

Dokumentacja programu

Game of Life

Spis treści

Opis gry i funkcjonalności.....	2
Interfejs po uruchomieniu.....	3
Interfejs gry.....	6
Opis kodu.....	8
Klasy:.....	8
GameInstance:.....	8
GameMenu:.....	10
TileMap (plik Board.h):.....	12
MainMenu:.....	14
Button:.....	15
Mapa:.....	16
PatternContainer:.....	18
PatternTemplate:.....	18

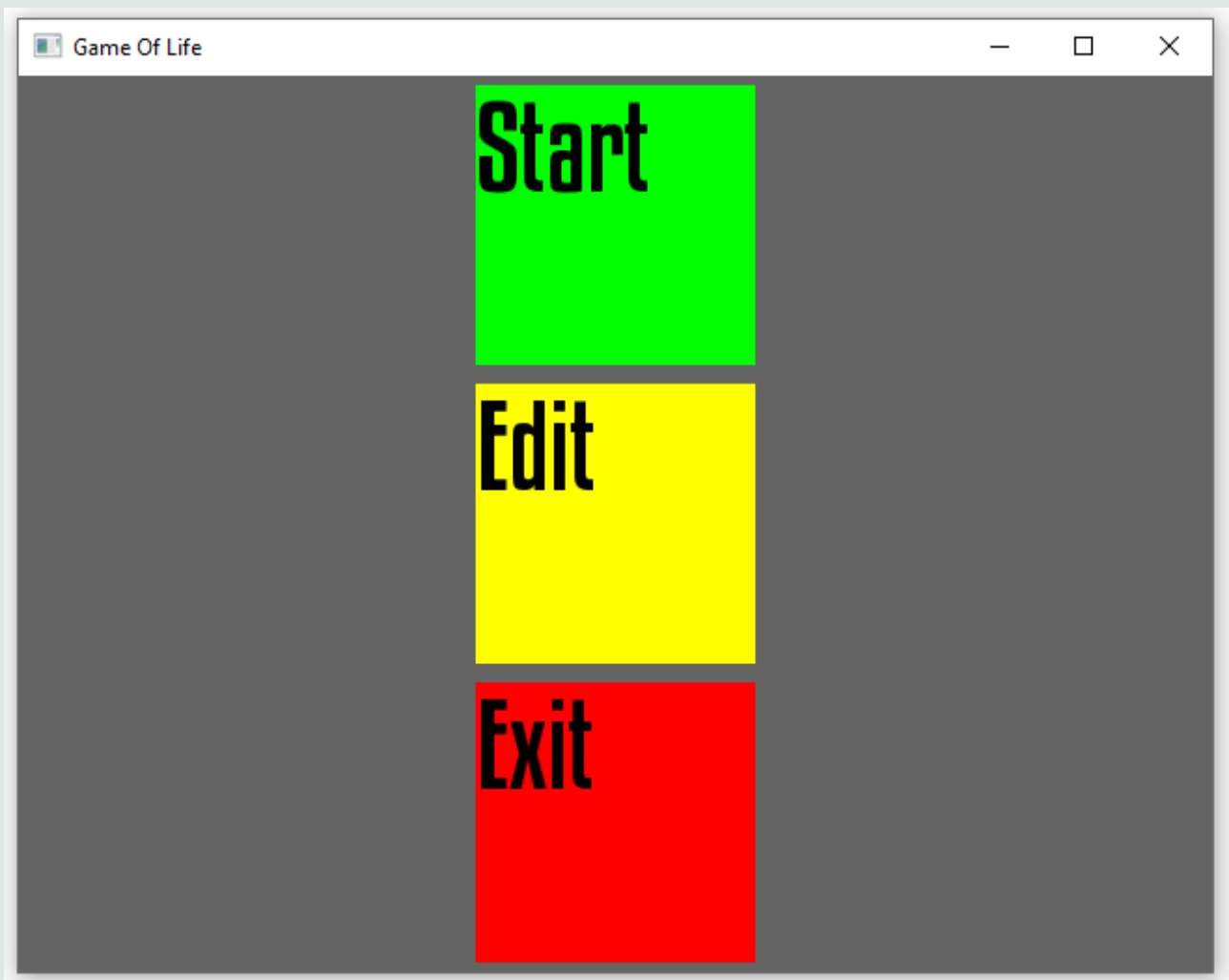
Opis gry i funkcjonalności

Gra polega na symulowaniu „Game of life” John’a Conway’a.

Plansza składa się z komórek, które mogą być żywe lub martwe. Gracz może dowolnie zmieniać stan komórki. Kiedy symulacja planszy jest uruchomiona co pewien czas jest obliczana następna iteracja planszy według 4 prostych zasad:

1. Żywa komórka która ma mniej niż 2 żywych sąsiadów umiera.
2. Żywa komórka z 2-3 żywymi sąsiadami pozostaje żywa.
3. Żywa komórka z więcej niż 3 żywymi sąsiadami umiera.
4. Martwa komórka z dokładnie 3 żywymi sąsiadami ożywa.

Interfejs po uruchomieniu



Po uruchomieniu gry pokaże się okno z 3 przyciskami.

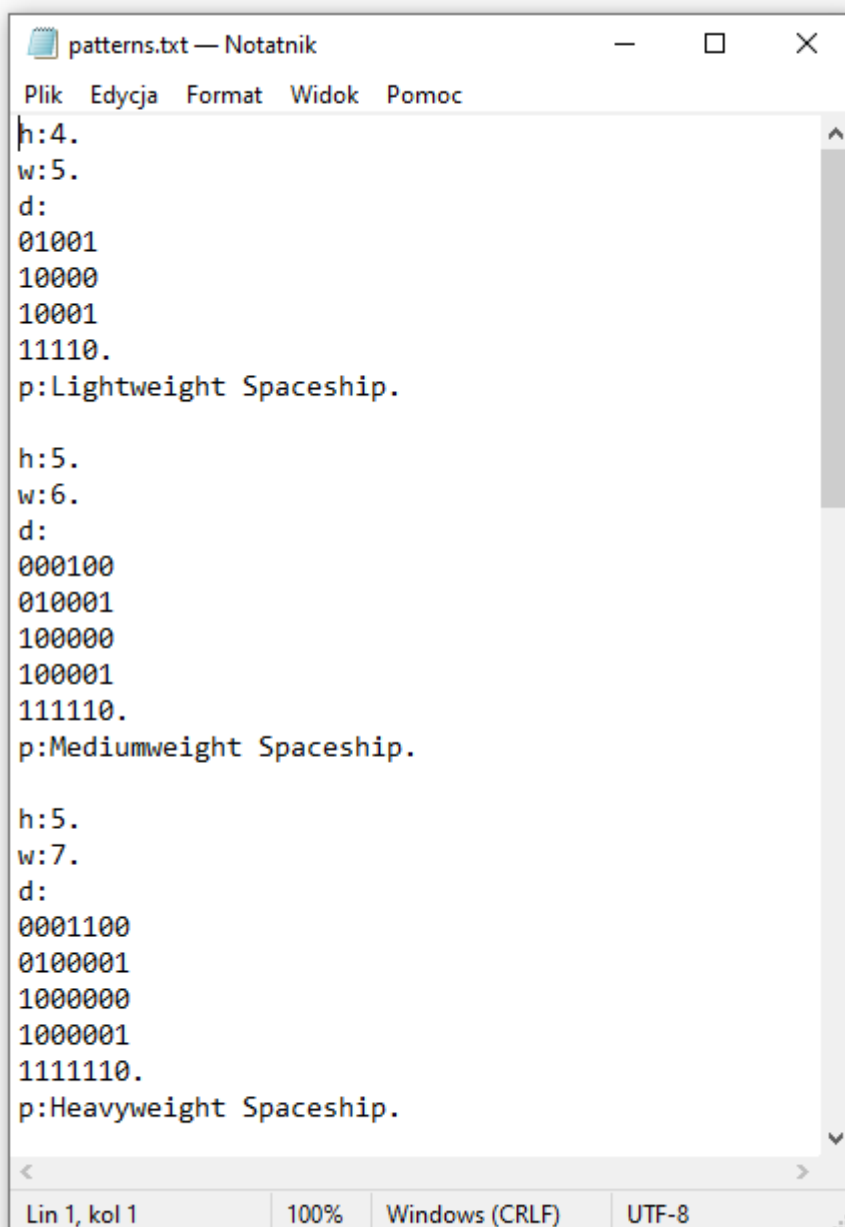
- Start przełącza ekran na ekran z planszą z komórkami
- Edit otwiera Eksplorator Plików w miejscu gdzie jest zapisany plik patterns.txt
- Exit wyłącza programu

Plik patterns.txt zawiera definicje wzorków które można wkleić w programie do planszy komórek.

Wzory są czytane w następujący sposób:

Program szuka znaku definiującego którąś ze zmiennych: h (height) definiuje wysokość wzoru, w (width) definiuje jego szerokość, d (data) definiuje dane i p (pattern) definiuje nazwę. Program zapamiętuje przeczytane zmienne o szerokości, wysokości i danych wzoru i wczytuje je do programu kiedy natknie się na definicję nazwy (p). Dlatego ważne jest, aby w pliku zapisać p jako ostatnią część definicji wzoru. Bardzo ważne jest też stosowanie kropek kiedy dane dla danej zmiennej się kończą, inaczej program będzie czytał kolejne znaki dalej jako część zmiennej.

Przykład definicji wzoru:



```
patterns.txt — Notatnik
Plik  Edycja  Format  Widok  Pomoc
h:4.
w:5.
d:
01001
10000
10001
11110.
p:Lightweight Spaceship.

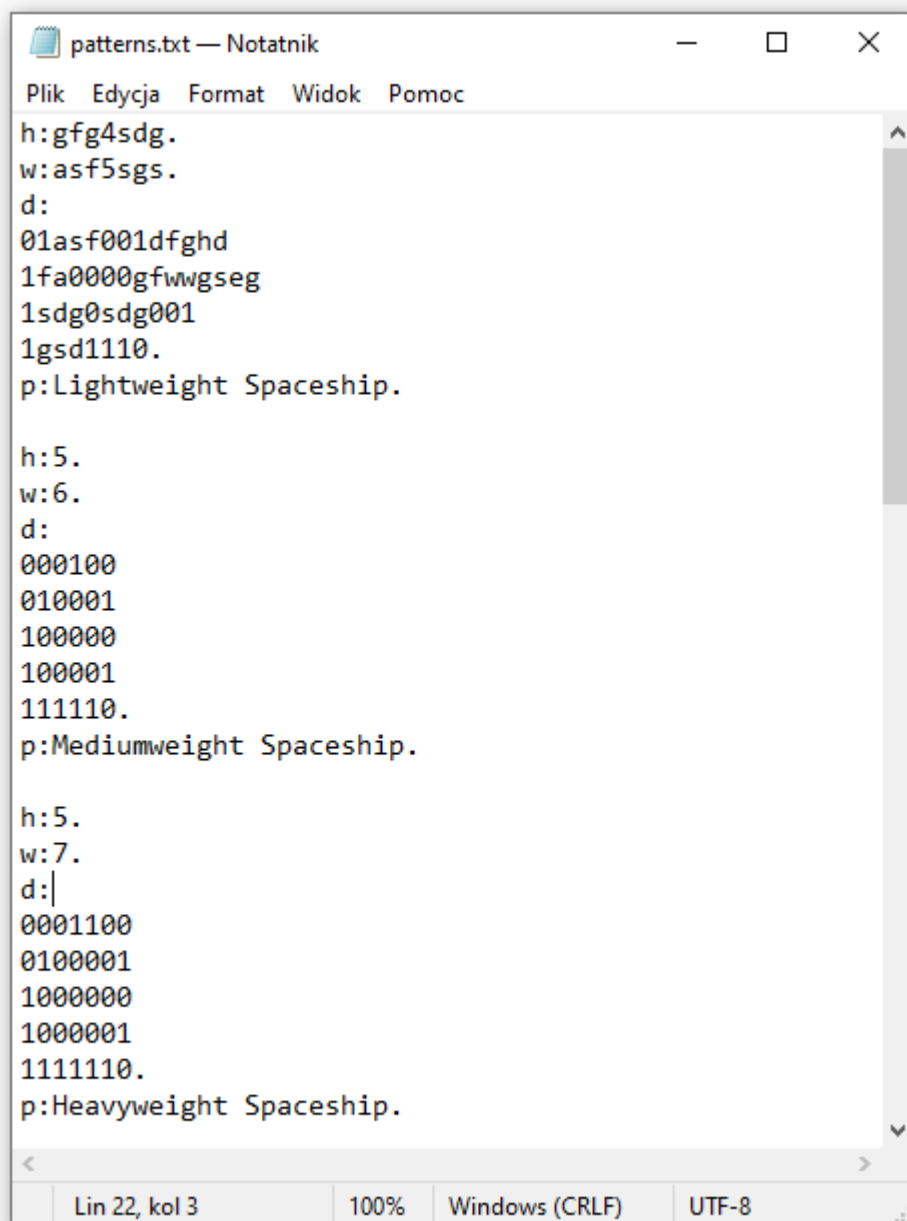
h:5.
w:6.
d:
000100
010001
100000
100001
111110.
p:Mediumweight Spaceship.

h:5.
w:7.
d:
0001100
0100001
1000000
1000001
1111110.
p:Heavyweight Spaceship.
```

Dane o wzorze (d) mogą być zapisane w dowolnej liczbie linii, ale zalecane jest ułożenie danych w sposób jaki mają być faktycznie wyświetlane (pomaga to w wpisywaniu wzoru i w czytelności). Dwukropki są również opcjonalne, ale zalecane bo poprawiają czytelność

Podczas wczytywania danych o wysokości i szerokości program wczyta wszystkie znaki między 'h'/'w' a kropką, ale pod uwagę weźmie tylko liczby; zapis „eee2aaa4bc56” zostanie zinterpretowany jako liczba 2456. Przy czytaniu danych o konstrukcji wzoru (d) program przeczyta wszystkie znaki między 'd' a kropką, ale zignoruje wszystkie niebędące '1' albo '0'. Przy wczytywaniu nazwy (p) program przeczyta wszystkie znaki między 'p' a kropką z wyjątkiem dwukropka

Następująca definicja wzoru jest technicznie poprawna i zostanie przeczytana przez program tak samo jak wcześniejsza, ale jest mało czytelna:



```
patterns.txt — Notatnik
Plik  Edycja  Format  Widok  Pomoc
h:gfg4sdg.
w:asf5sgs.
d:
01asf001dfghd
1fa0000gfwgseg
1sdg0sdg001
1gsd1110.
p:Lightweight Spaceship.

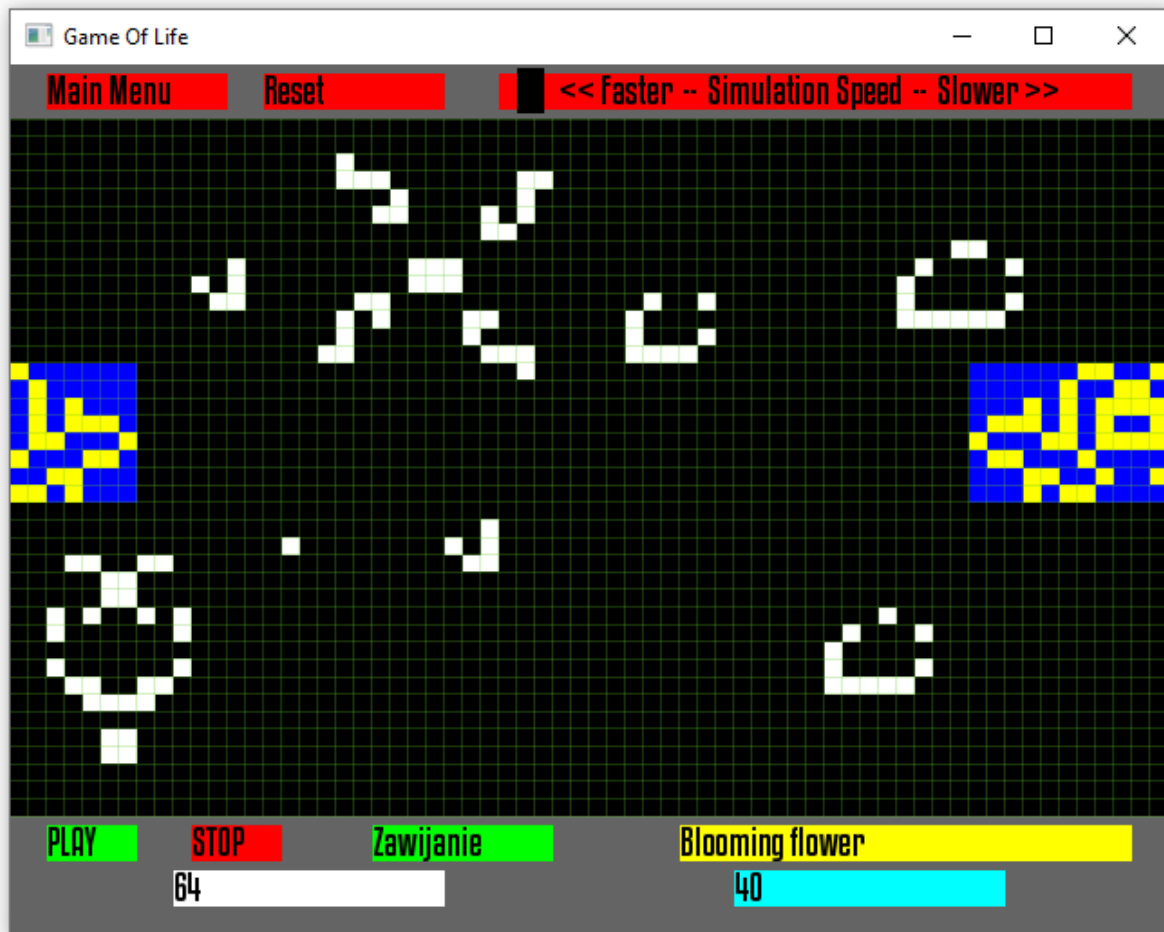
h:5.
w:6.
d:
000100
010001
100000
100001
111110.
p:Mediumweight Spaceship.

h:5.
w:7.
d:|
0001100
0100001
1000000
1000001
1111110.
p:Heavyweight Spaceship.
```

Program sprawdza czy ilość danych (d), czyli jedynek i zer zgadza się z podanymi wymiarami wysokości i szerokości (wysokość * szerokość). Jeśli te wartości się nie zgadzają to program zignoruje definicję wzoru i przeczyta następny.

Dodatkowo program sprawdza czy wartości w, h i d zostały podane przed każdym argumentem p, jeśli nie, to zostaną one zignorowane, a program przejdzie do następnego wzoru

Interfejs gry



Największą część ekranu zajmuje plansza z komórkami. Można na nie klikać aby zmieniać ich stan między żywymi i martwymi.

- Main Menu – zmienia ekran do poprzedniego z 3 przyciskami
- Reset – ustawia wszystkie komórki na martwe i pauzuje symulację
- Play/Stop – uruchamia/stopuje symulację
- Zawijanie – włącza/wyłącza zawijanie. Zawijanie polega na tym, że komórka na skraju planszy może wejść w interakcję z komórkami na przeciwnym skraju planszy. Jeśli ta opcja jest wyłączona, to wszystkie miejsca poza planszą są traktowane jako martwe
- Żółte pole – służy do wyboru wzoru wczytanego z pliku do wklejenia. Wzory można zmieniać kręcąc kółkiem myszy gdy kursor jest nad polem. Kliknięcie włączy jednorazowe wklejenie wzoru do planszy z podglądem, ponowne kliknięcie wyłączy wklejanie. Widoczny wzór z żółtych i niebieskich komórek jest

wizualizacją wzoru w trybie wklejania. Wciśnięcie lewego przycisku myszy wklei go we wskazanym miejscu

- Suwak „Simulation speed” – klikając na suwak możemy zmieniać prędkość przechodzenia do następnej iteracji. Na lewo przyspieszamy symulację, na prawo ją spowalniamy
- Biały przycisk – odpowiada za szerokość planszy, klikając lewym i prawym przyciskiem myszy można rozszerzać/zwężać planszę. Dodatkowo gdy przewiniemy kółkiem myszy gdy wskazujemy na przycisk możemy szybko zwiększać/zmniejszać szerokość. Minimalna szerokość to 3 komórki, nie ma maksymalnego rozmiaru
- Błękitny – robi to samo co biały, ale dla wysokości planszy

Opis kodu

Aplikacja jest napisana w C++ z użyciem Visual Studio 2022. Biblioteka graficzna to SFML 2.6. Program jest w wersji 32-bitowej

Klasy:

GameInstance:

Klasa zawierająca całą grę. Stworzenie dwóch obiektów i wejście w ich pętlę główną w dwóch różnych wątkach stworzy 2 niezależne okna z grą

Zmienne:

- **bool** czyGraDziala {**false**} – definiuje, czy symulacja ma być obliczana
- **int** odstepCzasu {200} – zmienna z odstępem czasu między iteracjami symulacji w ms
- **int** windowHeight {640}, windowHeight {480} – początkowy rozmiar okna aplikacji w pikselach
- **MENU** menu {MAIN} – enumerator z wartościami BOARD i MAIN, decyduje na którym ekranie jesteśmy (menu główne / plansza)
- **const sf::Color** background {100,100,100,150} – wyznacza kolor tła okienka
- **sf::Clock** zegar – obiekt zegara, używany do ustalenia kiedy należy obliczyć kolejną iterację planszy
- **PatternContainer** wzorki – obiekt przechowujący wczytane wzory z pliku
- **sf::RenderWindow** *window – wskaźnik na obiekt okna aplikacji
- **Mapa** *cellMap – wskaźnik na obiekt z informacjami o planszy, obsługuje obliczanie kolejnej iteracji
- **MainMenu** *mainMenu – wskaźnik na obiekt generujący 3 przyciski menu głównego
- **GameMenu** *gameMenu – wskaźnik na obiekt generujący kontrolki gry na ekranie z planszą
- **TileMap** *board – wskaźnik na obiekt rysujący planszę z komórkami, używa danych z obiektu cellMap

Funkcje:

- `GameInstance()` - konstruktor, inicjalizuje wstępne wartości aplikacji oraz zapełnia wskaźniki obiektami
- `~GameInstance()` - destruktor, zwalnia pamięć zajęta przy tworzeniu obiektów
- `void handleEvents()` - pętla obsługująca wydarzenia/wiadomości przysłane do aplikacji

GameMenu:

Klasa tworzy, rysuje i obsługuje przyciski znajdujące się na ekranie z planszą z komórkami

Zmienne:

- **int** oknoW, oknoH – aktualny rozmiar okna
- **int** origW, origH – oryginalny rozmiar okna, zmienne z oryginalnym i obecnym rozmiarem są potrzebne do skalowania pozycji kursora. SFML automatycznie dostosuje elementy w oknie do rozmiaru okna, więc w programie ich rozłożenie zostaje takie same, ale pozycja kursora będzie teraz wartością na podstawie aktualnego rozmiaru. Możemy pomnożyć wartości x i y kursora przez (orig/okno) i otrzymamy pozycję kursora względem rozmiaru oryginalnego okna, co ułatwia późniejsze obliczenia
- **int** sliderVal {0} – wartość suwaka szybkości symulacji, jest obliczany z pozycji samego suwaka względem paska suwaka
- **Button** scena[ILE_GUZILOW] – tablica z obiektami klasy Button. Te obiekty tworzą przyciski, na które można kliknąć

Funkcje:

- **virtual void draw**(sf::RenderTarget& target, sf::RenderStates states) **const** - redefinicja funkcji SFML aby obiekt klasy rysował elementy menu, za które odpowiada
- **int getSliderVal()** - funkcja przekazuje wartość suwaka
- **GameMenu(int w, int h, int boardWidth, int boardHeight)** – konstruktor klasy ustawiający wartości z rozmiarem okna oraz ustawiający parametry guzików w tablicy, takie jak ich położenie, kolor i tekst. Rozmiar planszy do gry jest potrzebny aby ustawić tekst na przyciskach zmieniających rozmiar planszy na jej aktualny rozmiar.
- **int handleEvent**(sf::Event event) – funkcja obsługująca wydarzenia. Jej zadaniem jest sprawdzenie na który przycisk naciśnięto i zwrócenie jego numeru oraz aktualizacja zmiennych z aktualnym rozmiarem okna kiedy użytkownik je zmieni.
- **void updateBtnText**(int index, std::string tekst) - wrapper do funkcji aktualizującej tekst na guziku w tabeli (funkcja klasy Button)

- `void updateBtnColor(int index, sf::Color nowyKolor)` - wrapper do funkcji zmieniającej kolor guzika (funkcja klasy Button)

TileMap (plik Board.h):

Klasa rysuje i obsługuje akcje użytkownika na planszy. Samą tablicę z wartościami komórek obsługuje inna klasa.

Zmienne:

- **Mapa** *cellMap - wskaźnik na obiekt przechowujący informacje o planszy do gry, potrzebne przy rysowaniu planszy aby sprawdzić stan komórek
- **const float** marginTop {30.0f}, marginBottom {65.0f} - zmienne ustawiające margines rysowanej planszy od górnej i dolnej krawędzi okna aplikacji
- **int** numer - zmienna pomocnicza oznaczająca indeks komórki na planszy
- **float** rozmiarX, rozmiarY - jaki ma być rozmiar rysowanych komórek, obliczany przy utworzeniu obiektu klasy i przy zmianie rozmiaru planszy
- **float** oknoW, oknoH, origW, origH - zmienne z obecnym i oryginalnym rozmiarem okna aplikacji. Mają tą samą funkcję co w klasie GameMenu oraz pomagają obliczyć odpowiedni rozmiar komórek do rysowania
- **int** holoX, holoY - ostatnie pozycja myszki na planszy, wyznaczana w rzędzie i wierszu komórki na której była myszka. Używane do obsługi rysowania hologramu i wklejania go do planszy.
- sf::**VertexArray** kwadraty - wektor z punktami, które tworzą trójkąty, które tworzą kwadraty reprezentujące rysowane komórki
- sf::**VertexArray** linie - wektor z punktami do rysowania linii między komórkami
- sf::**Color** paletaKolor[10] - tablica z kolorami jakie ma mieć komórka zależnie od jej wartości. Komórka może mieć wartość nie tylko 0 lub 1, ale może mieć wartości od 0 do 9. Te wartości są używane przez program do optymalizacji algorytmu symulacji oraz rysowania hologramu wklejanego wzoru.
- **bool** pasteInEnabled {**false**} - zmienna oznaczająca czy użytkownik włączył wklejanie wzorów, domyślnie wklejanie jest wyłączone
- **bool** showHolo {**false**} - zmienna oznaczająca czy mamy rysować hologram. Zmienna ma wartość true kiedy myszka znajduje się nad planszą i false gdy jest poza nią. Dzięki niej hologram nie jest niepotrzebnie rysowany gdy użytkownik nie wybiera komórki.
- std::**vector**<**int**> holoData - wektor do przechowywania danych komórek obecnie wybranego wzorku

- **int** holoWidth, holoHeight - rozmiar obecnie wybranego wzorku
- **enum DrawOutline** {YES, NO} - czy mają być rysowane linie między komórkami
- **DrawOutline** drawBounds {NO} - faktyczna zmienna z informacją czy mamy rysować linie między komórkami
- **sf::Color** kolorLinii {**sf::Color::Magenta**} - kolor linii między komórkami

Funkcje:

- **virtual void draw**(**sf::RenderTarget&** target, **sf::RenderStates** states) **const** - redefinicja funkcji SFML aby obiekt klasy rysował kwadraty i linie planszy
- **void updateKwadraty**() - funkcja aktualizująca kolor rysowanych kwadratów na podstawie wartości komórek
- **void updateLinie**() - funkcja aktualizująca pozycje punktów rysujących linie między komórkami
- **TileMap**(**Mapa** *cellMap, **int** wW, **int** wH) - konstruktor ustawiający wszystkie parametry. Ustawia wstępny rozmiar wektorów z punktami i te punkty ustawia w odpowiednich miejscach z pomocą metod klasy
- **void HandleEvent**(**sf::Event** event) - handler wydarzeń. Odpowiada za znalezienie indeksu komórki na którą kliknięto/najechno, zmianę jej stanu oraz rysowanie/wklejenie hologramu od tej komórki oraz za aktualizację zmiennych z obecnym rozmiarem okna.
- **void resizePlansza**() - aktualizuje rozmiar rysowanych kwadratów, rozmiar wektora z punktami, koordynaty punktów oraz rysowanie linii przy zmianie rozmiaru tablicy z kwadratami
- **void hologram**(**bool** clicked) - funkcja rysuje hologram wklejanego wzorku zaczynając od pozycji holoX holoY. Jeśli przekazany parametr jest false, to tylko go rysuje, jeśli true, to go wkleja do tablicy z danymi komórek. Funkcja jest używana przez HandleEvent()
- **void resizeKwadraty**() - aktualizuje pozycję punktów rysujących kwadraty. Używane przez konstruktor i resizePlansza().

MainMenu:

Klasa tworzy, rysuje i obsługuje 3 przyciski w menu głównym aplikacji

Zmienne:

- **int** oknoW, oknoH, origW, origH - taka sama funkcja co w poprzednich klasach; wartości oryginalne i aktualne rozmiaru okna do skalowania pozycji myszki do oryginalnego rozmiaru okna
- **Button** scena[ILE_GUZYKOW] - tablica z 3 guzikami rysowanymi w menu głównym

Funkcje:

- **virtual void draw**(sf::RenderTarget& target, sf::RenderStates states) **const** - rysowanie guzików, wrapper do funkcji draw obiektu Button
- **MainMenu(int w, int h)** - konstruktor. Ustawia parametry guzików oraz wartości okno i orig
- **int handleEvent**(sf::Event event) - obsługa wydarzeń, czyli kliknięcia i zmiany rozmiaru okna

Button:

Klasa odpowiadająca za pojedynczy przycisk

Zmienne:

- `sf::Font` czcionka - Zmienna z czcionką, czcionka jest ładowana z pliku.
- `sf::Text` text - Tekst wyświetlany na guziku. Zawiera też parametry tekstu, takie jak kolor i rozmiar.
- `int` height, width - Zmienne z rozmiarem guzika w pikselach.
- `int` posX, posY - Pozycja guzika. 0,0 znajduje się w lewym górnym rogu.
- `sf::RectangleShape` rectangle - Obiekt kwadratu, który jest rysowany jako guzik.

Funkcje:

- `virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const` - rysuje pojedynczy guzik który reprezentuje obiekt
- `int mouseOnButton(int x, int y)` - Przyjmuje pozycję myszki i sprawdza czy myszka znajduje się w obrębie guzika. Jeśli tak, to zwraca 1, inaczej 0.
- `void setButton(int w, int h, int X, int Y, std::string tekst, sf::Color tlo, int charSize)` - Funkcja ustawiająca wszystkie parametry guzika: rozmiar, pozycję, tekst, kolor tła i rozmiar czcionki.
- `void moveBttn(int x, int y)` - Zmienia pozycję guzika
- `void changeText(std::string tekst)` - Zmiana tekstu na guziku
- `void changeColor(sf::Color nowyKolor)` - Zmiana koloru tła guzika

Mapa:

Klasa odpowiadająca za przechowywanie tablicy z wartościami komórek, zmianę rozmiaru tablicy oraz za obliczanie następnej iteracji symulacji

Zmienne:

- **int** *map - Tablica z komórkami, alokowana dynamicznie, aby rozmiar zgadzał się z rozmiarem ustawionym przez użytkownika
- **int** width, height - Rozmiar planszy w komórkach, width * height to długość tablicy map
- **bool** zawijanie - Czy zawijanie planszy jest włączone

Funkcje:

- **int*** generateNeighbourNums(**int** cellNum) - Funkcja tworzy i zwraca tablicę z pozycją sąsiadów komórki, której numer przekazano jako argument. Jeśli zawijanie jest włączone, to dla komórki na skraju mapy komórki sąsiadujące wychodzące poza zakres będą komórkami na przeciwnym skraju mapy. Jeśli zawijanie jest wyłączone, to pozycja takich komórek będzie ustawiona na ujemną, co mówi innym funkcjom, ale pominąć te komórki
- **int** countNeighbours(**int** cellNum) - Funkcja liczy i zwraca ilość żywych sąsiadów komórki, używa generateNeighbourNums do znalezienia sąsiadów
- **int** decideCellState(**int** neighbourCnt, **int** currentState) - Funkcja zwraca jaki ma być następny stan komórki (jej wartość) na podstawie ilości sąsiadów i obecnego stanu komórki
- **void** calculateNeighbours(**int** cellNum) - Funkcja oblicza jaki powinien być następny stan sąsiadów komórki o podanym numerze. Używa generateNeighbourNums do znalezienia sąsiadów, a potem dla każdego oblicza jego następny stan. Funkcja oblicza tylko komórki martwe, bo komórki żywe są obliczane przez funkcję główną calculateMapa() oraz pomija komórki obliczone wcześniej (np. jako sąsiad innej żywej komórki obok).
- **void** calculateMapa() - Główna funkcja obliczająca następną iterację tablicy z komórkami. Przechodzi po całej tablicy, ale wykonuje obliczenia tylko dla żywych komórek. Martwe komórki są obliczane wokół znalezionej żywej komórki przez calculateNeighbours()(obliczanie martwej komórki bez żywych sąsiadów nie ma sensu, bo na pewno nie zmieni swojego stanu).
- **void** resetMapa() - Ustawia wszystkie komórki na wartość 0.

- **void** **resizePlansza**(**int** deltaWidth, **int** deltaHeight) - Funkcja zmienia rozmiar tablicy na podstawie przekazanych argumentów, które mówią o ile zmienić rozmiar w którym wymiarze (wysokość/szerokość). Wartość ujemna zmniejsza tablicę, dodatnia zwiększa.
- **Mapa**() oraz **Mapa**(**int** width, **int** height) – Konstruktory, ustawiają wysokość i szerokość oraz tworzą tablicę z komórkami. Można podać rozmiar, albo zostawić domyślne wartości (wysokość 64 oraz szerokość 40)
- **~Mapa**() - Destruktor, zwalnia pamięć zajęta przez tablicę

PatternContainer:

Klasa odpowiada za ładowanie wzorów z pliku oraz ich przechowywanie

Zmienne:

- `int` wybranyWzorek {0} – Numer obecnie wybranego wzorku
- `std::vector<PatternTemplate>` patterns – Wektor ze wzorkami do wyboru. Przechowywane jako obiekt klasy PatternTemplate

Funkcje:

- `PatternContainer()` - Konstruktor ładujący wzory z pliku patterns.txt do wektora patterns

PatternTemplate:

Klasa przechowuje informacje o pojedynczym wzorku

Zmienne:

- `int` height, width – Zmienne z wysokością i szerokością wzoru
- `std::vector<int>` data – wektor z danymi wzoru ()które komórki są żywe, a które martwe
- `std::string` name – nazwa wzoru

Funkcje:

- `PatternTemplate(int patternW, int patternH, std::vector<int> PatternData, std::string patternName)` – Konstruktor ustawiający wszystkie zmienne na odpowiednie wartości