Wydział Elektrotechniki, Automatyki i Informatyki
Politechnika Świętokrzyska

Studia: **Stacjonarne 1 Stopnia**
Kierunek: **Informatyka**
Przedmiot: **Programowanie w języku C2**

Politechnika Świętokrzyska
Kielce University of Technology

DOKUMENTACJA PROJEKTU
TETRIS

Autorzy:

- **Adrian Chmielowiec – 50%**

- Mateusz Kubas – 50%

Grupa: **2ID13B**

Spis treści

1. Wstęp

1.1. Cel

Celem projektu jest stworzenie popularniej i znanej wszystkim gry TETRIS stworzonej przez Aleksieja Pażytnowa i jego współpracowników 6 czerwca 1984r i wzbogacenie go o ranking graczy.

1.2. Zakres

W zakres projektu wchodzą głównie rzeczy związane z samą rozgrywką gry. Pierwszym etapem jest utworzenie prostokątnej planszy zwanej tetrionem, która ma wymiary zwykle 20 wierszy na 10 kolumn. Następnym krokiem jest stworzenie kolorowych klocków na planszy, które złożone są z czterech małych kwadratów nazywanych też blokami. Klocki te (określa się je mianem „tetrimino") mają przemieszczać się w kierunku dolnej krawędzi w miarę możliwości. Kiedy jeden z nich opadnie na samo dno, zostaje unieruchomiony a na planszy u góry ukazuje się kolejny klocek, który również porusza się w dół. Gra ma trwać do momentu, w którym klocek nie będzie mógł się pojawić na planszy. Zadaniem gra ma być układanie tetrimino na planszy poprzez przesuwanie klocków w poziomie i wykorzystywania rotacji w taki sposób aby utworzyć wiersz z klocków na całej szerokości prostokąta. Gdy tak się stanie wiersz zostaje usunięty, a pozostałe klocki opadają w dół, tworząc przestrzeń dla nowych elementów.

Występują 3 wersję algorytmów, które stosuje się w tej grze do obsługi spadania elementów po usunięciu wiersza. Pierwszy z nich to oryginalny algorytm gry w którym po usunięciu wiersza leżące ponad nim elementy spadają o poziom w dół ale nie dalej, nawet jeśli jakaś kolumna elementów mogłaby spaść dalej. Drugą wersja, najbardziej popularna jest to algorytm będący modyfikacją oryginalnego. W tej wersji po usunięciu wiersza kolumny elementów nad nim przesuwają się w dół, najniżej jak to jest możliwe, czyli może wystąpić sytuacja w której to usuniemy kilka wierszy. Ostatnią 3 wersją jest algorytm najmniej podobny do oryginału. Polega on na tym że usunięcie

wiersza pociąga za sobą reorganizację zarówno elementów nad, jak i pod nim, czyli że wszystkie elementy przesuwają się w dół planszy tak że wypełniają wszystkie luki powstałe pomiędzy nimi. Pozwala to również na usuwaniu kilku wierszy jednocześnie.

Zestaw klocków do tetrisa czyli tetrimino został szczegółowo opisany. Zestaw ten składa się z siedmiu różnych klocków złożonych z kwadratów.

- Tetrimino „I” – cztery elementy w jednym szeregu
- Tetrimino „T” – trzy elementy w rzędzie i jeden dołączony do środkowego elementu
- Tetrimino „O” – cztery elementy połączone w kwadrat
- Tetrimino „L” – trzy elementy w rzędzie i jeden dołączony do lewego elementu od spodu
- Tetrimino „J” – trzy elementy w rzędzie i jeden dołączony do prawego elementu od spodu
- Tetrimino „S” – tetrimino „O” po przesunięciu dwóch górnych elementów w prawo
- Tetrimino „Z” – tetrimino „O” po przesunięciu dwóch górnych elementów w lewo

Pary „L” i „J” oraz „S” i „Z” przedstawiają lustrzane odbicia, jednak nie można poprzez obrót jednego utworzyć drugiego.

1.3. Koncepcje

Naszą koncepcją na tą grę było utworzenie oryginalnego Tetrisa wzbogaconego o ranking oraz możliwość zapauzowania gry w dowolnej momencie i dodania okna ukazującego jaki klocek jako następny znajdzie się na planszy. Za każdy usunięty wiersz gracz ma dostawać punkty, który po zakończeniu gry mają zostać zapisywane do pliku tekstowego wraz z dowolną dla użytkownika nazwą. Za usuniecie 4 wierszy czyli zrobienie „Tetrisa” dostaniemy dodatkowe pkt. Ranking można będzie wyświetlić w dowolnym momencie. Rozpoczęcie gry będzie odbywało się poprzez naciśniecie przycisku. Pauza podczas gry możliwa

będzie do włączenia w menu jak i skrótu. Kształt klocków ma być losowany spośród możliwych stworzonych wcześniej elementów. Klocki będą posiadać różne kolory, które będą umilać rozgrywkę. Punkty zbierane przez gracza będą cały czas wyświetlane w bocznym panelu i aktualizowane na bieżąco podczas rozgrywki. Możliwe również ma być upuszczenie klocka przy pomocy skrótu klawiszowego oraz przyspieszania spadania. Na górnym panelu zostaną umieszczone informację o autorach i skrótach klawiszowych.

2. Specyfika projektu

W naszym projekcie znajdują się wszystkie funkcjonalności związane z oryginalnymi założeniami gry TETRIS. Jest to między innymi możliwość sterowania naszym klockiem po ukazaniu się na planszy, możemy wykonywać rotację w dowolną stronę i przesuwać go w poziomie co umożliwia ustawienie tetrimino w taki sposób by stworzyły wypełnienie wiersza planszy.

Inną z funkcjonalności jest tworzenie różnorodnych kształtów tetrimino na podstawie kwadratów. Nasze klocki są również kolorowane w celu umilenia rozgrywki. Sposób tworzenia kształtów jak i wszystkie możliwe zostały opisane powyżej.

Kolejną z funkcjonalności naszej gry jest opadanie klocków. Jest to jedna z ważniejszych funkcji ponieważ bez niej nie byłoby możliwe odbycia gry. Cała funkcja polega na tym że przy pomocy licznika rysujemy nasz klocek na planszy co pewien odliczony czas coraz niżej aż do końca tetrionu. Gdy dojdziemy na sam koniec lub uderzymy w któryś z klocków na planszy to tetrimino zostaje tam zatrzymany i zostanie umieszczony nowy klocek na planszy. Funkcja ta spełnia założenia występujące w pierwszej oryginalnej wersji algorytmu czyli po usunięciu wiersza leżące ponad nim elementy spadają o poziom w dół ale nie dalej, nawet jeśli jakaś kolumna elementów mogłaby spaść dalej.

Kolejną funkcją, która odpowiada za grę jest kasowanie wypełnionych wierszy po ułożeniu. Wykrywa ona czy użytkownik zapełnił wszystkie wolne miejsca i jeśli tak usuwa cały wiersz zwalniając miejsce, które potrzebne jest do dalszej gry.

Kolejną z funkcjonalności gry jest wykrycie momentu w którym nasze wiersze utworzone z klocków nie pozwolą na utworzenie nowego. Taka sytuacja oznacza koniec gry i jest sygnalizowana poprzez napis jak i prośbę o wpisaniu nazwy użytkownika w celu zapisania wyniku do rankingu.

Innymi funkcjonalności naszej gry to na przykład punktowanie użytkownika i wyświetlanie na bieżąco wyniku w bocznym panelu. Punkty naliczane są w momencie ułożenia przez użytkownika z klocków całego wiersza i jego usunięcia przez grę.

Dodatkową funkcjonalnością jest również wyświetlanie następnego klocka jaki pojawi się na planszy, ułatwia to w pewnym stopniu rozgrywkę.

Sama grę możemy rozpocząć poprzez przycisk Start. Uruchamia on funkcje odpowiadające za start gry czyli pojawienia się klocka w panelu gry. Inną z opcji jaką dodaliśmy jest możliwość zapauzowania gry. Uruchamiana jest tak jak start poprzez naciśniecie odpowiedniego przycisku co skutkuje wywołaniem funkcji. Możliwe jest również w menu przejrzenia rankingu graczy, w których znajdują się zapisane wyniki gier wraz z nazwami użytkowników, którzy dokonali tego. Odpowiada za to odpowiedni przycisk, który wywołuje funkcje tak jak w poprzednich funkcjonalnościach.

W momencie gdy chcemy wyłączyć naszego Tetrisa wystarczy kliknąć na przycisk „wyjście” a on zamknie nasz program.

Kolejną funkcjonalnością jest możliwość przyspieszenia spadania klocka przy pomocy skrótu klawiszowego.

Ostatnią z funkcjonalności jest możliwość natychmiastowego upuszczenia naszego klocka przy pomocy spacji do której przypisane jest wywołanie funkcji odpowiadającej za upuszczenie.

W grze są również zaprogramowane skróty klawiszowe dzięki którym możemy: pauzować, wyłączyć, upuścić klocek lub przyspieszyć.

Na górze naszej aplikacji znajduje się menu w którym możemy znaleźć informacje o autorach oraz dostępnych skrótach klawiszowych. Po kliknięciu na jedna z opcji zostanie wyświetlone okno z informacją.

WxWidgets jest biblioteką graficzną, dzięki której można graficznie tworzyć różne programy dlatego zdecydowaliśmy się jej użyć by stworzyć Tetrisa.

Używanie biblioteki wxWidgets podczas budowy Tetrisa wiązało się z pewnymi ograniczeniami samej biblioteki.

By umożliwić jakikolwiek ruch należało użyć wxTimer w celu stworzenia cyklów gry.

Jako że biblioteka wxWidgets w której tworzymy naszą grę nie posiada wbudowanej funkcji tworzącej kwadraty to musimy sami je stworzyć przy pomocy linii i dopiero tak stworzone kwadraty można było ukształtować w jedno z tetrimino.

W naszej wersji gry kształty poruszają kwadrat po kwadracie a nie piksel po pikselu.

Przed rozpoczęciem gry należało zainicjować kilka ważnych zmiennych takich jak to że nasz klocek nie spadł czy liczbę wierszy usuniętych.

Rysowanie jest podzielone na dwa etapy. W pierwszym kroku rysowane są wszystkie kształty, które zostają zapamiętane w tablicy do której dostęp uzyskujemy poprzez funkcję. Następnym krokiem jest narysowanie elementu, który spada.

W wxWidgets aby obsłużyć jakiekolwiek zdarzenie należy najpierw je stworzyć i przekazać do funkcji. Aby możliwe było używanie klawiatury należy ją zainicjować. Biblioteka posiada wbudowane kody klawiszy. Odpowiednia funkcja służy do sprawdzania jaki z klawiszy nacisnęliśmy i z jej pomocą wywołujemy odpowiednie zdarzenia po wciśnięciu klawisza.

Tworzenie nowego klocka lub jego opadanie realizowane jest za pomocą jednej metody, która sprawdza czy poprzedni znalazł się na dole, jeśli nie to przesuwamy spadający kawałek o jedną linię w dół a jeśli tak to tworzymy nowy.

Metoda pozwalająca na upuszczenie klocka działa tak że wykonuje funkcję, która przesuwa nasz klocek do momentu aż nie zwróci false.

Funkcja odpowiadająca za ustalenie ostatniej pozycji klocka realizowana jest przy pomocą współrzędnych, w tej samej funkcji sprawdzane jest również czy mamy co najmniej jedną pełną linię zapełnioną.

Usuwanie wierszy odbywa się poprzez sprawdzenie czy mamy pełną linie, jeśli tak zwiększamy licznik i przenosimy wszystkie linie powyżej pełnego w dół i tym sposobem niszczymy całą linie. Po zniszczeniu całej linii uruchamiana jest funkcja która zmienia stan punktów. Za usuniecie 1 linii otrzymamy 100 pkt, za usunięcie 4 czyli zrobienia Tetrisa dostaniemy 800. Każde użycie spacji da nam dodatkowe punkty. Punktacja zrealizowana jest za pomocą prostych zmiennych i funkcji.

Koniec gry został zaimplementowany w taki sposób że gra kończy się w momencie gdy nasza figura nie może wrócić do swojej początkowej pozycji.

Metoda odpowiadająca za ruch sprawdza najpierw czy nie znajdujemy się na krawędzi planszy lub nie przylegamy do innego kształtu, jeśli nie to umieszczany jest nasz opadający klocek w nowej pozycji i zwracamy wartość true.

Obracanie klocków jest obsługiwane przez tablice współrzędnych w którym znajdują się współrzędne reprezentujące kształty, kiedy zwracamy aktualny kawałek, który spada zwracamy jest współrzędne pozycji i następnie patrzymy na tabelę współrzędnych i rysujemy.

Informacje o kształcie są zapisywane w klasie.

Kolory naszych klocków są trzymany w tablicy w której dany kolor jest przypisany do kształtu.

Dodatkowa funkcjonalność związana z pauzowaniem gry zrealizowana jest tak że po prostu zatrzymujemy nasz timer i wyświetlamy napis informujący nas że gra jest wstrzymana. By użyć naszej pauzy wystarczy kliknąć w przycisk lub użyć skrótu klawiszowego co poskutkuje wywołaniem funkcji odpowiadającej za zatrzymanie timera, jeśli chcemy wznowić grę wystarczy że znowu użyjemy tak jak poprzednio przycisku lub skrótu.

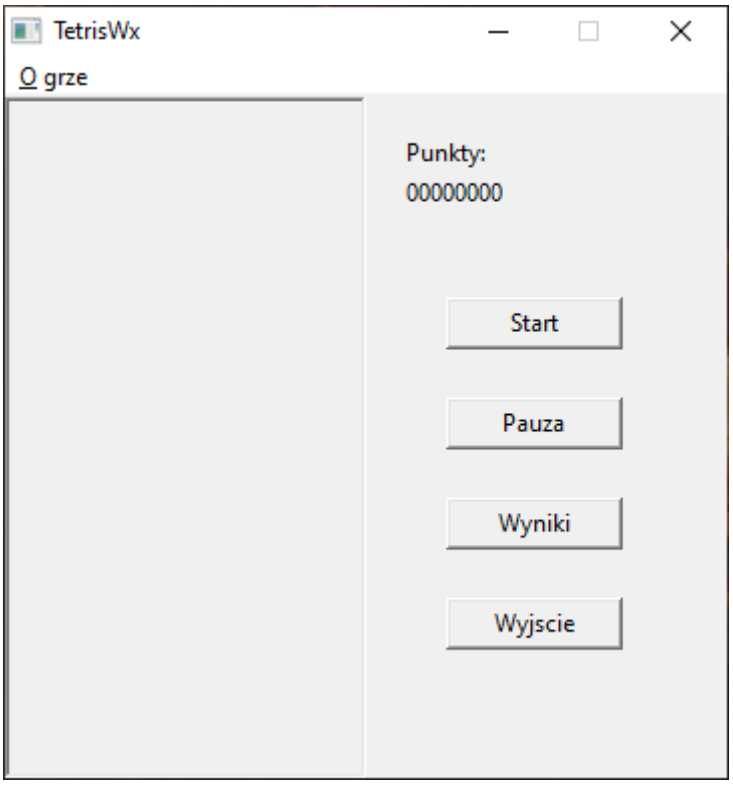
Wyłączanie naszej gry to po prostu wywołanie wbudowanej funkcji do kończenia programu. Po kliknięciu przycisku lub użycia skrótu jest ona realizowana.

Wyświetlanie tabeli wyników odbywa się poprzez użycia przycisku w menu gry. Wciśniecie go wywołuje funkcję, która sprawdza czy znajdują się dane jakieś w niej. Jeśli nie wywołany jest komunikat. Jeśli tak to ukazuje nam się okno dialogowe. Całość zrealizowana jest za pomocą mapy, która wczytuje dane z pliku wraz z startem gry. Została użyta mapa, ponieważ wpisanie tej samej nazwy użytkownika nie skutkuje wtedy utworzeniem jego kopii a zaktualizowaniem wyniku. Sortowanie odbywa się przy pomocy funkcji sortowania mapy. Mapa jest kopiowana do multimapy i ustawiany jest klucz. Dzięki temu kluczem multimapy są punkty, do których można przypisać wiele graczy, gdy ktoś miał tyle samo punktów. Metoda truncate() ucina multimapę po sortowaniu tak, żeby nie była dłuższa niż 10. Sortowanie jest rosnące. Użytkownik proszony jest o wpisanie swojej nazwy gracza po zakończeniu gry, jeśli jego wynik znalazł się wśród 10 najlepszych. Wyniki wraz z nazwą zapisywane są do pliku i z niego odczytywane.

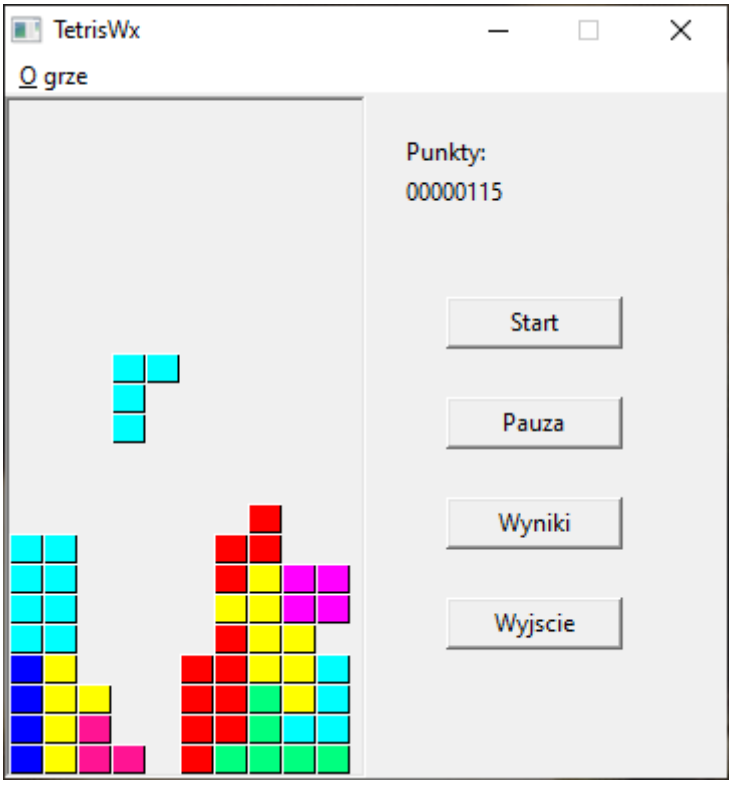
Innym rozwiązaniami dla naszej gry jest użycie biblioteki, która obsługuje akceleracją sprzętową grafiki przy użyciu OpenGL. Przykładem takiej biblioteki jest sfml. Ułatwiłaby ona w znacznym stopniu implementacji mechaniki gry i upiększyła naszą grę.

3. Techniczna implementacja

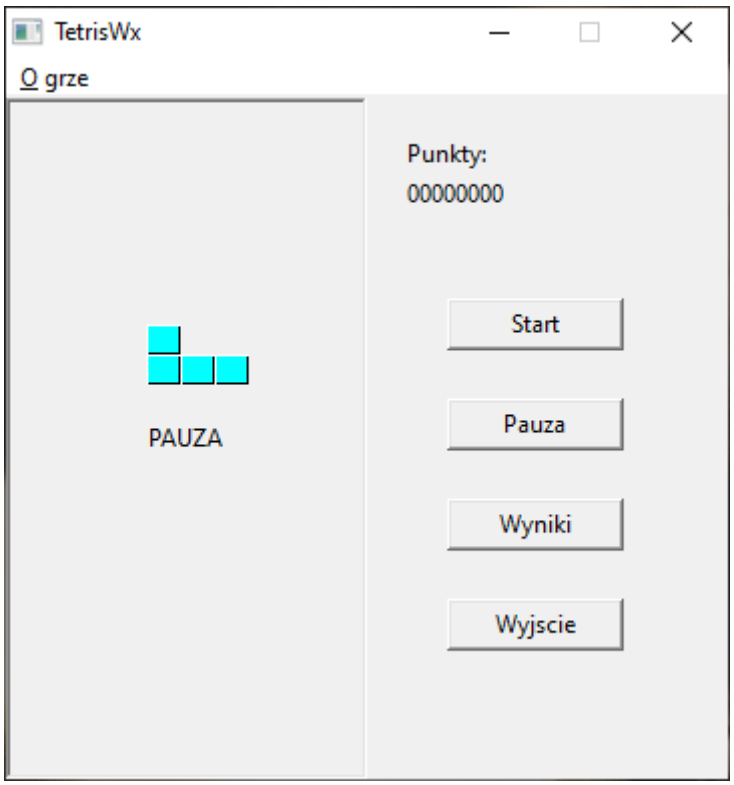
Okno gry po uruchomieniu:



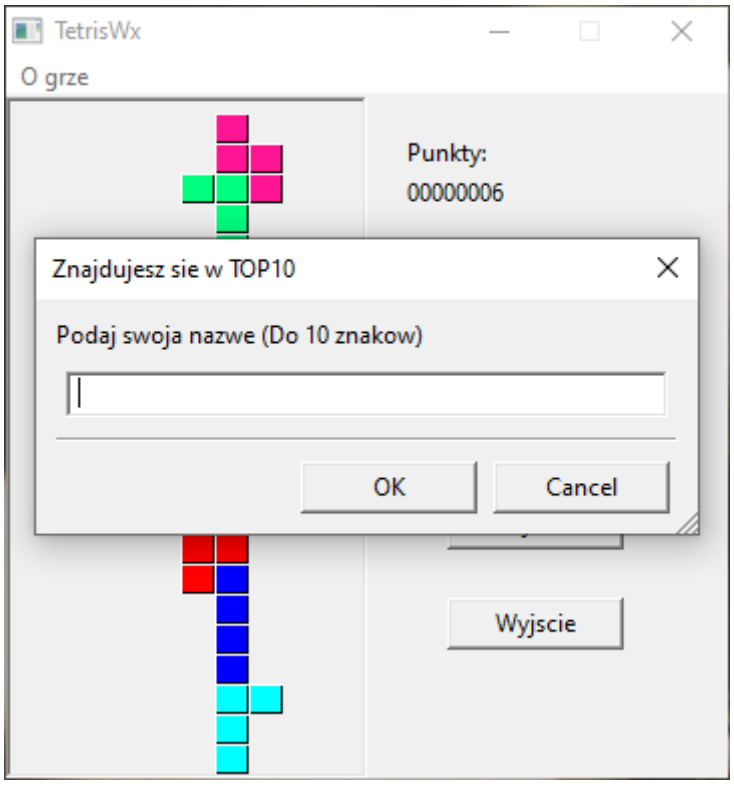
Zrzut przedstawiający rozgrywkę po naciśnięciu przycisku start:



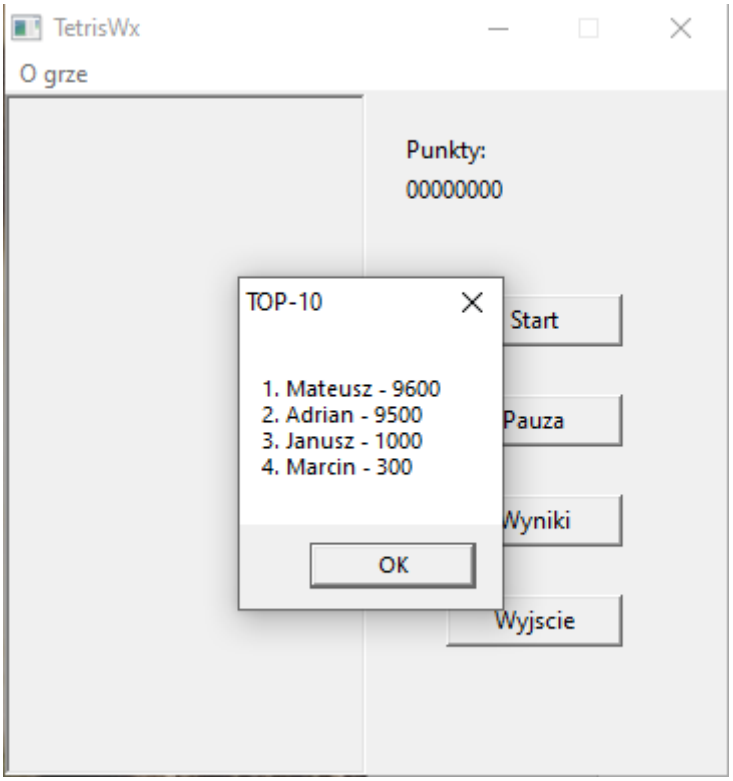
Zrzut przedstawiający grę po wciśnięciu pauzy:



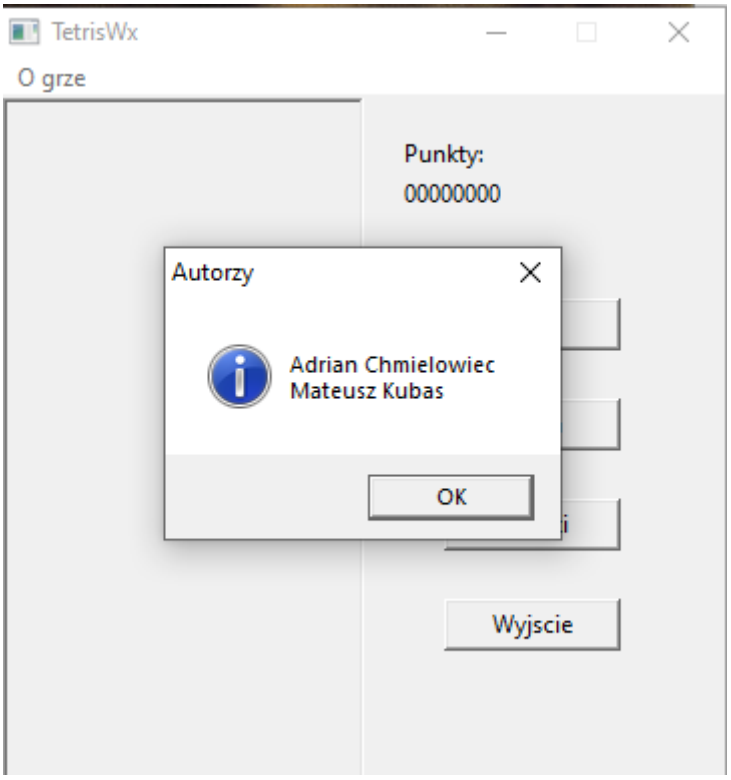
Zrzut przedstawiający okno gry po przegranej:



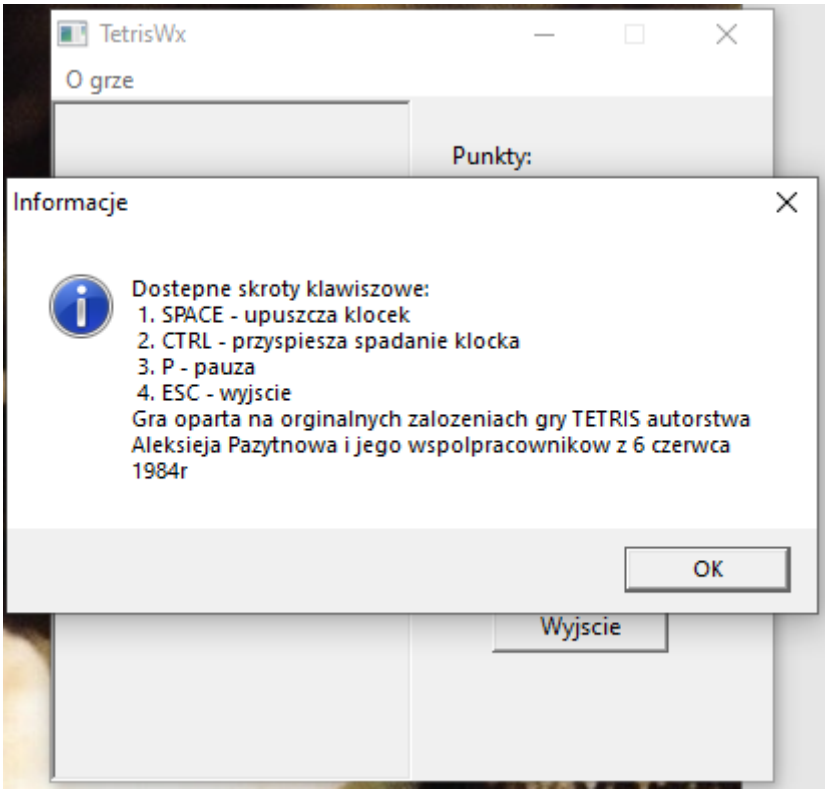
Zrzut przedstawiający okno aplikacji po naciśnięciu przycisku wyniki:



Zrzut przedstawiający okno programu po wybraniu opcji autorzy z górnego menu:



Zrzut przedstawiający okno programu po wybraniu opcji informacje z górnego menu:



3.1.1. Pliki cpp

Main.cpp

```
1. #include <wx/wx.h>
2. #include "mainframe.h"
3.
4. class TetrisApp : public wxApp
5. {
6. public:
7.     virtual bool OnInit();
8. };
9.
10.  IMPLEMENT_APP(TetrisApp)
11.
12.  bool TetrisApp::OnInit()
13.  {
14.      srand(time(NULL));
15.
16.      MainFrame *mainframe = new MainFrame("TetrisWx");
   //Utworzenie glownego okna aplikacji i ustawienie tytulu
```

```
17.
18.       mainframe->Centre();                                    //Sprawia
   ze aplikacja pojawia sie na srodku ekranu
19.       mainframe->Show(true);                                  //Sprawia
   ze aplikazja wogole sie pojawia na ekranie
20.       return true;
21.   }
```

Mainframe.cpp

```
1. #include "mainframe.h"
2.
3. MainFrame::MainFrame(const wxString& title) : wxFrame(NULL, wxID_ANY,
   title, wxDefaultPosition, wxDefaultSize, wxMINIMIZE_BOX |
   wxSYSTEM_MENU | wxCAPTION | wxCLOSE_BOX | wxCLIP_CHILDREN)
4. {
5.     wxBoxSizer *maingrid = new wxBoxSizer(wxHORIZONTAL);
   //Tworzenie BoxSizer w ktorym zostana umieszczone panele gry i
   punktacji
6.     SetSizer(maingrid);
   //Ustawianie utworzonego BoxSizer jako sizera.
7.
8.     scorepanel = new ScorePanel(this);
   //Utworzenie i wskazanie panelu punktacji.
9.     tetrispanel = new TetrisPanel(this, scorepanel);
   //Utworzenie i wskazanie panelu gry, oraz przekazanie utworzonemu
   obiektowi obiektu scorepanel.
10.
11.       maingrid->Add(tetrispanel);
   //Dodaje panel z gra do sizera.
12.       maingrid->Add(scorepanel);
   //Dodaje panel z punktacja do sizera.
13.
14.       //-O grze-//
15.       wxMenu *menuMenu = new wxMenu;
   //tworzy opcje w menu
16.       menuMenu->Append(1, "Autorzy");
   //dodaje opcje do panela menu
17.       menuMenu->Append(2, "Informacje");
   //dodaje opcje do panela menu
18.
19.       wxMenuBar *menu = new wxMenuBar;
   //tworzy gorny panel menu
20.       menu->Append(menuMenu, "&O grze");
   //dodaje do gornego panela
21.
22.       SetMenuBar(menu);
   //ustawia menu
23.       //przylaczanie opcji z menu do wykonywania odpowiednich akcji
24.
   Connect(1,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&TetrisP
   anel::Autorzy); //wyswietlenie autorow
25.
   Connect(2,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&TetrisP
   anel::Info);   //informacje o grze
26.       //-------//
27.
28.       //Tworzy przyciski odpowiadajace za Start, Pauze, Wyniki i
   wyjcie z gry i umieszcza je w odpowiednich miejscach.--------------//
```

```
29.      wxButton *btn1 = new wxButton(scorepanel, ID_START,
   wxT("Start"),wxPoint(40,100), wxDefaultSize, 0, wxDefaultValidator);
30.      wxButton *btn2 = new wxButton(scorepanel, ID_PAUZA,
   wxT("Pauza"),wxPoint(40,150), wxDefaultSize, 0, wxDefaultValidator);
31.      wxButton *btn3 = new wxButton(scorepanel, ID_WYNIKI,
   wxT("Wyniki"),wxPoint(40,200), wxDefaultSize, 0, wxDefaultValidator);
32.      wxButton *btn4 = new wxButton(scorepanel, ID_EXIT,
   wxT("Wyjscie"),wxPoint(40,250), wxDefaultSize, 0,
   wxDefaultValidator);
33.      //-------------------------------------------------------------
   ------------------------------------------------------------//
34.
35.      //Przylaczenie przyciskow do wykonywania odpowiednich akcji.---
   ------------------------------------------------------------//
36.      Connect(ID_START,
   wxEVT_COMMAND_BUTTON_CLICKED,wxCommandEventHandler(TStart));
   //Start/Restart gry.
37.      Connect(ID_PAUZA,
   wxEVT_COMMAND_BUTTON_CLICKED,wxCommandEventHandler(TPause));
   //Pauza.
38.      Connect(ID_WYNIKI,
   wxEVT_COMMAND_BUTTON_CLICKED,wxCommandEventHandler(TTop));
   //Tabela wynikow.
39.      Connect(ID_EXIT,
   wxEVT_COMMAND_BUTTON_CLICKED,wxCommandEventHandler(TExit));
   //Wyjscie z gry.
40.      //-------------------------------------------------------------
   ------------------------------------------------------------//
41.
42.      maingrid->Fit(this);                  //Dopasosywanie rozmiaru
   siatki do pol.
43.      tetrispanel->SetFocus();              //Ustawienie skupienia na
   panel z gra.
44.  }
```

Scoreboard.cpp

```
1. #include "scoreboard.h"
2. #include <wx/file.h>
3. #include <wx/textfile.h>
4.
5. void scoreboard::saveScore(wxString username, double newscore)
6. {
7.     wxFile file;                          //Tworzy zmienna file
   odpowiadajaca za plik z wynikami.
8.     /*
9.     Uzylem wxFile zamiast wxTextFile poniewaz ten pierwszy udostepnia
10.      funkcje tworzenia pliku z flaga Overwrite, co pozwala mi na
11.      tworzenie calego pliku od poczatku za kazdym razem, dzieki
   czemu
12.      nie musze go czyscic, lub nadpisywac specyficznej linijki, a
   moge
13.      od razu zapisac cala tabele umieszczona w zmiennej scoreMap w
   pliku scoreboard.h.
14.      */
15.
16.      scoreMap[username] = newscore;       //Przypisywanie punktow do
   nazwy urzytkownika za pomoca mapy zdefniowanej w scoreboard.h.
17.      file.Create("wyniki", true);         //Tworzenie pliku i
   jednoczesne go otwarcie.
```

```
18.      sortBoard();                          //Sortowanie tablicy wedlug
   punktacji.
19.
20.      wxString scorestr;                    //String ten sluzy do
   konwersji liczby typu double na ciag wxString.
21.      for (auto& it : sorted_Mmap)                    //Petla
   zapisujaca wartosci mapy do pliku.
22.      {
23.          scorestr = "";                            //"Czyszczenie"
   ciagu scorestr, by nastepna przypisana do niej wartosc nie dopisala
   sie do poprzedniej.
24.
25.          scorestr.sprintf("%.0f", it.first);    //Konwersja typu
   double na wxString za pomoca metody sprintf().
26.                                                    //Matoda ta jest
   uzyta by poprzez %.0f bozbyc sie zapisywania wartosci double
27.                                                    //w notacji
   naukowej, co robily inne metody.
28.
29.          file.Write(it.second);                     //Wpisanie do
   pliku nazwy uzytkownika.
30.          file.Write("\n");                         //Wstawienie nowej
   linijki.
31.          file.Write(scorestr);                     //Wpisanie do pliku
   punktow uzytkownika.
32.          file.Write("\n");
33.      }
34.
35.
36.      file.Close();                                 //Zamykanie pliku
37.
38.      return;
39.  }
40.
41.  ////////////////////////////////////////////
42.  ////////////////////////////////////////////
43.  ////////////////////////////////////////////
44.
45.  void scoreboard::sortBoard()
46.  {
47.
48.      sorted_Mmap.erase(sorted_Mmap.begin(), sorted_Mmap.end());
   //Czyszczenie dotychczasowej zawartosci mmultimapy.
49.
50.      for(auto& it : scoreMap)
   //Przepisywanie wartosci z mapy do multimapy, gdzie wartosc pierwsza
   w mapie wchodzi do multimapy jako druga,
51.      {
   //a wartosc druga jako pierwsza.
52.          sorted_Mmap.insert({it.second, it.first});
53.      }
54.
55.      truncate();
56.
57.      return;
58.  }
59.
60.  ////////////////////////////////////////////
61.  ////////////////////////////////////////////
62.  ////////////////////////////////////////////
63.
```

```
64.  bool scoreboard::readScoreboardFromFile()
65.  {
66.
67.      wxTextFile file("wyniki");              //Tworzenie zmiennej
   file o typie wxTextFile pozwalajacym na czytanie pliku linijka po
   linijce.
68.
69.      if (!file.Exists())                     //Sprawdza czy plik
   istnieje.
70.          return false;
71.
72.      file.Open();                            //Otwarcie przypisanego
   wyzej pliku.
73.
74.      if(!file.IsOpened())                    //Sprawdzanie, czy plik
   zostal poprawnie otworzony.
75.          return false;
76.
77.
78.      wxString filestr, name;                 //wxString filestr
   sluzy do zapisywania w niej kazdej kolejnej linijki pliku.
79.                                              //wxString name sluzy
   do zapisania w niej nazwy uzytkownika ze zmiennej filestr.
80.                                              //"name" jest
   potrzebne, poniewaz zmienna filestr przechodzi na nastepna linie w
   pliku zanim dane
81.                                              //zaczna zapisywac sie
   do mapy.
82.
83.      double dscore;                          //double dscore sluzy
   do konwersji punktow uzytkownika z typu wxString.
84.
85.      for(filestr = file.GetFirstLine(); !file.Eof(); filestr =
   file.GetNextLine())       //Petla iterujaca przez caly plik.
86.      {
87.
88.          name = filestr;                     //Pierwszy, oraz co
   drugi wiersz w pliku, to nazwa uzytkownika.
89.          filestr = file.GetNextLine();       //Odczytanie nastepnej
   linii, ktora musi byc wynikiem punktowym danego uzytkownika.
90.          filestr.ToDouble(&dscore);          //konwersja typu
   wxString na double.
91.          scoreMap[name] = dscore;            //Przypisanie wartosci
   punktowej do uzytkownika w mapie.
92.
93.          //Poniewaz w srodku petli zostala wykonana metoda
   GetNextLine(), wskaznik linijki pliku zostal przesuniety, czyli
94.          //nastepne wywolanie petli ponownie natrafi na nazwe
   uzytkownika.
95.      }
96.      file.Close();                           //Zamykanie pliku.
97.      return true;                            //Funkcja bool zwraca
   true po poprawnie wykonanej operacji.
98.  }
99.
100. bool scoreboard::isTop(double score)        //sprawdzanie czy
   znajdujemy sie w top
101. {
102.     if ( sorted_Mmap.size() < 10)
103.         return true;
104.
```

```
105.     auto it = sorted_Mmap.begin();
106.     if (it->first >= score)
107.         return false;
108.     return true;
109. }
110.
111. void scoreboard::truncate()          //ustawianie rozmiaru
112. {
113.     auto rozm = sorted_Mmap.size();
114.     if (rozm <= 10)
115.         return;
116.
117.     auto it = sorted_Mmap.begin();
118.     advance(it, rozm - 10);
119.     sorted_Mmap.erase(sorted_Mmap.begin(), it);
120. }
```

Scorepanel.cpp

```
1. #include "scorepanel.h"
2. #include "mainframe.h"
3.
4. ScorePanel::ScorePanel(wxFrame *parent) : wxPanel(parent, 0, 0, 180,
   340, wxBORDER_NONE)                //Tworzony panel ma wymiary
   180x340px.
5. {
6.     wxStaticText *text = new wxStaticText(this, wxID_ANY, "Punkty:",
   wxPoint(20, 20), wxDefaultSize);    //Tworzy tekst mowiacy "Punkty:"
   i umieszcza go w odpowiednim miejscu w panelu.
7.     scoreText = new wxStaticText(this, wxID_ANY, "00000000",
   wxPoint(20, 40), wxDefaultSize);            //Tworzy tekst z
   punktacja aktualizowana podczas gry i umieszcza go na panelu.
8.
9. }
10.
11.  void ScorePanel::displayScore(double score)
12.  {
13.      wxString wynik;                                          //Tworzy
   string "wynik" ktory bedzie uzyty do wyswietlenia punktacji.
14.      size_t l = wxString::FromDouble(score).length();    //Zapisuje
   dlugosc ciagu znakow zajmowanych przez ilosc punktow.
15.
16.      if (l < 8)                                               //Sprawdza
   czy "dlugosc" punktacji nie przekracza 8.
17.      {
18.          for(int i=0; i < 8 - l; i++)                        //Najpierw
   wkleja do wyniku tyle zer, ile brakuje punktom znakow, by osiagnac
   dlugosc 8
19.          {
20.              wynik.append('0');
21.          }
22.      }
23.      wynik.append(wxString::FromDouble(score));          //Dolacza
   do dolaczanych wczesniej zer punkty gracza.
24.      scoreText->SetLabel(wynik);
   //Aktualizuje punktacje w scoreText.
25.  }
```

Shape.cpp

```
1. #include <stdlib.h>
2. #include <algorithm>
3. #include "Shape.h"
4.
5. using namespace std;
6.
7. void Shape::SetShape(Tetrominoes shape)
8. {
9.     static const int coordsTable[8][4][2] = {
10.          { { 0, 0 },   { 0, 0 },   { 0, 0 },   { 0, 0 } },
11.          { { 0, -1 },  { 0, 0 },   { -1, 0 },  { -1, 1 } },
12.          { { 0, -1 },  { 0, 0 },   { 1, 0 },   { 1, 1 } },
13.          { { 0, -1 },  { 0, 0 },   { 0, 1 },   { 0, 2 } },
14.          { { -1, 0 },  { 0, 0 },   { 1, 0 },   { 0, 1 } },
15.          { { 0, 0 },   { 1, 0 },   { 0, 1 },   { 1, 1 } },
16.          { { -1, -1 }, { 0, -1 },  { 0, 0 },   { 0, 1 } },
17.          { { 1, -1 },  { 0, -1 },  { 0, 0 },   { 0, 1 } }
18.      };
   //tablica wspolrzednych kawalkow ksztaltu
19.
20.      for (int i = 0; i < 4 ; i++) {
21.          for (int j = 0; j < 2; ++j)
22.              coords[i][j] = coordsTable[shape][i][j];
   //zapisywanie wspolrzednicyh kawalkow ksztaltu
23.      }
24.      pieceShape = shape;
25.  }
26.
27.
28.  void Shape::SetRandomShape()            //losowanie kawalka
29.  {
30.    int x = rand() % 7 + 1;
31.    SetShape(Tetrominoes(x));
32.  }
33.
34.  //pobieranie wspolrzednych i zapisywanie
35.  int Shape::MinX() const
36.  {
37.    int m = coords[0][0];
38.    for (int i=0; i<4; i++) {
39.        m = min(m, coords[i][0]);
40.    }
41.    return m;
42.  }
43.
44.  int Shape::MaxX() const
45.  {
46.    int m = coords[0][0];
47.    for (int i=0; i<4; i++) {
48.        m = max(m, coords[i][0]);
49.    }
50.    return m;
51.  }
52.
53.  int Shape::MinY() const
54.  {
55.    int m = coords[0][1];
56.    for (int i=0; i<4; i++) {
57.        m = min(m, coords[i][1]);
58.    }
59.    return m;
```

```
60.  }
61.
62.  int Shape::MaxY() const
63.  {
64.    int m = coords[0][1];
65.    for (int i=0; i<4; i++) {
66.        m = max(m, coords[i][1]);
67.    }
68.    return m;
69.  }
70.
71.  //rotacje klockow
72.  Shape Shape::RotateLeft() const
73.  {
74.      if (pieceShape == SquareShape)
75.          return *this;
76.
77.      Shape result;
78.      result.pieceShape = pieceShape;
79.      for (int i = 0; i < 4; ++i) {
80.          result.SetX(i, y(i));
81.          result.SetY(i, -x(i));
82.      }
83.      return result;
84.  }
85.
86.  Shape Shape::RotateRight() const
87.  {
88.      if (pieceShape == SquareShape)
89.          return *this;
90.
91.      Shape result;
92.      result.pieceShape = pieceShape;
93.      for (int i = 0; i < 4; ++i) {
94.          result.SetX(i, -y(i));
95.          result.SetY(i, x(i));
96.      }
97.      return result;
98.  }
```

Tetrispanel.cpp

```
1. #include "tetrispanel.h"
2. #include "scorepanel.h"
3. //-------------------------------------------------------------------
   ----------------------------------------------//
4. //#######################################//  Konstruktor TetrisPanel
   //##############################################//
5. //-------------------------------------------------------------------
   ----------------------------------------------//
6. TetrisPanel::TetrisPanel(wxFrame *parent, ScorePanel *span) :
   wxPanel(parent, 0, 0, 180, 340, wxBORDER_THEME)
7. {
8.      timer = new wxTimer(this, 1);    //Utworzenie timera.
9.
10.      //Inicjalizacja podstawowych zmiennych-//
11.      isFallingFinished = false;
12.      isStarted = false;
13.      isPaused = false;
14.      numLinesRemoved = 0;
```

```
15.       curX = 0;
16.       curY = 0;
17.       sp = span;
18.       sb = new scoreboard();
19.       //-----------------------------------//
20.
21.       pausedText = new wxStaticText(this, wxID_ANY, "PAUZA");
    //Utworzenie napisu PAUZA.
22.       pausedText->Center();
    //Wysrodkowanie go w stosunku do panelu z gra.
23.       pausedText->Hide();
    //Ukrycie tekstu.
24.
25.       ClearBoard();
    //Wyczyszczenie planszy.
26.
27.       //Przypisanie zdarzen do odpowiednich funkcji.----------------
    --------//
28.       Connect(wxEVT_PAINT,
    wxPaintEventHandler(TetrisPanel::OnPaint));
29.       Connect(wxEVT_KEY_DOWN,
    wxKeyEventHandler(TetrisPanel::OnKeyDown));
30.       Connect(wxEVT_TIMER,
    wxCommandEventHandler(TetrisPanel::OnTimer));
31.       //-----------------------------------------------------------
    --------//
32.
33.       if (sb->readScoreboardFromFile())
34.           sb->sortBoard();
35.   }
36.   //-----------------------------------------------------------------
    ------------------------------------------------//
37.   //#################################################################
    ##############################################//
38.   //-----------------------------------------------------------------
    ------------------------------------------------//
39.
40.   using namespace std;
41.
42.   void TetrisPanel::Start()
43.   {
44.       if (isPaused)       //Start nie dziala gdy gra jest spauzowana.
45.           return;
46.
47.       isStarted = true;               //Zadeklarowanie zaczenia gry.
48.       isFallingFinished = false;      //Spadanie klocka nie moze byc
    zakonczone gdy gra dopiero sie zaczyna.
49.       numLinesRemoved = 0;            //Resetowanie licznika
    usunietych linii.
50.       punkty = 0;                     //wyzerowanie punktow
51.       sendScore();
52.       ClearBoard();                   //wyczyszczenie panelu z gra
53.
54.       pausedText->Hide();             //ukrycie napisu
55.       pausedText->SetLabel("PAUZA");
56.
57.       NewPiece();                     //tworzenie nowego obiektu
58.       this->SetFocus();               //wybranie panela z gra
59.       timer->Start(300);              //uruchomienie timera
60.   }
61.
```

```
62.  void TetrisPanel::Pause()
63.  {
64.      this->SetFocus();
65.      if (!isStarted)                  //sprawdzenie czy gra sie
   uruchomila
66.          return;
67.
68.      isPaused = !isPaused;                    //zapauzowanie gry i
   sprawdzanie czy jest juz zapauzowana
69.      if (isPaused) {
70.              pausedText->Show();              //wyswietlenie tekstu
   pauza
71.          timer->Stop();                       //zatrzymanie timera
72.      } else {
73.          pausedText->Hide();                  //ukrycie tekstu
74.          timer->Start(300);                   //wznowienie gry
75.      }
76.      Refresh();
77.  }
78.
79.  void TetrisPanel::showTopT()
80.  {
81.      if(isStarted)
82.          if(!isPaused)                    //zapauzowanie gry
83.              Pause();
84.
85.      if ( sb->isBoardEmpty() )                    //jesli tablica
   wynikow jest puta wysiwetl komunikat
86.      {
87.          wxMessageBox("Brak wynikow");
88.          return;
89.      }
90.
91.      sb->sortBoard();                     //wyswietl tablice wynikow
92.
93.      wxString strtop;
94.      int i = 1;
95.      for(auto it = sb->sorted_Mmap.rbegin(); it != sb-
   >sorted_Mmap.rend(); it++)      //Wyswietlanie mapy od konca
96.      {
97.          wxString buf;
98.          buf.sprintf("%d. %s - %.0f\n",i, it->second, it->first);
   //wswietlanie
99.          strtop << buf;
100.         i++;
101.     }
102.
103.     wxMessageDialog msgD(this, strtop, "TOP-10", wxICON_NONE);
   //utworzenie okna
104.
105.     msgD.ShowModal();   //wyswietlenie utworzonego okna
106. }
107.
108. void TetrisPanel::OnPaint(wxPaintEvent& event) //rysowanie
   ksztaltow
109. {
110.     wxPaintDC dc(this);
111.
112.     wxSize size = GetClientSize(); //pobieranie rozmiaru
113.     int boardTop = size.GetHeight() - BoardHeight * SquareHeight();
114.
```

```
115.
116.     for (int i = 0; i < BoardHeight; ++i) { //rysowanie wszystkich
   ksztaltow
117.         for (int j = 0; j < BoardWidth; ++j) {
118.             Tetrominoes shape = ShapeAt(j, BoardHeight - i - 1);
   //pobieranie kwadratow
119.             if (shape != NoShape)
120.                 DrawSquare(dc, 0 + j * SquareWidth(),
121.                            boardTop + i * SquareHeight(), shape);
   //umieszczanie pod spodem planszy
122.         }
123.     }
124.
125.     if (curPiece.GetShape() != NoShape) { //rysowanie rzeczywistego
   elementu ktory spada
126.         for (int i = 0; i < 4; ++i) {
127.             int x = curX + curPiece.x(i);   //wpisywanie kordow w
   zmienne
128.             int y = curY - curPiece.y(i);
129.             DrawSquare(dc, 0 + x * SquareWidth(),   //rysowanie
130.                        boardTop + (BoardHeight - y - 1) *
   SquareHeight(),
131.                        curPiece.GetShape());    //ustawianie
132.         }
133.     }
134. }
135.
136. void TetrisPanel::OnKeyDown(wxKeyEvent& event)
137. {
138.     if (!isStarted || curPiece.GetShape() == NoShape) {
   //Pominiecie zdarzenia, gdy gra sie nie rozpoczela, lub gracz nie ma
   klocka.
139.         event.Skip();
140.         return;
141.     }
142.
143.     int keycode = event.GetKeyCode();
   //Pobranie wartosci przycisku z klawiatury.
144.     if (keycode == WXK_ESCAPE) {                                //Po
   nacisnieciu klawisza escape nastepuje wylaczenie gry.
145.         exit(true);
146.     }
147.
148.     if (keycode == 80) {                                    //Po
   nacisnieciu klawisza P nastepuje pauza.
149.         Pause();
150.         return;
151.     }
152.     if (isPaused)                                               //Nic
   nie rob, gdy gra jest spauzowana.
153.         return;
154.
155.     switch (keycode) {
156.     case WXK_LEFT:
157.         TryMove(curPiece, curX - 1, curY);
   //Strzalka w lewo probuje przesunac klocek w lewo.
158.         break;
159.     case WXK_RIGHT:
160.         TryMove(curPiece, curX + 1, curY);
   //Strzalka w prawo probuje przesunac klocek w prawo.
161.         break;
```

```
162.     case WXK_DOWN:
163.         TryMove(curPiece.RotateRight(), curX, curY);
   //Strzalka w dol probuje obrocic klocek w prawo.
164.         break;
165.     case WXK_UP:
166.         TryMove(curPiece.RotateLeft(), curX, curY);
   //Strzalka w gore probuje obrocic klocek w lewo.
167.         break;
168.     case WXK_SPACE:
   //Spacja upuszcza klocek na sam dol.
169.         DropDown();
170.         break;
171.     case WXK_CONTROL:
   //Przycisk control przyspiesza opadanie klocka.
172.         OneLineDown();
173.         break;
174.     default:
175.         event.Skip();
   //Ignoruj zdarzenie, gdy wcisniety zostal inny przycisk.
176.     }
177.
178. }
179.
180. void TetrisPanel::OnTimer(wxCommandEvent& event)
181. {
182.     if (isFallingFinished) {        //Jezeli klocek spadnie wez
   nowy klocek.
183.         isFallingFinished = false;
184.         NewPiece();                 //tworzymy nowy klocek
185.     } else {
186.         OneLineDown();              //Klocek jeszcze nie spadl,
   wiec przesunecie linie w dol.
187.     }
188. }
189.
190. void TetrisPanel::ClearBoard()
191. {
192.     for (int i = 0; i < BoardHeight * BoardWidth; ++i)
   //Iterowanie przez cala tablice i ustawianie pustych klockow.
193.         board[i] = NoShape;
194. }
195.
196. void TetrisPanel::DropDown()
197. {
198.     int newY = curY;
199.     while (newY > 0) {
200.         if (!TryMove(curPiece, curX, newY - 1))
   //przesuwanie klocka dopoki jest mozliwosc
201.             break;
202.         --newY;
203.     }
204.     PieceDropped();                                     //wywolanie
   metody zapisujacej ostatnia pozycje klocka
205.     AddToPoints(1);                                     //Gracz
   dostaje 1 pkt. za kazde natychmiastowe upuszczenie klocka.
206.     sendScore();                                        //wysylanie
   punktu
207. }
208.
209. void TetrisPanel::OneLineDown()
210. {
```

```
211.     if (!TryMove(curPiece, curX, curY - 1))              //spadanie
   o 1 linie
212.         PieceDropped();                                  //wywolanie
   metody zapisujacej ostatnia pozycje klocka
213. }
214.
215. void TetrisPanel::PieceDropped()
216. {
217.     for (int i = 0; i < 4; ++i) {
218.         int x = curX + curPiece.x(i);
219.         int y = curY - curPiece.y(i);
220.         ShapeAt(x, y) = curPiece.GetShape();        //pobieranie
   pozycji klocka
221.     }
222.
223.     RemoveFullLines();                                   //usuwamy
   zapelniona linie
224.
225.     if (!isFallingFinished)                              //sprawdzamy
   czy spadl i tworzymy nowy
226.         NewPiece();
227. }
228.
229. void TetrisPanel::RemoveFullLines()
230. {
231.      int numFullLines = 0;
232.
233.      for (int i = BoardHeight - 1; i >= 0; --i) {
234.          bool lineIsFull = true;
235.
236.          for (int j = 0; j < BoardWidth; ++j) {
237.              if (ShapeAt(j, i) == NoShape) {
238.                  lineIsFull = false;
239.                  break;
240.              }
241.          }
242.
243.          if (lineIsFull) {
   //jesli linia jest pelna
244.              ++numFullLines;
   //zwieksz licznik
245.              for (int k = i; k < BoardHeight - 1; ++k) {
   //przeniesc wszystkie linie powyzej pelnego o 1 w dol
246.                  for (int j = 0; j < BoardWidth; ++j)
247.                      ShapeAt(j, k) = ShapeAt(j, k + 1);
248.              }
249.              AddToPoints(100);          //daj punkty za usuniecie
250.              sendScore();
251.          }
252.      }
253.
254.      if (numFullLines > 0) {
255.          numLinesRemoved += numFullLines;
256.          if (numFullLines == 4)
257.          {
258.              AddToPoints(400);
259.              iloscTetris += 1;
260.              sendScore();
261.          }
262.          isFallingFinished = true;
263.          curPiece.SetShape(NoShape);
```

```
264.          Refresh();
265.      }
266.  }
267.
268. void TetrisPanel::NewPiece()                          //metoda
   tworzenia nowego kawalka
269. {
270.     curPiece.SetRandomShape();                   //losowanie
   ksztaltu
271.     curX = BoardWidth / 2 + 1;
272.     curY = BoardHeight - 1 + curPiece.MinY();    //pobieranie
   pozycji
273.
274.     if (!TryMove(curPiece, curX, curY))
275.     {
276.         endGame();      //Gdy klocek nie jest w stanie sie ruszyc
   zaraz po pojawieniu, zakoncz gre.
277.     }
278. }
279.
280. bool TetrisPanel::TryMove(const Shape& newPiece, int newX, int
   newY)            //przesuwanie ksztaltow
281. {
282.     for (int i = 0; i < 4; ++i) {
283.         int x = newX + newPiece.x(i);
284.         int y = newY - newPiece.y(i);
   //pobieranie pozycji i sprawdzanie
285.         if (x < 0 || x >= BoardWidth || y < 0 || y >= BoardHeight)
286.             return false;
   //jesli nasz obiekt znajduje sie na krawedzi planszy
287.         if (ShapeAt(x, y) != NoShape)
   //lub przylega do innego zwracamy false
288.             return false;
   //w przeciwnym razie umieszczamy opadajacy ksztalt w nowej pozycji i
   zwracamy true
289.     }
290.
291.     curPiece = newPiece;
292.     curX = newX;
293.     curY = newY;
294.     Refresh();
295.     return true;
296. }
297.
298. void TetrisPanel::endGame()
299. {
300.     curPiece.SetShape(NoShape);              //Ustawienie uzywanego
   klocka na brak klocka.
301.     timer->Stop();                           //Zatrzymanie timera.
302.     isStarted = false;                       //Deklaracja
   zakonczenia gry.
303.     pausedText->SetLabel("KONIEC GRY");      //Zmiana napisu pauzy
   na KONIEC GRY.
304.     pausedText->Show();                      //Wyswietlenie napisu
   Koniec Gry.
305.
306.     if (!sb->isTop(punkty))
307.         return;
308.
309.     wxTextEntryDialog Dnazwa(this, "Podaj swoja nazwe (Do 10
   znakow)", "Znajdujesz sie w TOP10");
```

```
310.     Dnazwa.SetMaxLength(10);
311.     Dnazwa.ShowModal();
312.
313.     sb->saveScore(Dnazwa.GetValue(), punkty);
314. }
315.
316. void TetrisPanel::DrawSquare(wxPaintDC& dc, int x, int y,
   Tetrominoes shape)
317. {
318.     static wxColour colors[] = { wxColour(0, 0, 0), wxColour(255,
   0, 0),  //kolor klocków
319.             wxColour(255, 20, 147), wxColour(0, 0, 255),
320.             wxColour(255, 255, 0), wxColour(255, 0, 255),
321.             wxColour(0,255, 127), wxColour(0, 255, 255) };
322.
323.     static wxColour light[] = { wxColour(0, 0, 0), wxColour(255,
   255, 255), //obramowanie zewnatrz
324.             wxColour(255, 255, 255), wxColour(255, 255, 255),
325.             wxColour(255, 255, 255), wxColour(255, 255, 255),
326.             wxColour(255, 255, 255), wxColour(255, 255, 255) };
327.
328.     static wxColour dark[] = { wxColour(0, 0, 0), wxColour(0, 0,
   0),  //obramowanie wewnatrz
329.             wxColour(0, 0, 0), wxColour(0, 0, 0),
330.             wxColour(0, 0, 0), wxColour(0, 0, 0),
331.             wxColour(0, 0, 0), wxColour(0, 0, 0) };
332.
333.
334.     wxPen pen(light[int(shape)]);       //rysowanie zewnetrzych
   linii
335.     pen.SetCap(wxCAP_PROJECTING);
336.     dc.SetPen(pen);
337.
338.     dc.DrawLine(x, y + SquareHeight() - 1, x, y);
339.     dc.DrawLine(x, y, x + SquareWidth() - 1, y);
340.
341.     wxPen darkpen(dark[int(shape)]);        //rysowanie
   wewnetrznych linii
342.     darkpen.SetCap(wxCAP_PROJECTING);
343.     dc.SetPen(darkpen);
344.
345.     dc.DrawLine(x + 1, y + SquareHeight() - 1,
346.         x + SquareWidth() - 1, y + SquareHeight() - 1);
347.     dc.DrawLine(x + SquareWidth() - 1,
348.         y + SquareHeight() - 1, x + SquareWidth() - 1, y + 1);
349.
350.     dc.SetPen(*wxTRANSPARENT_PEN);                  //zapelnanie
   kolorem powstalego ksztaltu
351.     dc.SetBrush(wxBrush(colors[int(shape)]));
352.     dc.DrawRectangle(x + 1, y + 1, SquareWidth() - 2,
353.         SquareHeight() - 2);
354. }
355.
356. void TetrisPanel::Autorzy(wxCommandEvent& event)
357. {
358.     char autorzy[100] = "Adrian Chmielowiec \nMateusz Kubas";
359.     wxString msg = autorzy;
360.     wxMessageBox(msg, _("Autorzy"));
361. }
362.
363. void TetrisPanel::Info(wxCommandEvent& event)
```

```
364. {
365.     char informacje[300] = "Dostepne skroty klawiszowe:\n 1. SPACE
   - upuszcza klocek\n 2. CTRL - przyspiesza spadanie klocka\n 3. P -
   pauza\n 4. ESC - wyjscie\nGra oparta na orginalnych zalozeniach gry
   TETRIS autorstwa Aleksieja Pazytnowa i jego wspolpracownikow z 6
   czerwca 1984r";
366.     wxString msg = informacje;
367.     wxMessageBox(msg, _("Informacje"));
368. }
```

3.1.2. Pliki nagłówkowe

Mainframe.h

```
1. #ifndef MAINFRAME_H_INCLUDED
2. #define MAINFRAME_H_INCLUDED
3.
4. #include <wx/wx.h>
5. #include "tetrispanel.h"
6. #include "scorepanel.h"
7.
8. //ID ktore zostana pozniej przydzielone do odpowiednich przyciskow.
9. const int ID_START = 1;
10.  const int ID_WYNIKI = 3;
11.  const int ID_PAUZA = 2;
12.  const int ID_EXIT = 4;
13.  //-----------------------------------------------------------------//
14.
15.  class MainFrame : public wxFrame        //Glowne okno programu.
16.  {
17.  public:
18.      MainFrame(const wxString& title);   //Konstruktor.
19.
20.      TetrisPanel *tetrispanel;           //Wskaznik do obiektu
   odpowiadajacego za panel z gra.
21.      ScorePanel *scorepanel;             //Wskaznik do obiektu
   odpowiadajacego za panel z punktacja.
22.
23.      void TStart(wxCommandEvent & event) {tetrispanel->Start();};
   //Event uruchamiajacy gre po wcisnieciu przypisanego mu przycisku.
24.      void TPause(wxCommandEvent & event) {tetrispanel->Pause();};
   //Event pauzujacy gre po wcisnieciu przypisanego mu przycisku.
25.      void TTop(wxCommandEvent & event) {tetrispanel->showTopT();};
   //Event otwierajacy okno z najlepszymi wynikami.
26.      void TExit(wxCommandEvent & event) {Close(true);};
   //Event zamykajacy okno z gra konczac gre.
27.
28.  };
29.
30.  #endif // MAINFRAME_H_INCLUDED
```

Scoreboard.h

```
1. #ifndef SCOREBOARD_H_INCLUDED
2. #define SCOREBOARD_H_INCLUDED
```

```
3.
4. #include <map>
5. #include <wx/string.h>
6. #include <wx/textfile.h>
7.
8. class scoreboard {
9. public:
10.      std::multimap<double, wxString> sorted_Mmap;        //Multimapa
   przechowujaca posortowana wg. wynikow tabele gotowa do wyswietlenia.
11.      void saveScore(wxString username, double newscore); //Metoda
   zapisujaca informacje znajdujace sie w mapie scoreMap do pliku.
12.      bool readScoreboardFromFile();                      //Metoda
   czytajaca wyniki z pliku i zapisujaca je do mapy scoreMap.
13.      bool isBoardEmpty() {return scoreMap.empty();};
   //sprawdzanie czy tablica wynikow jest pusta
14.      bool isTop(double score);
   //sprawdzanie czy jestesmy w top
15.      void sortBoard();                                   //Metoda
   sorujaca mape wedlug punktacji.
16.      void truncate();
17.
18.  private:
19.      std::map<wxString, double> scoreMap;                //Mapa
   przechowujaca dane o tabeli wynikow.
20.  };
21.
22.  #endif // SCOREBOARD_H_INCLUDED
```

Scorepanel.h

```
1. #ifndef SCOREPANEL_H_INCLUDED
2. #define SCOREPANEL_H_INCLUDED
3.
4. #include <wx/wx.h>
5.
6. class ScorePanel : public wxPanel           //Klasa dzedziczy z
   wxPanel.
7. {
8. public:
9.     ScorePanel(wxFrame *parent);
10.      void displayScore(double score);        //Metoda wyswietlajaca
   przekazana jej liczbe punktow w polu scoreText.
11.
12.  private:
13.      wxStaticText *scoreText;                //Sluzy do wyspietlania
   aktualnej liczby punktow w rozgrywce.
14.  };
15.
16.  #endif // SCOREPANEL_H_INCLUDED
```

Shape.h

```
1. #ifndef SHAPE_H_INCLUDED
2. #define SHAPE_H_INCLUDED
3.
```

```
4. enum Tetrominoes { NoShape, ZShape, SShape, LineShape,
5.                    TShape, SquareShape, LShape, MirroredLShape };
   //typy mozliwych ksztaltow klockow
6.
7. class Shape
8. {
9. public:
10.      Shape() { SetShape(NoShape); }          //konstruktor kawalkow
11.      void SetShape(Tetrominoes shape);       //funkcja uswaiajaca
   kawalki
12.      void SetRandomShape();                  //funkcja losujaca
13.
14.      Tetrominoes GetShape() const { return pieceShape; }
   //funkcja pobierajaca kawalek
15.      int x(int index) const { return coords[index][0]; }
   //pobieranie wspolrzednych
16.      int y(int index) const { return coords[index][1]; }
17.
18.      int MinX() const;       //zmienne do zapisu wspolrzednych
19.      int MaxX() const;
20.      int MinY() const;
21.      int MaxY() const;
22.
23.      Shape RotateLeft() const;       //rotacje klockow
24.      Shape RotateRight() const;
25.
26.  private:
27.      void SetX(int index, int x) { coords[index][0] = x; }
   //ustawianie wspolrzednych
28.      void SetY(int index, int y) { coords[index][1] = y; }
29.      Tetrominoes pieceShape;
   //ustalenie ksztaltu
30.      int coords[4][2];
31.  };
32.
33.  #endif // SHAPE_H_INCLUDED
```

Tetrispanel.h

```
1. #ifndef TETRISPANEL_H_INCLUDED
2. #define TETRISPANEL_H_INCLUDED
3.
4. #include <wx/wx.h>
5. #include "Shape.h"
6. #include "scorepanel.h"
7. #include "scoreboard.h"
8.
9. class TetrisPanel : public wxPanel                        //Klasa
   dziedziczy z wxPanel.
10.  {
11.  public:
12.      TetrisPanel(wxFrame *parent, ScorePanel *span);
   //Konstruktor.
13.      void Start();                                        //Zacznij
   gre
14.      void Pause();                                        //Pauza
15.      void AddToPoints(double value) {punkty += value;};  //Dodawanie
   wartosci double do zmiennej "punkty".
16.      double GetPoints() {return punkty;};
   //Pobieranie wartosci zmiennej "punkty".
```

```
17.      void sendScore() {sp->displayScore(punkty);};
   //Aktualizacja wyswietlanej na panelu obok punktacji.
18.      void endGame();                                    //Kroki
   podejmowane podczas gdy gracz przegra.
19.      void showTopT();                                   //tabela
   wynikow
20.
21.      void Autorzy(wxCommandEvent& event);               //event
   otwierajacy okno z autorami
22.      void Info(wxCommandEvent& event);                  //event
   otwierajaco okno z informacjami
23.
24.      ScorePanel *sp;                                    //Wskaznik
   na panel punktacji wykorzystywany w celu wysyłania tam punktow i
   informacji o nastepnym klocku.
25.      wxStaticText *pausedText;                          //tekst
   pokazujacy sie gdy gra jest spauzowana lub zakonczona.
26.      scoreboard *sb;
   //Utworzenie wskaznika do obiektu tabeli wynikow.
27.
28.  protected:
29.      void OnPaint(wxPaintEvent& event);                 //Event
   wykonujacy sie za kazdym razem, gdy plansza jest rysowana.
30.      void OnKeyDown(wxKeyEvent& event);                 //Event
   wywolujacy sie za kazdym razem, gdy wciskany jest przycisk.
31.      void OnTimer(wxCommandEvent& event);               //Event
   wywolujacy sie z czestotliwoscia timera.
32.
33.
34.  private:
35.      enum { BoardWidth = 10, BoardHeight = 22 };        //Szerokosc
   i wysokosc planszy uzywane potem m.in. do czyszczenia jej.
36.
37.      Tetrominoes & ShapeAt(int x, int y) { return board[(y *
   BoardWidth) + x]; }     //
38.
39.      int SquareWidth() { return GetClientSize().GetWidth() /
   BoardWidth; }           //szerokosc ksztaltow
40.      int SquareHeight() { return GetClientSize().GetHeight() /
   BoardHeight; }        //wysokosc ksztaltow
41.      void ClearBoard();
   //Czyszczenie planszy z jakichkolwiek klockow.
42.      void DropDown();
   //Natychmiastowe upuszczenie klocka na sam dol.
43.      void OneLineDown();
   //przesuwanie linii
44.      void PieceDropped();
   //ustalenie ostatniej pozycji klocka
45.      void RemoveFullLines();
   //usuwanie wierszy
46.      void NewPiece();
   //Wylosowanie i ustawienie nowego klocka.
47.      bool TryMove(const Shape& newPiece, int newX, int newY);
   //poruszanie klockami
48.      void DrawSquare(wxPaintDC &dc, int x, int y, Tetrominoes
   shape);                //rysowanie ksztaltow
49.
50.      wxTimer *timer;             //Timer dzialajacy na okreslonej
   czestotlwosci i poruszajacy klockami.
51.      bool isStarted;             //sprawdzanie czy gra sie zaczela
```

```
52.      bool isPaused;              //sprawdzanie czy gra jest
   zapauzowana
53.      bool isFallingFinished;     //sprawdzanie czy klocek spadl
54.      Shape curPiece;             //pozycja klocka
55.      int curX;                   //wspolrzedna x
56.      int curY;                   //wspolrzedna y
57.      int numLinesRemoved;        //liczba wpelni ulozonych wierszy.
58.      int iloscTetris;            //liczba pokazujaca ile razy udalo
   sie graczowi wykonac zagranie "tetris".
59.      double punkty;              //Punkty zdobyte w rozgrywce.
60.      Tetrominoes board[BoardWidth * BoardHeight];    //tablica
   ksztaltow klockow
61.
62.  };
63.
64.  #endif // TETRISPANEL_H_INCLUDED
```

4. Testowanie projektu

Testy wykonane metodą czarnej skrzynki nie ukazały żadnych problemów z naszą grą. Po włączeniu naszego programu, zachowuje się on zgodnie z naszymi założeniami czyli czeka na naciśniecie klawisza start, próba włączenia pauzy nie przyniesie żadnych rezultatów w tym momencie. Wyświetlanie wyników działa zgodnie z planem. Okno z wynikami zostanie wyświetlone tylko jeśli się one tam znajdują, sortowane są rosnąco, gdy jest puste dostaniemy informację że nie ma żadnych wyników. Nie jest możliwe zapisania kilku graczy o takiej samej nazwie do tabeli wyników. W przypadku wybrania takiej samej nazwy wynik jest aktualizowany dla danego gracza. Wyłączanie gry działa poprawnie, program się zamyka w pełni wraz z procesem odpowiedzialnym za niego. Górne menu również włącza się bez żadnych problemów, możliwe jest wybranie jednej z 2 opcji, w obu wyświetla nam się okno dialogowe z informacjami i możliwe jest zamkniecie ich. Nie możliwe jest bez zamknięcia okna wykonanie innych czynności. Informacja dotycząca punktów jest zawsze dostępna, nie da się nią przesuwać, czy zmienić ilość w jakikolwiek sposób punktów. Po naciśnięciu klawisza start uruchamia się gra, pojawia się 1 klocek, który zaczyna spadać, kolejne naciśniecie klawisza powoduje restart gry. Skróty działają bez problemu. Jeśli zechcemy wyświetlić podczas gry wyniki gra zostanie zapauzowana i będzie oczekiwała na ponowne użycie pauzy w celu dalszej gry. Klikanie start podczas pauzy nie zresetuje nam gry. Spadającymi klockami

możemy manipulować obracając je lub przesuwając w bok, możliwe jest również upuszczenie i przyspieszenie spadania. Nie zauważono żadnych błędów podczas rozgrywki. Na koniec wyskoczy okno dialogowe z prośbą o wpisanie nazwy gdy nasz wynik znajduje się wśród 10 najlepszych, jeśli nie to okno się nie pojawi. Opuszczenie okna bez wpisywania nazwy skutkuje wpisaniem wyniku bez nazwy. Nazwa użytkownika jest ograniczona do 10 znaków i nie ma możliwości napisania ich więcej. Jeśli wpiszemy nazwę pojawi się ona później w wynikach wraz z naszą ilością punktów wskazywanych przez punktacje. Za upuszczenie klocka dostajemy punkty, za usuniecie 1 wiersza 100, za ułożenie 4 dostaniemy 800. Podczas testowania nie znaleźliśmy żadnych błędów. Gra przebiega zgodnie z założeniami.

Przy sprawdzaniu kodu metodą białej skrzynki również nie natknęliśmy się na większe problemy. Jedyną z rzeczy, które nie są planowane to możliwość wpłynięcia na tabele wyników poprzez edycje pliku w którym one się znajdują.

5. Wnioski

Projekt został wykonany w wskazanej przez nas bibliotece wxWidgets i z użyciem języka C++. Został zaimplementowany w pełni mechanizm pozwalający na odbycie gry tak jak zakładaliśmy bez problemów. Możemy wykonywać wszystkie czynności takie jak ruch naszym klockiem, obrót, upuszczenie, przyspieszenie spadania, usuwanie wierszy, tworzenie tetrimino o różnych kolorach i kształtach. Ponad to zostało zaimplementowane proste menu na bocznym panelu, w którym łatwo możemy włączyć grę lub zresetować, pauzować naszą grę, wyświetlić ranking graczy czy wyłączyć. Jest również menu u góry programu, które zawiera informacje o autorach i skrótach klawiszowych dostępnych w grze. Działa również licznik punktów, które są dodawane po usunięciu wiersza. Podczas tworzenia naszej aplikacji natknęliśmy się na wiele trudności i problemów związanych głównie z ograniczeń biblioteki wxWidgets przy tworzeniu takich projektów. Największym problemem było zaprogramowanie ruchu klocka i tworzenie nowych. Niestety ale nie udało nam się zrealizować wyświetlania kolejnego klocka jaki się pojawi w bocznym panelu. Problemem jest to że klocek losowany jest w momencie gdy aktualny

na planszy opadnie. Innym problemem jest przesyłanie elementów pomiędzy dwoma panelami. Podczas testowania naszej gry nie napotkaliśmy się na żadne bugi, które uniemożliwiają grę lub sprawiają że niektóre elementy nie mogą działać poprawnie.

Naszym zdaniem biblioteka wxWidgets jest bardzo dobra do tworzenia różnego typu kalkulatorów, menu, klientów oferuje szereg funkcji dzięki którym bardzo szybko i łatwo stworzymy graficzny interfejs. Niestety ale do tworzenia aplikacji w którym główny mechanizm opiera się na ruchu elementów, tym bardziej jeśli chodzi o gry lepiej użyć którejś z biblioteki graficznej, która wspiera akceleracje sprzętową grafiki przy użyciu OpenGL.

Naszą aplikację możemy rozwinąć pod wieloma względami. Jednym z pomysłów jest dodanie chociażby poziomów do gry, które sprawią że im dłużej będziemy grać tym klocki będą spadać szybciej przez co będzie trudniej nam dopasować je w wiersze. Kolejne możliwości rozwoju są głównie graficzne. W przyszłości można dodać chociażby zmieniające się tło gry co poziom, różne animacje związane z usuwaniem wierszy czy tworzeniem nowych klocków. Menu gry również zostałoby wzbogacone o kolory. Można by dodać muzykę, która w tle umilałaby rozgrywkę oraz efekty dźwiękowe. Postaralibyśmy się również napisać wyświetlanie klocka, który jako następny się ma pojawić.

6. Załączniki

6.1. Instrukcja instalacji i obsługi aplikacji

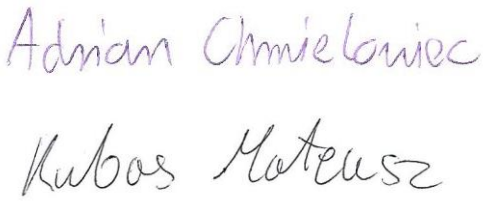
Instalacja naszego programu opiera się w głównej mierze na zainstalowaniu środowiska Code::Blocks najlepiej w wersji 20.03 oraz dołączenia biblioteki wxWidgets 3.0.5 z pomocą, której powstał nasz projekt. Po zainstalowaniu wystarczy odtworzyć nasz plik projektowy i go uruchomić przy pomocy skrótu F7 lub klikając na pasku Code:Blocks „build and run”. Możliwą opcją również jest odpalenie naszego programu bezpośrednio z pliku .exe. W tym celu należy przejść do katalogu Tetris_exe i tam powinien znajdować się nasz plik o nazwie TetrisWx.exe. Po uruchomieniu programu w taki sposób nam już się nasza aplikacja.

Obsługa aplikacji jest bardzo prosta. Po lewej stronie znajduje się panel w której będzie odbywać się nasza gra, natomiast po prawej znajduje się menu wraz z informacjami o rozgrywce. Po uruchomieniu naszego Tetrisa, mamy możliwość sterowania klockiem przesuwając go na boki lub obracając go przy pomocy strzałek na klawiaturze. Naciśniecie klawisza spacji spowoduje upuszczenie naszego klocka w dół. Przy pomocy ctrl możemy przyspieszyć spadanie naszego klocka. By użyć pauzy po prostu możemy nacisnąć P lub przycisk pauza, by wznowić wystarczy ponownie wykonać tą czynność. Po prawej stronie zostaną wyświetlone nasze punkty za każdym razem gdy zapełnimy wiersz klockami. Możemy w każdym momencie zresetować grę używając ponownie przycisku start. Istnieje także możliwość wyświetlenia rankingu graczy oraz wyłączenia gry. Informacje o autorach i skrótach znajdziemy w górnym menu naszej aplikacji, które otwieramy po prostu klikając.

6.2. Oświadczenie o samodzielności

Kielce 24.01.2021

Oświadczam, że projekt pt.: „*TETRIS*" został wykonany przez nas samodzielnie.



Podpisy osób wykonujących projekt