AGH University of Science and Technology

January 2025

	REV	DATA	ZMIANY
	0.1	23.01.2025	Filip Izworski (fizworski@student.agh.edu.pl)

Gra Komputerowa Tetris

Autor : Filip Izworski

Spis treści

1 Wstęp

Dokument dotyczy opracowania gry komputerowej Tetris, stworzonej przez Aleksieja Pażytnowa w 1984 roku. Celem gry jest układanie losowo generowanych figur w taki sposób, aby tworzyć w pełni zapełnione linie na planszy. Zapełnione linie są usuwane, co daje więcej miejsca na manewry gracza. Za każdą usuniętą linię przyznawane są punkty. Gra kończy się, gdy na planszy zabraknie miejsca na umieszczenie nowej figury.

Projekt zostanie zrealizowany w języku C++ z wykorzystaniem biblioteki SFML (Simple and Fast Multimedia Library), która posłuży do stworzenia aplikacji okienkowej i obsługi klawiatury.

2 Podstawowe założenia projektu

I. Platforma docelowa: komputery z systemem Windows x64
II. Użycie języka programowania obiektowego C++
III. Stworzenie interfejsu użytkownika za pomocą biblioteki SFML
IV. Podział projektu na dwie części: logikę gry i graficzny interfejs użytkownika
V. Rozgrywka zgodna z klasycznymi zasadami gry Tetris

3 Funkcjonalność

a) Figury
- Tworzenie wszystkich figur przewidzianych w grze: I, O, T, L, J, S, Z
- Tworzenie figur w różnych kolorach
- Możliwość przesuwania figury w lewo, w prawo, w dół oraz ich obracania
- Możliwość zapisu poprzedniej pozycji figury przed wykonaniem przemieszczenia w celu jej wczytania w przypadku wystąpienia kolizji

b) Plansza
- Tworzenie planszy o zadanej szerokości i wysokości
- Przechowywanie informacji dotyczących pojedynczych bloków takich jak pozycja i kolor
- Umieszczanie i usuwanie figur z planszy
- Sprawdzanie występowania kolizji między poruszającą się figurą, a resztą bloków
- Czyszczenie pojedynczych wierszy
- Przesuwanie bloków na wolne miejsce po wyczyszczonym wierszu

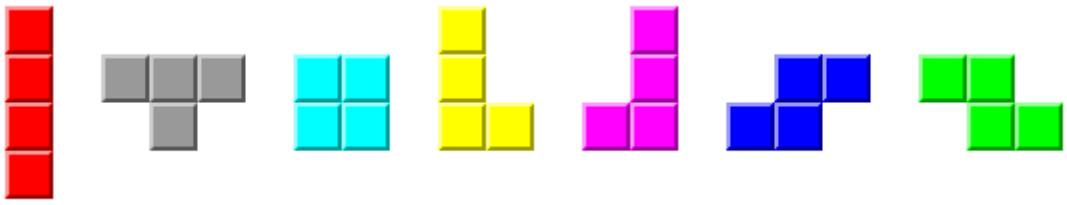
c) Logika gry
- Generowanie losowej figury na górze planszy
- Automatyczne przesuwanie figury w dół z zadaną prędkością lub szybciej w zależności od tego czy naciśnięto odpowiedni przycisk
- Sprawdzanie czy jakiś wiersz jest zapełniony w całości blokami
- Czyszczenie zapełnionego wiersza i dodawanie punktów dla gracza

d) Graficzny interfejs użytkownika
- Obsługa klawiatury
- Wyświetlanie planszy z blokami i figurą sterowaną przez gracza
- Wyświetlanie wyniku i następnej figury
- Wyświetlanie komunikatów o rozpoczęciu nowej gry lub zakończeniu jej

4 Analiza problemu

4.1 Figury

W grze Tetris występuje siedem figur zwanych także tetromino, widać je na rysunku 1. Każda figura może przybrać jeden z dostępnych w grze kolorów. Każdą figurę można podzielić na cztery osobne bloki, a każdy taki blok można scharakteryzować pozycją w przestrzeni dwuwymiarowej i kolorem. Rozdzielenie figur na bloki pozwala na łatwiejsze operowanie na planszy, na przykład kiedy wykonane zostanie czyszczenie wiersza to połowa figury może zostać usunięta. W przypadku traktowania figury jako całość może być to problematyczne, ale nie w przypadku podziału na bloki, gdzie po prostu zostaną usunięte dwa odrębne elementy.



Rysunek 1. Siedem figur występujących w grze Tetris, odpowiednio od lewej do prawej: I, T, O, L, J, S, Z, źródło: https://pl.wikipedia.org/wiki/Tetris

4.2 Sterowanie figurą

Realizacja przemieszczania figury w lewo, w prawo lub w dół nie jest skomplikowana. To co wystarczy zrobić to dodać lub odjąć jedynkę od współrzędnej x lub y dla wszystkich bloków w danej figurze. Większy problem stanowi obracanie. Jednym ze sposobów realizacji obracania jest wykorzystanie macierzy obrotu, w przypadku tego projektu sprawa jest uproszczona, gdyż potrzebny jest tylko obrót o kąt 90°. Macierz tą przedstawiono poniżej.

$$\text{Macierz obrotu o kąt } 90^\circ = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Mając macierz obrotu można wyprowadzić wzory 1 i 2 służące do obliczenia nowych pozycji po obrocie, należy jeszcze wybrać punkt wokół którego nastąpi obrót.

$$x'_n = x_c - y_n + y_c \quad (1)$$

$$y'_n = y_c + x_n - x_c \quad (2)$$

gdzie:

(x'_n, y'_n) – współrzędne po obrocie

(x_n, y_n) – współrzędne przed obrotem

(x_c, y_c) – punkt centralny wokół którego następuje obrót

4.3 Plansza

Podczas rozgrywki może wystąpić tylko jedna aktywna figura, czyli taka która może się poruszać i być sterowana przez gracza. Figura ta jak i wszystkie inne bloki muszą znajdować się w obrębie dwuwymiarowej planszy. W celu stworzenia takiej planszy można użyć macierzy, gdzie dany wierszy i kolumna są pozycją pojedynczego bloku, natomiast zawartość danej komórki macierzy to kolor bloku. Należy także zdefiniować symbol oznaczający że żadnego bloku na danej pozycji nie ma, gra musi zaczynać się z pustą planszą, czyli z macierzą wypełnioną właśnie tym symbolem.

4.4 Wykrywanie kolizji

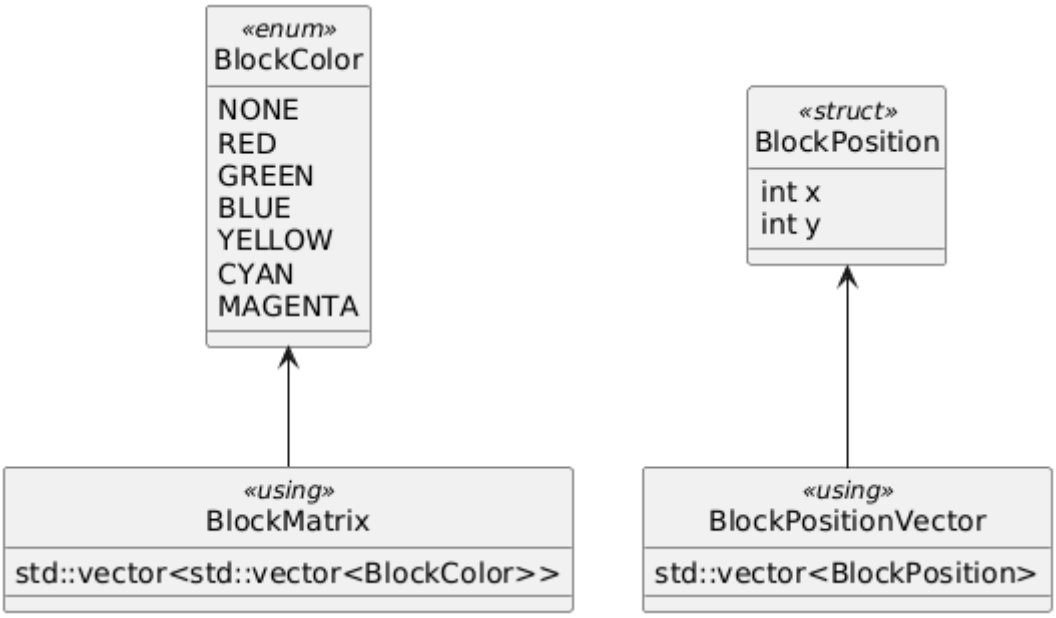
Żeby zadbać o pozostanie sterowanej figury w obrębie planszy jak i o poprawne zachowanie podczas dotknięcia podłoża lub innych bloków potrzebny jest system detekcji kolizji. W tym przypadku taki system jest dość prosty do realizacji, mając współrzędne bloków składających się na daną figurę można porównywać je z wymiarami planszy i sprawdzać czy na podanych współrzędnych jest już jakiś blok w macierzy.

5 Projekt techniczny

5.1 Podstawowe parametry bloków

Żeby opisać pojedynczy blok potrzebne są jego współrzędne i kolor. Współrzędne można zawrzeć w prostej strukturze przechowującej dwie liczby typu int, natomiast kolor może reprezentować wyliczenie. Warto podkreślić że tego typu kolory to tylko symbole, które nie są związane z żadną biblioteką służącą do tworzenia interfejsu graficznego. To pozwala na niezależne pisanie logiki gry.

Figury jak i samą planszę można interpretować jako kontenery przechowujące opisane tutaj parametry. W celu łatwego rozróżniania tych kontenerów zdefiniowano aliasy takie jak *BlockPositionVector* i *BlockMatrix*, oba korzystają z klasy *std::vector* co widać na rysunku 2.

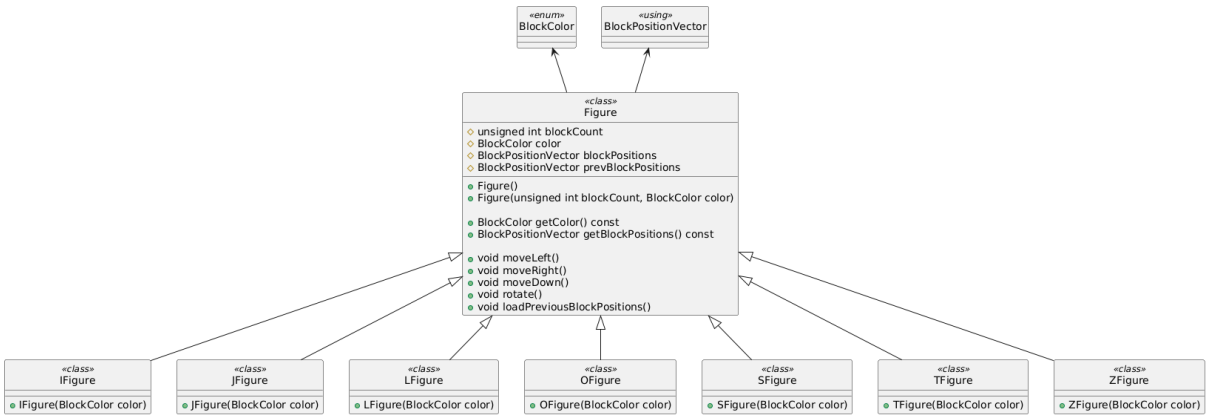


Rysunek 2. Diagram przedstawiający podstawowe parametry klocków i kontenery do ich przechowywania

5.2 Klasa *Figure*

Klasa *Figure* widoczna na rysunku 3 jest klasą bazową dla figur występujących w grze. Wykorzystano tutaj mechanizm dziedziczenia z tego powodu że wszystkie figury mają tą samą funkcjonalność, jedyne czym się różnią to kształt. Dlatego też jedyne co jest nowego w klasach potomnych to konstruktor, wewnątrz którego ustawiane są odpowiednie pozycje dla danej figury.

Funkcjonalność realizowana przez te klasy jest zgodna z tą opisaną w punkcie 3, jak widać na rysunku 3 są tam metody odpowiedzialne za przemieszczanie takie jak *moveLeft()*, *moveRight()*, *moveDown()* i *rotate()*, każda z tych metod przed wykonaniem właściwej akcji wykonuje zapis wektora obecnych pozycji bloków *blockPositions* do wektora *prevBlockPositions*. Z zapisanych pozycji korzysta metoda *loadPreviousPosition()* służąca do ich wczytania. Jest to użyteczne przy wykrywaniu kolizji, dzięki tej metodzie można łatwo wczytać poprzedni stan w przypadku gdy nowa pozycja nie spełnia wymagań.



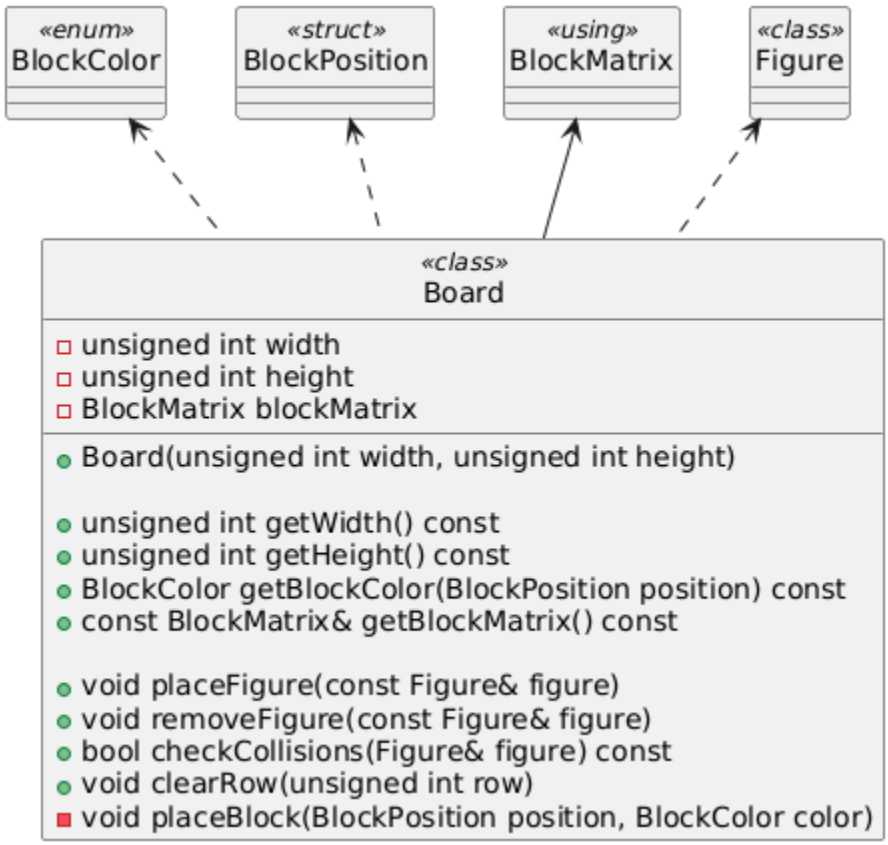
Rysunek 3. Diagram przedstawiający klasę Figure oraz jej potomków

5.3 Klasa *Board*

Plansza jest reprezentowana przez klasę *Board*, widoczną na rysunku 4. Jej wymiary są określane przy tworzeniu obiektu, czyli w konstruktorze. Dzięki metodzie *getBlockMatrix()* można pobrać całą macierz przechowującą informacje o zawartości planszy, jest to użyteczne przy rysowaniu w aplikacji okienkowej. Pojedyncze komórki można pobierać metodą *getBlockColor()*, która przyjmuję pozycję jako swój parametr.

Do manipulacji figurą sterowaną przez gracza na planszy służą metody: *removeFigure()*, *checkCollisions()* i *placeFigure()*. Metody te powinny być wywoływane w kolejności w jakiej zostały wypisane. Po ściągnięciu figury z planszy wykonywana jest jedna z metod służących do przemieszczenia, potem sprawdzane są kolizje. Jeśli kolizja wystąpiła to ładowane są poprzednie pozycje zapisane przed wykonaniem ruchu, jeśli nie to ten krok jest pomijany. Na końcu figura jest powtórnie umieszczana na planszy.

Klasa ta daje również możliwość czyszczenia pojedynczego wiersza dzięki metodzie *clearRow()*. Oprócz wyczyszczenia wiersza metoda ta również przesuwa całą zawartość planszy nad wyczyszczonym wierszem o jedną pozycję w dół.

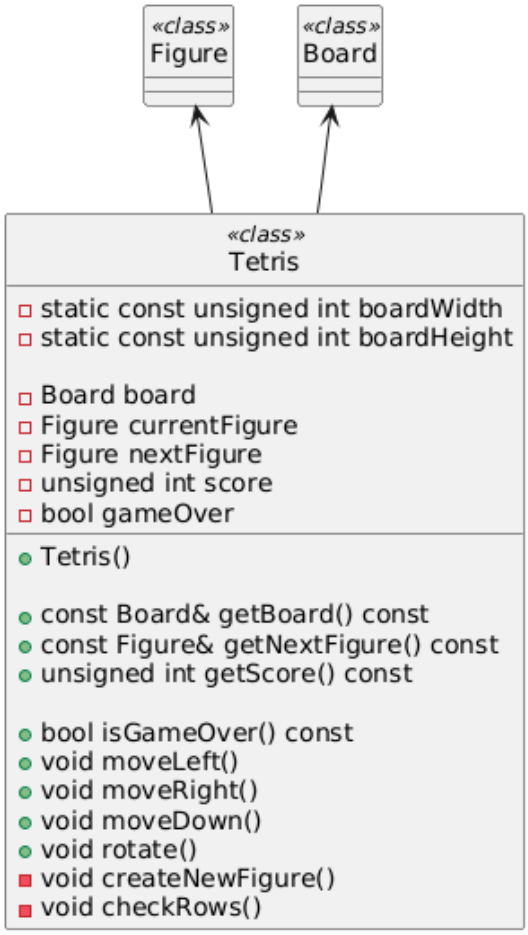


Rysunek 4. Diagram przedstawiający klasę Board

5.4 Klasa *Tetris*

Implementację logiki gry stanowi klasa *Tetris*, jest ona także klasą łączącą wszystkie wcześniej opisane w jedną całość. Jak widać na rysunku 5 składa się ona z jednego obiektu klasy *Board* i dwóch obiektów klasy *Figure*. Dzięki temu drugiemu obiektowi, czyli *nextFigure*, można wyświetlać w aplikacji okienkowej następną figurę nad którą kontrolę przejmie gracz. Sterowanie w obrębie planszy zostało zaimplementowane zgodnie z opisem z punktu 5.3. Metoda *moveDown()* sprawdza dodatkowo wiersze metodą *checkRows()*, która czyści zapełnione wiersze i zwiększa punktację w zmiennej *score*.

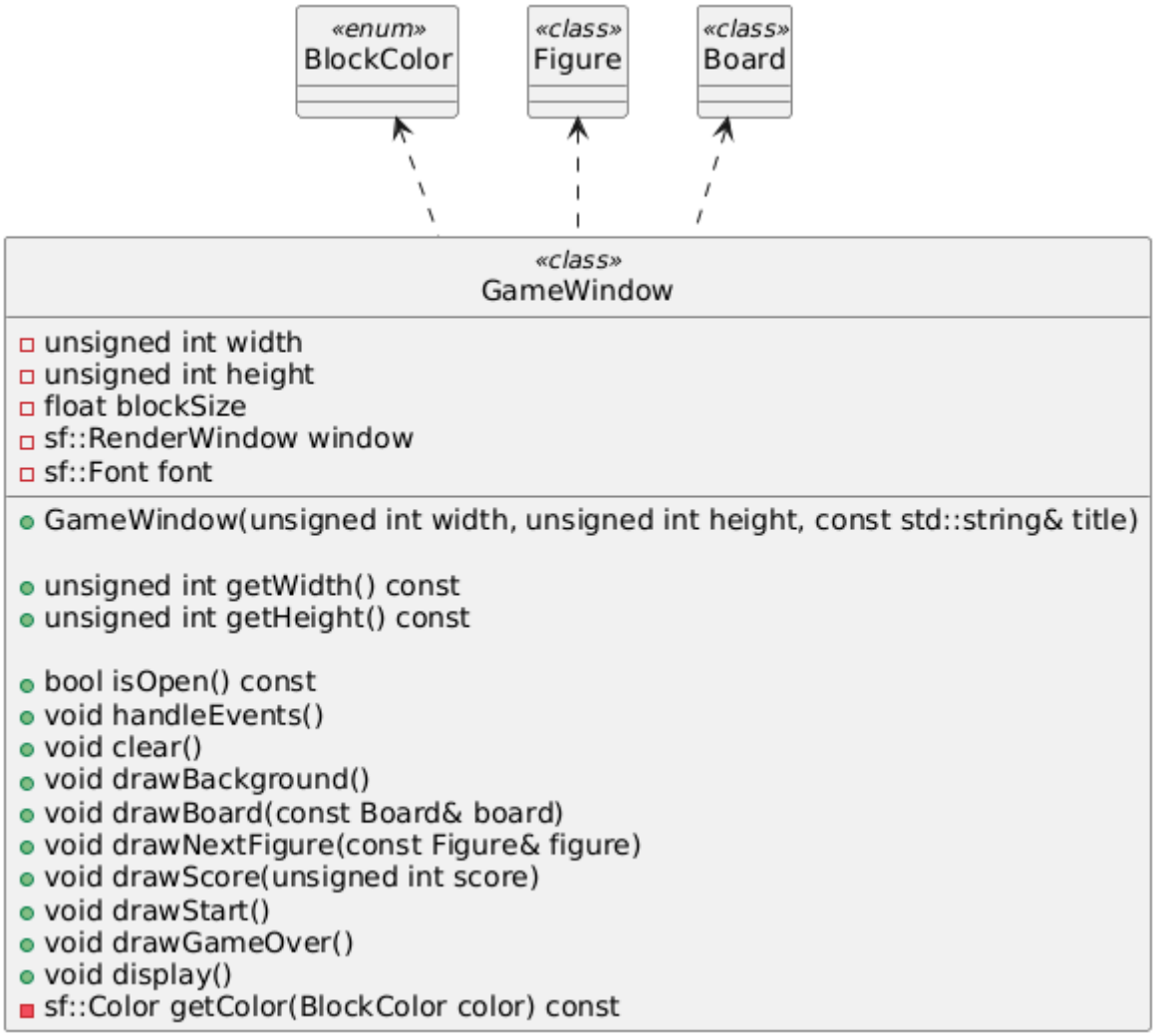
Nowe figury tworzone są w sposób losowy metodą *createNewFigure()*, jeśli nie ma możliwości żeby umieścić figurę na planszy to ustawiana jest flaga *gameOver*. Flaga ta sygnalizuje zakończenie rozgrywki i zatrzymuje całą grę, oczekując na restart.



Rysunek 5. Diagram klasy Tetris

5.5 Klasa *GameWindow*

Głównym komponentem stworzonej aplikacji okienkowej jest klasa *GameWindow*, odpowiada ona za rysowanie planszy, kolejnej figury czy wyświetlanie wyniku i innych komunikatów. Opiera się na klasie *sf::RenderWindow* z biblioteki SFML, która to zapewnia narzędzia do rysowania w okienku. Prywatna metoda *getColor()* znajdująca się na samym dole rysunku 6, służy do konwersji symbolicznego koloru na kolor widoczny w okienku obsługiwany przez bibliotekę SFML.

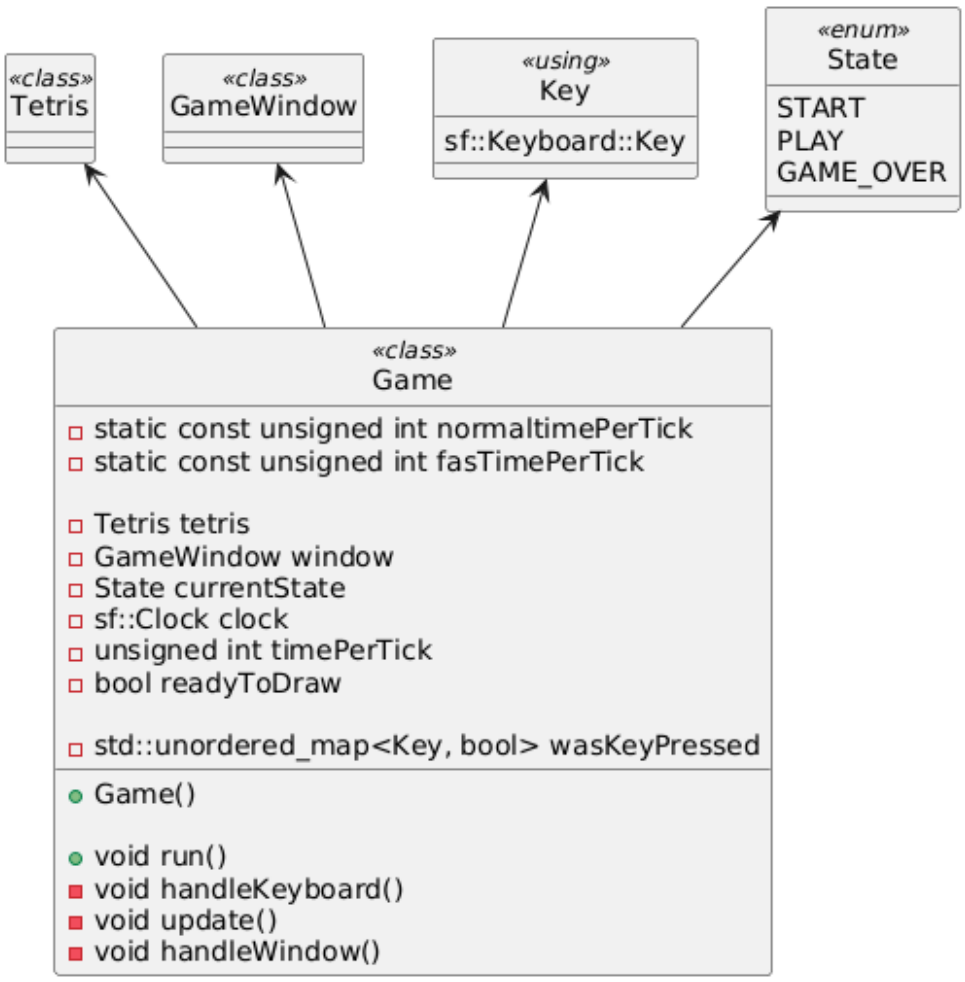


Rysunek 6. Diagram klasy GameWindow

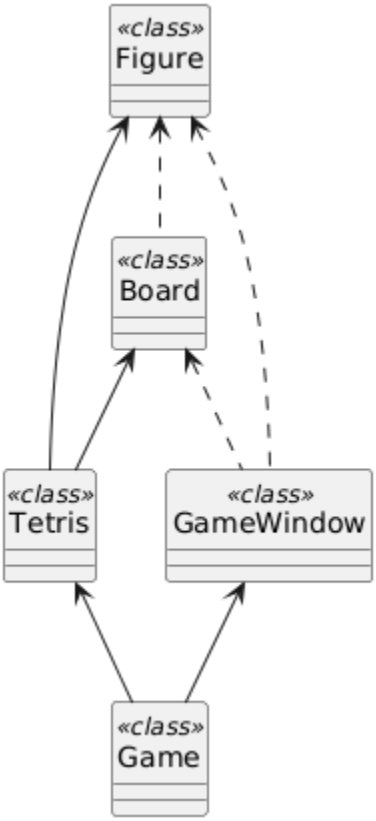
5.6 Klasa *Game*

Klasa *Game* łączy w sobie logikę gry i funkcjonalność aplikacji okienkowej, przedstawia to diagram zależności klas widoczny na rysunku 8. W celu automatycznego przesuwania figur w dół z zadanym odstępem czasowym, użyto obiektu klasy *sf::Clock* z biblioteki SFML. Klasa ta dodatkowo określa stan w którym znajduje się obecnie gra, po to żeby lepiej zorganizować przejścia pomiędzy włączeniem aplikacji, rozgrywką i końcem rozgrywki.

Żeby uruchomić grę należy stworzyć obiekt klasy *Game* i wywołać metodę *run()*, metoda ta składa się z trzech prywatnych metod widocznych na dole rysunku 7. Metody te służą kolejno do obsługi klawiatury, aktualizacji obecnego stanu gry i obsługi okienka.



Rysunek 7. Diagram klasy Game



Rysunek 8. Diagram zależności klas projektowanej gry

6 Opis realizacji

Poniżej przedstawiono opis platformy testowej i narzędzi użytych przy realizacji projektu.

6.1 Maszyna

- System operacyjny: Windows x64
- Procesor: Intel Core i5-1135G7 @ 2.4GHz
- RAM: 8 GB

6.2 Narzędzia programistyczne

- IDE: Visual Studio 2022
- Kompilator: MSVC
- Zewnętrzne biblioteki: SFML
- System budowania: CMake
- Platforma testowa: Google Test
- System utrzymania źródeł: GitHub

6.3 Opis systemu budowania

Do budowania projektu wykorzystano narzędzie CMake, żeby skonfigurować to narzędzie należy utworzyć plik CMakeLists.txt. Plik ten stworzono na bazie przykładu z podrozdziału opisującego CMake w książce "Programowanie w języku C++ Wprowadzenie dla inżynierów" autorstwa profesora Bogusława Cyganka. Konieczne było także dołączenie biblioteki SFML do projektu, zrobiono to zgodnie z instrukcjami przedstawionymi w repozytorium GitHub biblioteki SFML: https://github.com/SFML/cmake-sfml-project.

6.4 Opis platformy testowej

Jako platformę testową wykorzystano Google Test, jest ona zintegrowana ze środowiskiem Visual Studio 2022, dlatego nie było konieczne pobieranie kolejnych

zewnętrznych bibliotek. Żeby być w stanie przeprowadzić napisane testy należy najpierw zbudować projekt w folderze build, który trzeba utworzyć w katalogu zawierającym plik CMakeLists.txt. Testowanie możliwe jest tylko poprzez Visual Studio 2022.

Testy które zostały napisane dotyczą logiki gry, testują klasy *Figure*, *Board* i *Tetris*. Wszystkie przeprowadzone testy oraz ich wyniki widać na rysunku 9. Nie wszystkie metody nadawały się do przetestowania, na przykład z powodu tego że nic nie zwracają lub ich wpływ na działanie projektu jest ukryty. Takie metody i klasy obsługujące aplikację okienkową przetestowano manualnie podczas rozgrywki.

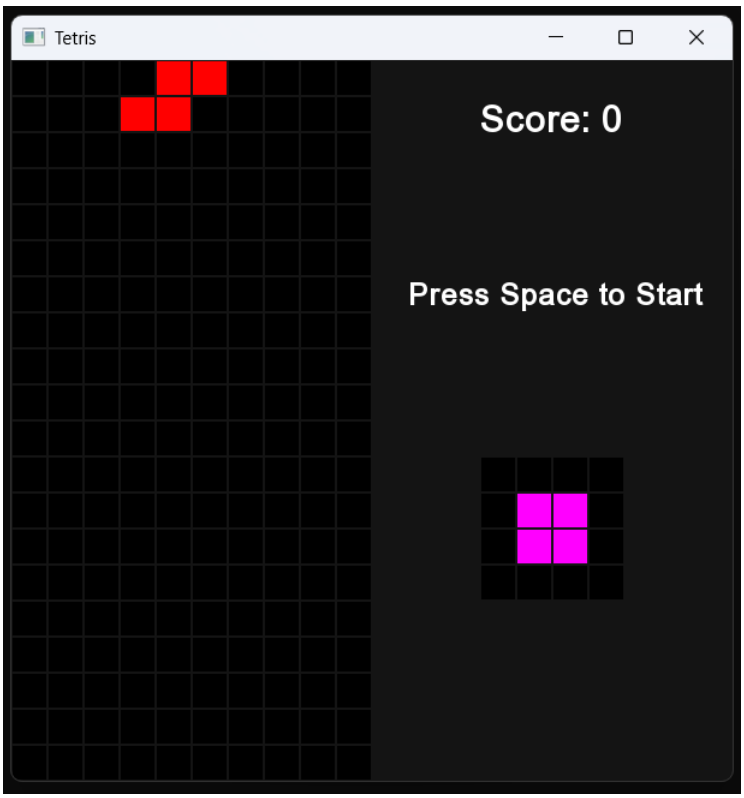
```
Running main() from D:\a\_work\1\s\googletest\googletest\src\gtest_main.cc
[==========] Running 9 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 5 tests from FigueTest
[ RUN      ] FigueTest.MoveLeft
[       OK ] FigueTest.MoveLeft (3 ms)
[ RUN      ] FigueTest.MoveRight
[       OK ] FigueTest.MoveRight (1 ms)
[ RUN      ] FigueTest.MoveDown
[       OK ] FigueTest.MoveDown (0 ms)
[ RUN      ] FigueTest.Rotate
[       OK ] FigueTest.Rotate (1 ms)
[ RUN      ] FigueTest.LoadPreviousBlockPositions
[       OK ] FigueTest.LoadPreviousBlockPositions (0 ms)
[----------] 5 tests from FigueTest (5 ms total)

[----------] 3 tests from BoardTest
[ RUN      ] BoardTest.PlaceRemoveFigure
[       OK ] BoardTest.PlaceRemoveFigure (0 ms)
[ RUN      ] BoardTest.CheckCollisions
[       OK ] BoardTest.CheckCollisions (0 ms)
[ RUN      ] BoardTest.ClearRow
[       OK ] BoardTest.ClearRow (0 ms)
[----------] 3 tests from BoardTest (1 ms total)

[----------] 1 test from TetrisTest
[ RUN      ] TetrisTest.GameOver
[       OK ] TetrisTest.GameOver (1 ms)
[----------] 1 test from TetrisTest (1 ms total)

[----------] Global test environment tear-down
[==========] 9 tests from 3 test cases ran. (15 ms total)
[  PASSED  ] 9 tests.
```

Rysunek 9. Testy zaprojektowanej gry przeprowadzone z użyciem Google Test i ich wyniki

7 Podręcznik użytkownika

7.1 Rozpoczęcie rozgrywki

Po uruchomieniu aplikacji pokaże się prawie pusta plansza, bo z gotową figurą do sterowania na górze. Przedstawia to rysunek 10. W celu rozpoczęcia rozgrywki należy nacisnąć spację, co sugeruje również komunikat po prawej stronie.



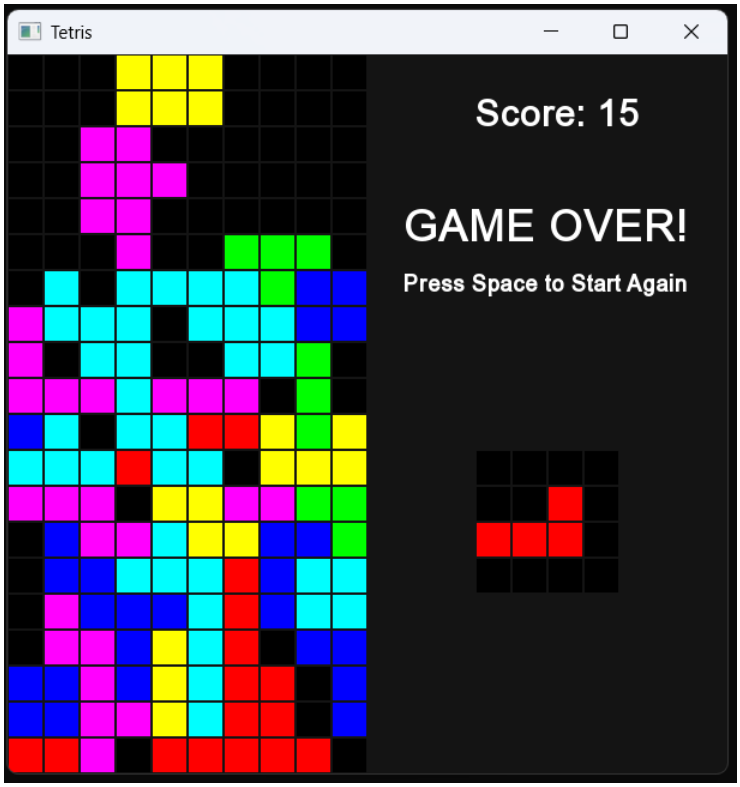
Rysunek 10. Ekran startowy zaprojektowanej gry

7.2 Sterowanie

W trakcie rozgrywki do sterowania dostępne są klawisze W, A, S i D. Klawisz W służy do obracania figury, A i D to odpowiednio ruch w lewo lub w prawo o jedną pozycję, natomiast S to przyspieszenie spadania figury.

7.3 Koniec rozgrywki

W momencie w którym nie jest już możliwe umieszczenie nowej figury na planszy bez żadnych kolizji gra się kończy. Aplikacja nie będzie odpowiadać na wciskanie klawiszy do sterowania, tak jak na początku należy nacisnąć spację by rozpocząć od nowa. Ekran końcowy przedstawia rysunek 11.



Rysunek 11. Ekran końcowy zaprojektowanej gry

7.4 Punktacja

Punkty otrzymuje się przy kompletowaniu pełnych wierszy. Wiersze są sprawdzane po każdym przesunięciu figury w dół. Jeśli w danym cyklu wykryto więcej niż jeden wiersz to otrzymuje się dodatkowe punkty. Ilość dodanych punktów w takich przypadkach to kwadrat liczby linii, na przykład jeśli udało się skompletować trzy linie za jednym razem to otrzymamy dziewięć punktów.

Bibliografia

[1] https://pl.wikipedia.org/wiki/Tetris [dostęp 22.01.2025]

[2] Cyganek B.: Programowanie w języku C++. Wprowadzenie dla inżynierów. PWN, 2023.

[3] https://www.sfml-dev.org/ [dostęp 22.01.2025]