
		{
			if ( abs(C.jumper_alpha - (ogame.skocznia->gradients[C.position.x]-
			PI/5)) > PI/1000 )
			{
				if( abs(C.jumper_alpha-C.skis_alpha) > MIN_ANGLE_VARIANCE)
					C.jumper_alpha += PI/1000;
			}
			if ( C.skis_alpha - (ogame.skocznia->gradients[C.position.x]-PI/7) >
			PI/1000 )
			{
				C.skis_alpha -= PI/1000;
			}
			else if ( C.skis_alpha - (ogame.skocznia->gradients[C.position.x]-
			PI/7) < -PI/1000 )
			{
				C.skis_alpha += PI/1000;
			}
		}
	}
	//powolne korygowanie kąta skoczka i nart do pozycji idealnej dla lądowania
	if( C.x_precise >= ogame.skocznia->rozbieg_width + start_landing_from )
	{
		//ciało
		if ( 90 - (abs(ogame.skocznia->gradients[C.position.x] - C.jumper_alpha)
		*180/PI) < 0 )
			C.jumper_alpha += 0.3*PI/180;
		else
```

```
                C.jumper_alpha -= 0.3*PI/180;
            //narty
            if ( 20 - (abs(ogame.skocznia->gradients[C.position.x] - C.skis_alpha)
            *180/PI) < 0 )
                C.skis_alpha += 0.3*PI/180;
            else
                C.skis_alpha -= 0.3*PI/180;
      }
}
```

- Procedurą odpowiadającą za wykrywanie kolizji skoczka ze stokiem jest metoda *collision_check(double x, double y)* klasy **game**.
  Stok w grze podzielony jest na obszary nachylone pod różnymi kątami. Każdemu z tych obszarów przyporządkowany jest obiekt klasy **slope** zapisany w liście o nazwie *slopes*. Sprawdzanie kolizji odbywa się poprzez wyciągnięcie z listy informacji o obszarze, nad którym obecnie znajduje się zawodnik, a następnie sprawdzenie czy skoczek styka się z krawędzią stoku. Jeżeli tak jest to następuje kolizja. W przypadku poprawnego ułożenia ciała i nart względem stoku następuje lądowanie, w przeciwnym wypadku skoczek przewraca się do przodu lub do tyłu, a jego narty odczepiają się od nóg i zjeżdżają po stoku. Odległość skoku naliczana jest poprzez dodanie do siebie długości wszystkich krawędzi stoku na lewo od zawodnika. Jeżeli zawodnik się wywróci, wtedy długość wynosi 0. Na koniec sprawdzamy czy odległość uzyskana przez zawodnika jest większa, niż dotychczasowy rekord skoczni; jeżeli tak jest to wyświetlana jest informacja o pobiciu rekordu.

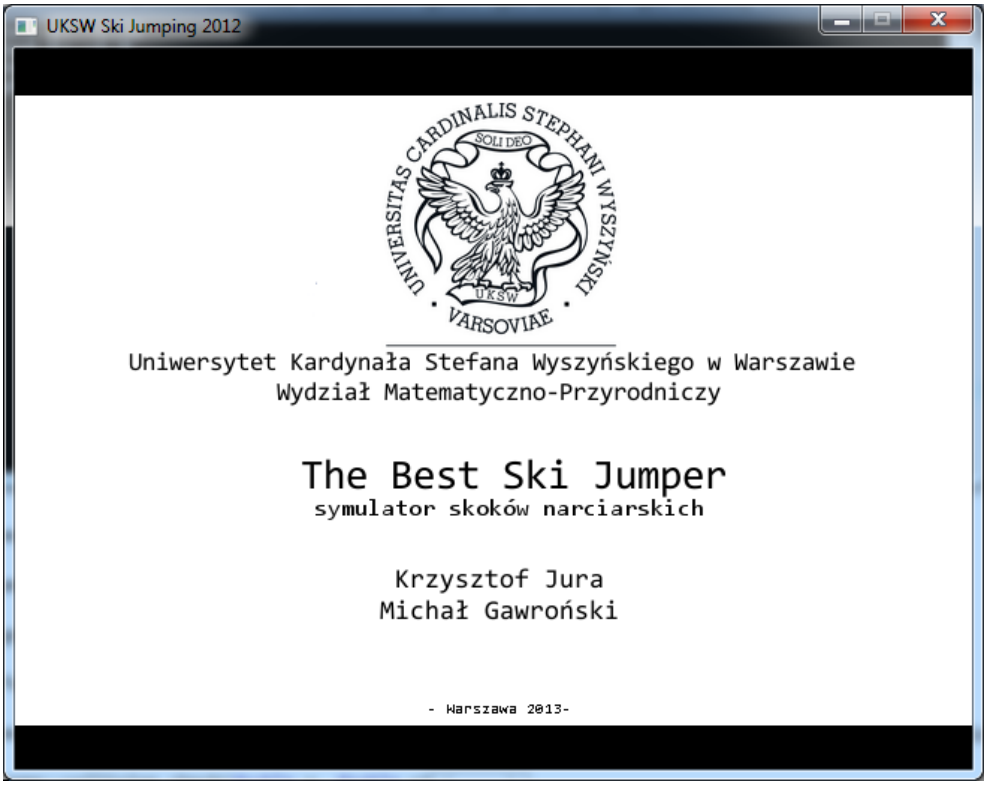
```
//wykrywa kolizje, zwraca 0.0 jeżeli zachodzi kolizja, w przeciwnym wypadku zwraca
współrzędną y stoku
double game::collision_check(double x, double y)
{
      list<slope>::iterator it;
      for(it = ogame.skocznia->slopes.begin(); it!=ogame.skocznia->slopes.end(); it++)
      {
            if( x >= it->position.x && x < it->position.x + it->position.w)
            {
                  //wysokość i szerokość kawałka stoku nad którym znajduje się skoczek
                  double slope_h = it->position.h;
                  double slope_w = it->position.w;

                  double tg = slope_h/slope_w;

                  //współrzędne kawałka stoku nad którym znajduje się skoczek
                  double slope_x = x - it->position.x;
                  double slope_y = it->position.y + (slope_x * tg);
                  //czy współrzędna y zawodnika znajduje się poniżej współrzędnej y
                  stoku
                  if( y >= slope_y )
                  {
                        list<slope>::iterator it2;

                        //obliczamy odległość skoku wzdłuż stoku i przypisujemy do
suma
                        double suma = 0.0;
                        for(it2 = ogame.skocznia->slopes.begin();it2!=ogame.skocznia->
                        slopes.end(); it2++){
                              if( x >= it2->position.x ){
                                    if(x < it2->position.x + it2->position.w){
                                          suma+= sqrt(pow((C.x_precise - it2->
                                          position.x)/METER_TO_PIXEL,2.0)+pow((C.posi
                                          tion.y - it2->
                                          position.y)/METER_TO_PIXEL,2.0));
                                    }else{
                                          suma+= sqrt(pow(it2->
```

```
                                        position.h/METER_TO_PIXEL,2.0)+pow(it2->
                                        position.w/METER_TO_PIXEL,2.0));
                                }
                        }

                }
                C.jump_length = suma;

                C.flies = false;
                C.landed = true;
                if( abs(this->skocznia->gradients[C.position.x]*180/PI -
                C.jumper_alpha*180/PI) < MIN_JUMPER_ANGLE){
                        C.fell_down=2;
                        //Skok nieustany! Zawodnik sie przewrocil do przodu!
                }else if( abs(this->skocznia->gradients[C.position.x]*180/PI -
                C.jumper_alpha*180/PI) > MAX_JUMPER_ANGLE){
                        C.fell_down=1;
                        //Skok nieustany! Zawodnik sie przewrocil do tylu!
                }else if( abs(this->skocznia->gradients[C.position.x]*180/PI -
                C.skis_alpha*180/PI) < MIN_SKIS_ANGLE || abs(this->skocznia-
                >gradients[C.position.x]*180/PI - C.skis_alpha*180/PI) >
                MAX_SKIS_ANGLE){
                        if( abs(this->skocznia->gradients[C.position.x]*180/PI -
                        C.jumper_alpha*180/PI) < 90) C.fell_down=2;
                        else C.fell_down=1;
                        //Skok nieustany! Zawodnik sie przewrocil - zły kąt
                        nart!
                }
                //jezeli zawodnik się przewrócił to automatycznie odczepiają
                się narty i traktujemy je jako odzielny obiekt
                if(C.fell_down){
                        ogame.FRICTION_FACTOR = 0.8;
                        ogame.zawodnik->ogledlosci_skokow[ogame.seria_skokow-1]
                        = 0.0;

                        ogame.narty_po_upadku.position.x = C.position.x;
                        ogame.narty_po_upadku.position.y = C.position.y;
                        ogame.narty_po_upadku.x_precise = C.x_precise;
                        ogame.narty_po_upadku.y_precise = C.y_precise;
                        ogame.narty_po_upadku.v = C.v;
                }else{
                        ogame.FRICTION_FACTOR = 0.06;
                        ogame.zawodnik->ogledlosci_skokow[ogame.seria_skokow-1]
                        = C.jump_length;
                }
                //zaokraglamy odleglosc skoku do 2 miejsc po przecinku i
                sprawdzamy czy jest większa od dotychczasowego rekordu skoczni
                double odleglosc_skoku = (int)(ogame.zawodnik->
                ogledlosci_skokow[ogame.seria_skokow-1]*100)/100.0;
                if(!ogame.zawodnik->is_computer && odleglosc_skoku >
                ogame.skocznia->rekord){
                        ogame.rekord_skoczni_type = 1;
                        ogame.skocznia->rekord = odleglosc_skoku;
                        ogame.skocznia->rekordzista = ogame.zawodnik->name;
                }
                ogame.zawodnik->wynik += odleglosc_skoku ;

                return slope_y;
        }

    }
  }
  return 0.0;
}
```

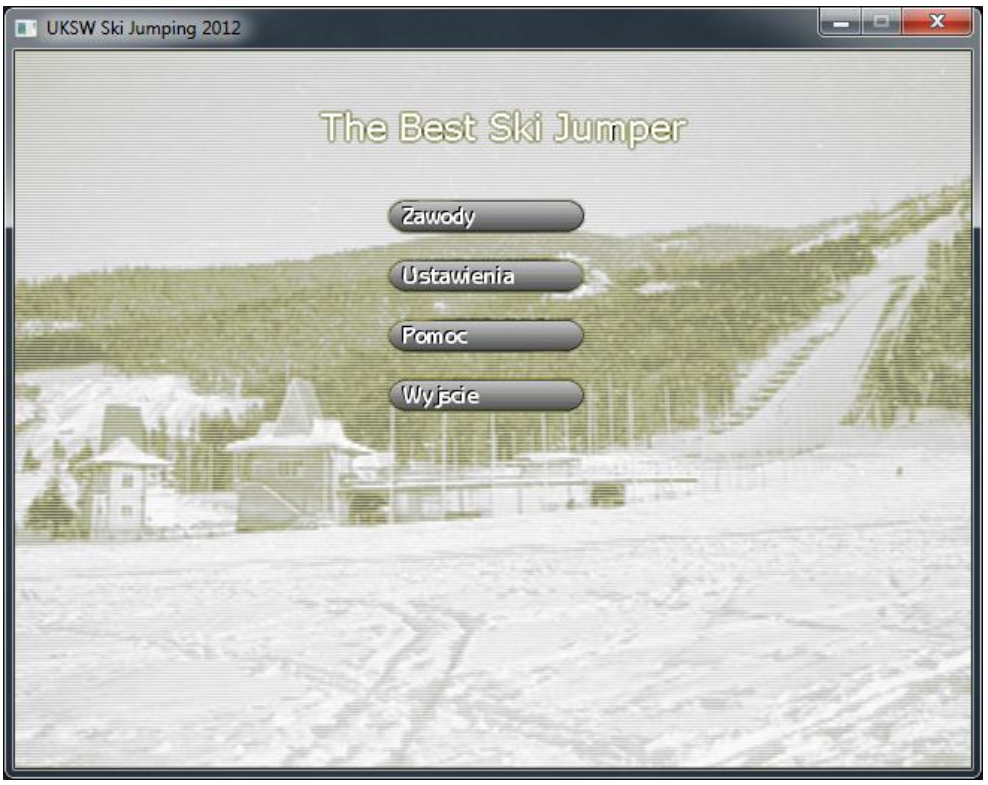
4. Instrukcja obsługi

4.1 Menu

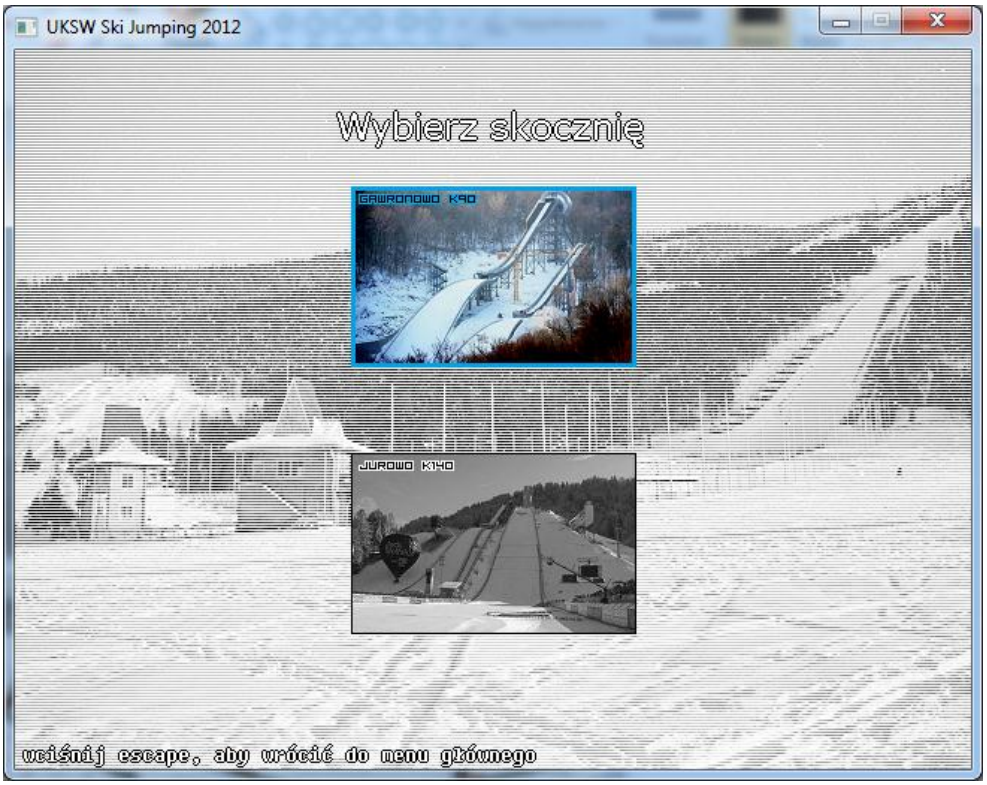
- Ekran powitalny



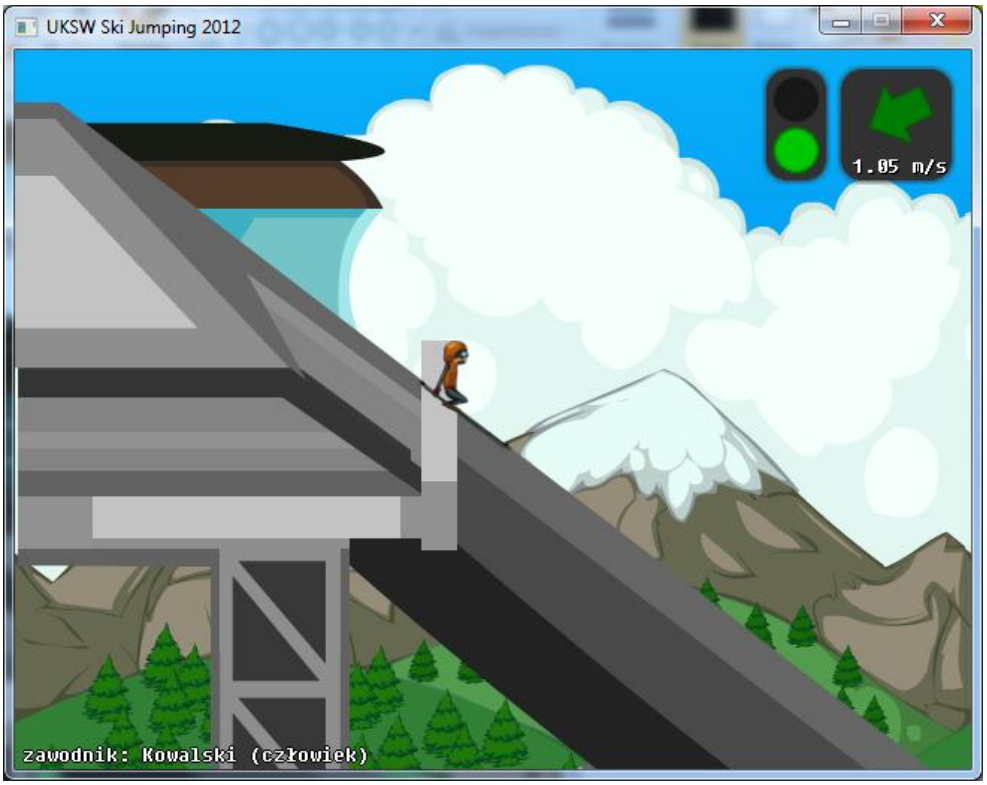
- Menu główne



- Zawody
    - przejście do menu wyboru jednej z dwóch skoczni (K-90 lub K-140)



    - po wybraniu odpowiadającej nam opcji przejście do właściwego konkursu



- Ustawienia
  - zmiana trybu wyświetlania: okienko lub pełny ekran
  - edycja listy zawodników biorących udział w zawodach (max.10):
    - możliwość dodawania skoczków sterowanych przez człowieka lub przez komputer,
    - zmiana nazw zawodników (3-8 liter)
    - ustalanie poziomu komputerowych graczy (łatwy, średni, trudny),
    - usuwanie graczy z listy (kwadratem z X, po prawej stronie)

- Pomoc
  - pomoc, dokładny opis sterowania wraz z rysunkami

- Wyjście
  - wyjście z gry

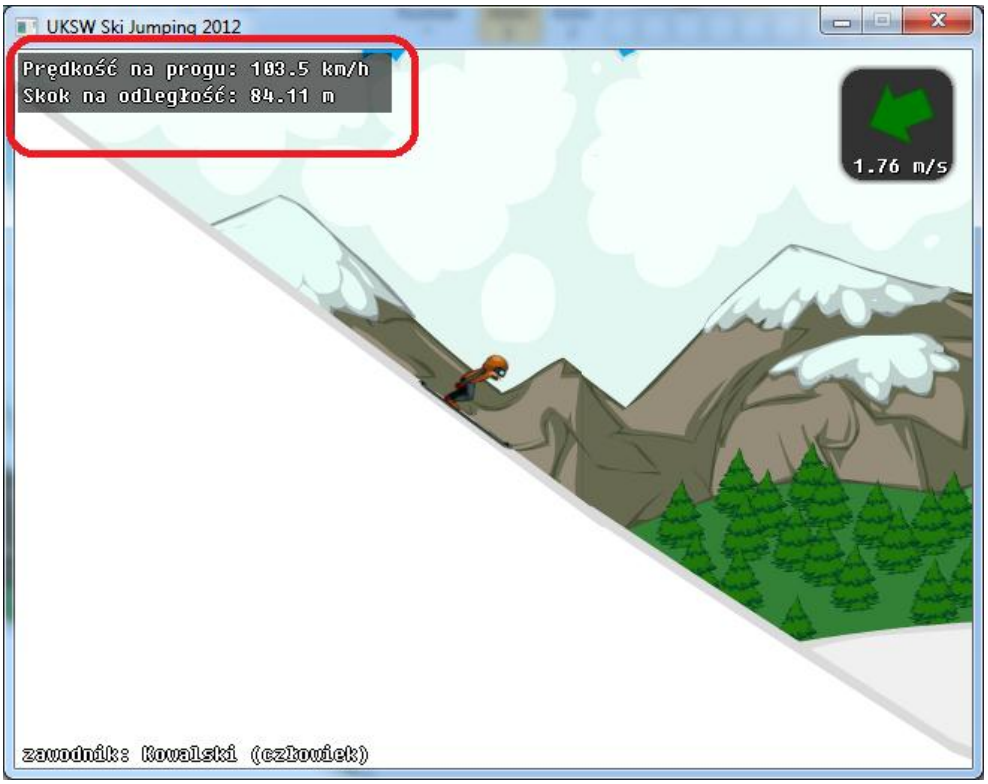
4.2 HUD (head-up display)

Na ekranie poza skocznią i skoczkiem wyświetlane są także informacje pomocnicze, tj.:

- sygnalizator z czerwonym i zielonym światłem, wskazujący możliwość rozpoczęcia skoku siedzącemu na belce zawodnikowi,
- strzałka pokazująca kierunek wiatru oraz jego prędkość w m/s,
- nazwa gracza aktualnie oddającego skok,

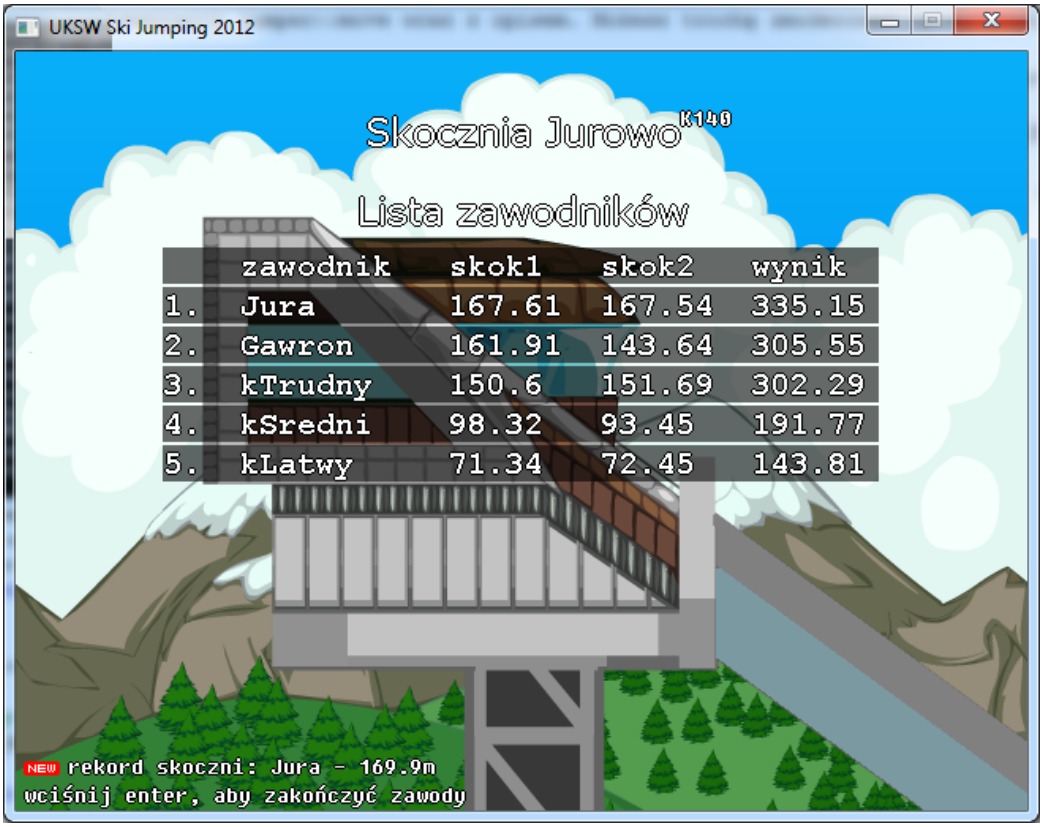
- prędkość uzyskana w momencie wybicia z progu w km/h,

- długość oddanego skoku w metrach,



- informacja o pobiciu rekordu skoczni (nie licząc wyników komputera),

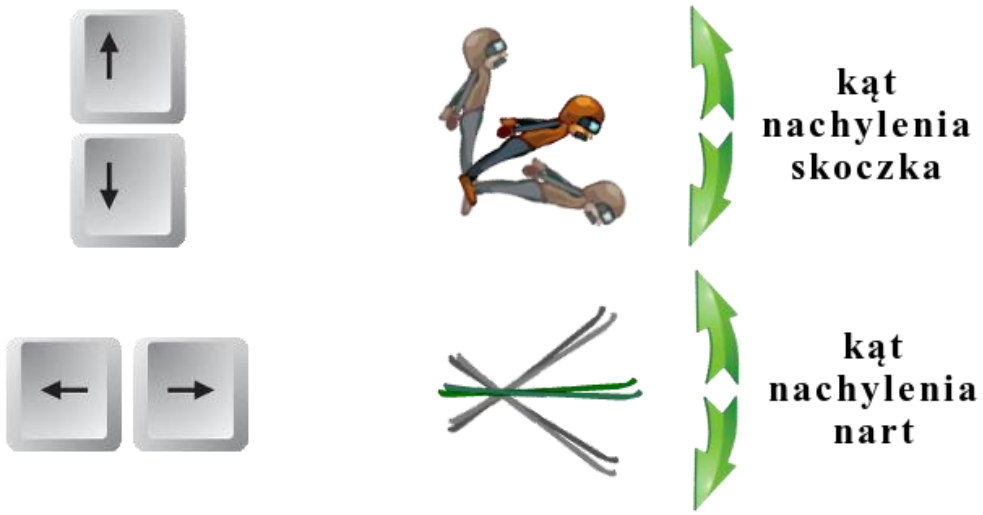
- obecny rekord wybranej skoczni (nie licząc wyników komputera),
- tabela wyników końcowych, pojawia się po turnieju



4.3 Sterowanie

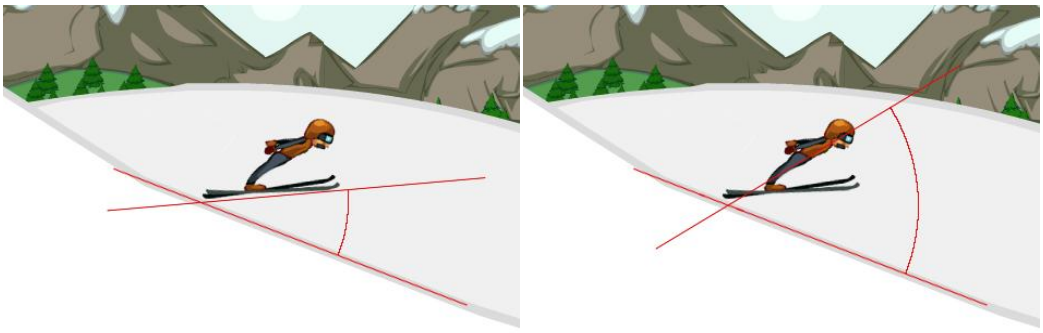
- Odepchnięcie się od belki startowej, po którym nastąpi zjazd po rozbiegu następuje po naciśnięciu klawisza spacji, możliwe wówczas gdy na sygnalizatorze w górnym prawym rogu będzie zapalone zielone światło.
- Wybicie się z progu jest możliwe dopiero w końcowej fazie rozbiegu za pomocą wciśnięcia klawisza spacji.

- Podczas skoku mamy możliwość manipulacji następującymi kątami:
  - nachylenia skoczka do podłoża (strzałki góra i dół)
  - nachylenia nart do podłoża (strzałki lewo i prawo)



W ostatniej fazie lotu przygotowujemy skoczka do lądowania za pomocą ustawienia odpowiedniego kąta nachylenia nart i skoczka do stoku. Widełki dla nart to od -10^{O} do 50^{O}, a dla skoczka od 60^{O} do 110^{O}.

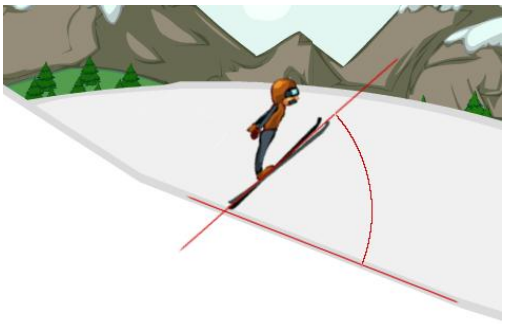
- przykład prawidłowego lądowania; zaznaczone kąty nachylenia nart i skoczka:

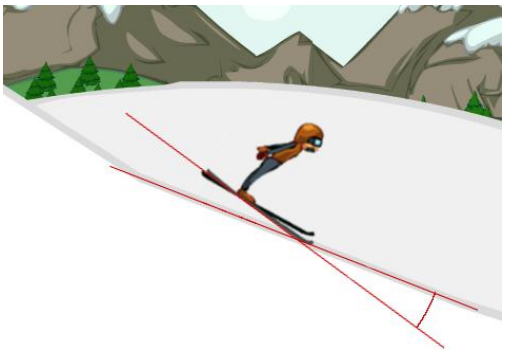


- sylwetka po prawidłowym wylądowaniu:

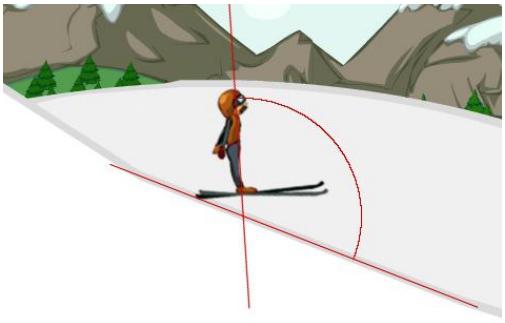
- cztery przykłady nieprawidłowego lądowania:

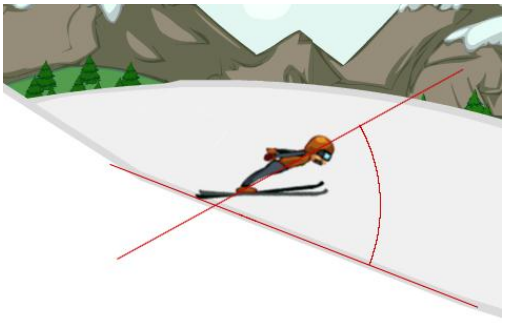
za duże odchylenie nart (> 50^{O}), za małe odchylenie nart (< -10^{O}),



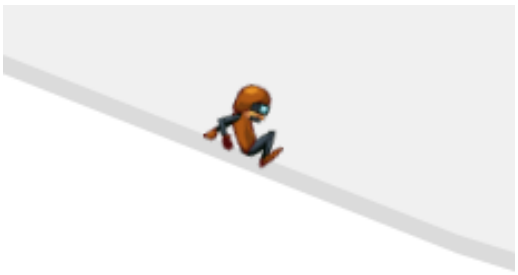


za duże odchylenie tułowia (> 110^{O}), za małe odchylenie tułowia (< 60^{O})

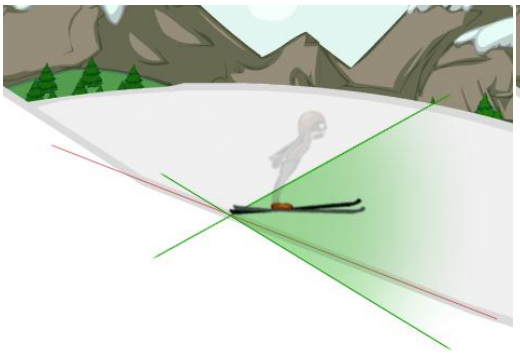


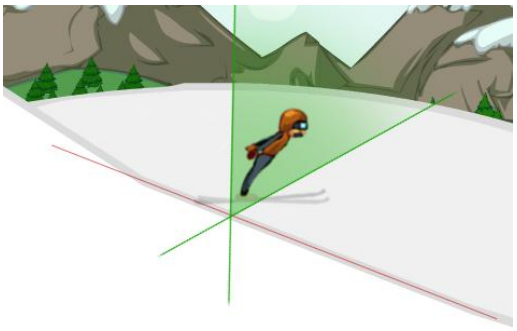


- sylwetka po nieprawidłowym wylądowaniu (upadek do tyłu i do przodu):



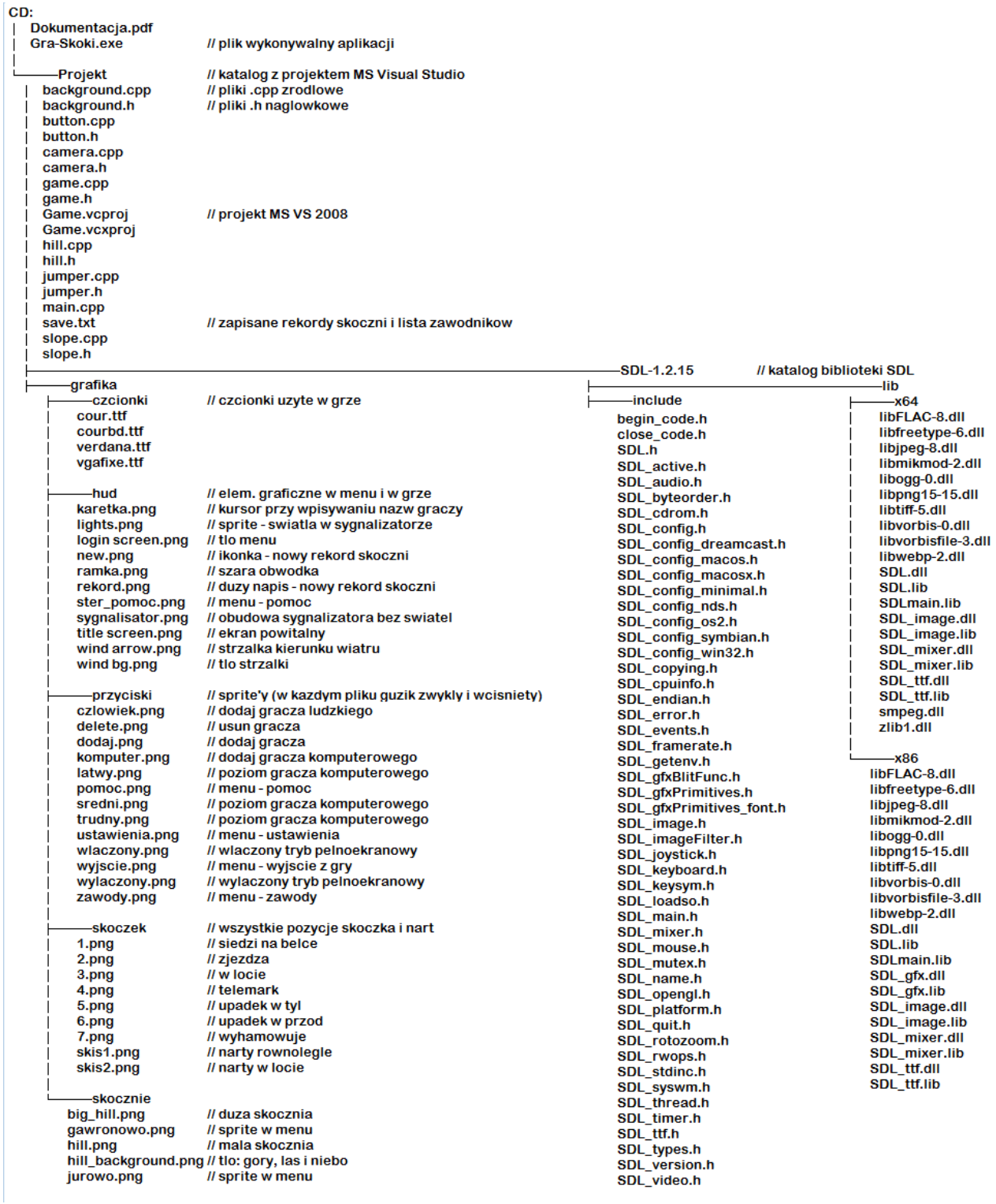
- granice prawidłowych kątów nachylenia nart i tułowia (na zielono):





5. Zawartość CD

- plik wykonywalny .exe
- dokumentacja w formacie .pdf (ten dokument)
- drzewo katalogów i plików:



6. Bibliografia

[1] M. Kozielski, “Fizyka dla szkół średnich” wyd.V, Warszawa 1999

[2] B. Stroustrup, “Język C++”

[3] Kurs SDL - http://www.sdltutorials.com/

7. Oświadczenie

Oświadczamy, że niniejszy projekt oraz dokumentację wykonaliśmy samodzielnie i wyrażamy zgodę na wykorzystanie ich do celów dydaktycznych.

Krzysztof Jura

.....

Michał Gawroński

.....