**Dokumentacja Gry Boggle**

Zespół:

* Tomasz Młynarczyk
* Karolina Móżdżyńska

MenuFunctions

1**. `getLogoWidth()`**

* Oblicza szerokość najdłuższej linii ASCII logo zapisanego w tablicy `ascii\_logo`.
* Iteruje przez kolejne linie logo, używając `strlen()` do znalezienia długości każdej z nich i przechowuje największą wartość.
* Wynik jest używany do centrowania logo na ekranie.

**2. `printASCIILogo(WINDOW\* window)`**

* Wyświetla ASCII logo w podanym oknie ncurses (`window`).
* Oblicza początkowe współrzędne (x, y), aby logo było wycentrowane na ekranie.
* Wykorzystuje funkcję `mvwprintw()` do wyświetlenia każdej linii logo w odpowiednim miejscu.

**3. `makeLogo()`**

* Tworzy okno na logo o określonych rozmiarach i lokalizacji na ekranie.
* Ustawia ramkę wokół okna za pomocą `box()`, a następnie wywołuje `printASCIILogo()`, aby narysować logo wewnątrz.
* Zwraca wskaźnik do utworzonego okna.

**4. `buttonInit(WINDOW\*\* button, int y, int x, const char\* text)`**

* Tworzy przycisk (okno ncurses) o określonej pozycji (`y`, `x`) i wyświetla na nim podany tekst.
* Oblicza, gdzie wyśrodkować tekst wewnątrz przycisku, a następnie używa `mvwprintw()` do jego umieszczenia.
* Rysuje ramkę wokół przycisku za pomocą `box()`.

**5. `makeButtons(WINDOW\* buttons[2][2])`**

* Tworzy tablicę 2x2 przycisków i rozmieszcza je w różnych sekcjach ekranu (np. "GRAJ", "LEADERBOARD", "INSTRUKCJA GRY", "WYJŚCIE").
* Wywołuje `buttonInit()` dla każdego przycisku, definiując jego położenie i tekst.

**6. `displayButtons(WINDOW\* buttons[2][2])`**

* Odświeża zawartość wszystkich przycisków, aby były widoczne na ekranie.
* Dla każdego przycisku w tablicy wywołuje `wrefresh()`.

**7. `enterInteractionLoop(WINDOW\* buttons[2][2])`**

* Wykonuje główną pętlę interakcji użytkownika w menu.
* Obsługuje ruch kursora między przyciskami za pomocą strzałek (`KEY\_UP`, `KEY\_DOWN`, `KEY\_LEFT`, `KEY\_RIGHT`).
* Podświetla aktualnie wybrany przycisk, zmieniając jego tło (`COLOR\_PAIR`), a po wciśnięciu `ENTER` zwraca indeks wybranego przycisku.

**8. `convertUserChoice(int userChoice)`**

* Konwertuje indeks wybranego przycisku na wartość typu `ScreenChoice` (np. `GAME`, `LEADERBOARD`, `EXIT`).
* Ułatwia komunikację między menu a główną pętlą programu.

**9. `menu()`**

Obsługuje cały ekran menu:

* Czyści ekran i przygotowuje go do rysowania.
* Tworzy logo i przyciski, wyświetla je na ekranie.
* Oczekuje na wybór użytkownika, wchodząc w pętlę `enterInteractionLoop()`.
* Zwalnia pamięć przydzieloną na logo i przyciski.
* Zwraca wybór użytkownika jako wartość `ScreenChoice`.

GameFunctions

**1. `displayBoard(string board[4][4], WINDOW\* okno)`**

* Wyświetla planszę gry w oknie `okno`.
* Iteruje przez wszystkie komórki planszy (4x4) i drukuje ich zawartość w odpowiednich miejscach okna, w formacie `| X |`.
* Odświeża okno po wprowadzeniu zmian za pomocą `wrefresh()`.

**2. `updateFoundWordsCounterDisplay(int liczba\_znalezionych\_slow)`**

* Aktualizuje licznik znalezionych słów wyświetlany w rogu ekranu.
* Używa `mvprintw()` do zapisania aktualnej liczby słów w określonym miejscu ekranu.

**3. `wyswietlListeZnalezionychSlow(WINDOW\* lista\_okno, vector<string> found\_words)`**

* Wyświetla listę znalezionych słów w oknie `lista\_okno`.
* Czyści poprzednią zawartość okna za pomocą `werase()`, rysuje ramkę (`box()`), a następnie iteruje przez listę słów, wyświetlając je w kolejnych wierszach.
* Odświeża zawartość okna.

**4. `gameScreenInit()`**

* Inicjalizuje ekran gry, wyświetlając instrukcje obsługi w lewym górnym rogu oraz liczbę znalezionych słów.
* Tekst jest drukowany w stałych lokalizacjach za pomocą `mvprintw()`.

**5. `updateBoard(WINDOW\* plansza\_okno, GameState state, struct coordinates cursor)`**

* Aktualizuje planszę gry na podstawie stanu gry (`GameState`) i pozycji kursora (`cursor`).
* Podświetla komórkę z kursorem (`A\_REVERSE | COLOR\_PAIR(1)`) i wcześniej wybrane litery (`A\_REVERSE | COLOR\_PAIR(2)`).
* Po zakończeniu rysowania odświeża okno za pomocą `wrefresh()`.

**6. `restartGameDisplay(GameState state, WINDOW\* lista\_okno, WINDOW\* plansza\_okno)`**

* Resetuje i odświeża ekran po restarcie gry.
* Wyświetla nową planszę, czyści listę znalezionych słów oraz usuwa komunikaty w dolnej części ekranu (`clrtoeol()`).

**7. `tryAddSelectedWord(GameState\* state, WINDOW\* lista\_okno)`**

* Próbuje dodać wybrane przez gracza słowo do listy znalezionych.
* Sprawdza, czy słowo istnieje na planszy oraz czy nie zostało już wcześniej dodane.
* Jeśli słowo jest poprawne, wyświetla komunikat i aktualizuje listę w oknie `lista\_okno`.
* W przeciwnym razie wyświetla odpowiedni komunikat o błędzie.
* Na koniec czyści wybrane litery.

**8. `enterInteractionLoop(WINDOW\* plansza\_okno, WINDOW\* lista\_okno, GameState\* state)`**

* Główna pętla gry, obsługująca interakcje użytkownika.
* Zarządza czasem gry i aktualizuje licznik czasu w czasie rzeczywistym.
* Obsługuje ruch kursora za pomocą strzałek oraz akcje takie jak wybór litery (`ENTER`), sprawdzanie słowa (`W`), restart gry (`R`) lub wyjście (`X`).
* Aktualizuje wygląd planszy i licznik znalezionych słów w trakcie gry.
* Po zakończeniu (limit czasu lub wyjście) przerywa pętlę.

**9. `game(Leaderboard\* leaderboard)`**

* Rozpoczyna grę, inicjalizując nową planszę gry i tworząc okna dla planszy (`plansza\_okno`) oraz listy słów (`lista\_okno`).
* Wyświetla początkowy stan planszy i ekranu gry za pomocą `gameScreenInit()`.
* Uruchamia główną pętlę gry za pomocą `enterInteractionLoop()`.
* Po zakończeniu gry oblicza wynik gracza i sprawdza, czy zmienia on ranking (`Leaderboard`).
* Jeśli tak, przechodzi do ekranu aktualizacji rankingu (`LEADERBOARD\_UPDATE`).
* W przeciwnym razie wraca do menu (`MENU`).

GameState

**1. `void fillBoard()`**

* Wypełnia planszę gry (`board`) losowymi wartościami z tablicy `dice`, która zawiera konfiguracje kości Boggle.
* Dla każdej komórki planszy losuje jedną z sześciu możliwych liter na podstawie indeksu kości (`rand() % 6`).

**2. `bool hasBeenSelected(struct coordinates cursor)`**

* Sprawdza, czy dana komórka (reprezentowana przez `cursor`) została już wcześniej wybrana.
* Iteruje przez listę wybranych współrzędnych (`selected\_coordinates`) i porównuje je z aktualnym kursorem.

**3. `bool isValidSelection(struct coordinates selected\_letter)`**

Sprawdza, czy wybór nowej litery jest poprawny:

* Jeśli żadna litera nie została wybrana, wybór jest automatycznie poprawny.
* Jeśli ta sama litera została wybrana ponownie, wybór jest niepoprawny.
* Dla innych przypadków sprawdza, czy nowa litera jest w sąsiedztwie ostatnio wybranej (odległość maksymalnie 1 pole w pionie, poziomie lub po skosie).

**4. `void tryAddLetter(struct coordinates cursor)`**

* Próbuje dodać literę z pozycji kursora do listy wybranych liter (`selected\_letters`) i współrzędnych (`selected\_coordinates`).
* Najpierw sprawdza poprawność wyboru za pomocą `isValidSelection()`.

**5. `void startGame()`**

Rozpoczyna nową grę:

* Wypełnia planszę (`fillBoard()`).
* Inicjalizuje obiekt `BoardAnalyzer` z aktualną planszą i ścieżką do słownika.
* Pobiera wszystkie słowa możliwe do znalezienia na planszy i zapisuje je w `all\_words`.

**6. `bool isOnBoard(string word)`**

* Sprawdza, czy dane słowo znajduje się na planszy (jest w zbiorze `all\_words`).
* Iteruje przez wszystkie słowa możliwe na planszy i porównuje je z podanym słowem.

**7. `bool tryAddSelectedWord()`**

* Próbuje dodać aktualnie wybrane słowo do listy znalezionych słów (`found\_words`).
* Jeśli słowo nie zostało jeszcze znalezione (`isFoundWord()`), dodaje je do listy i zwraca `true`. W przeciwnym razie zwraca `false`.

**8. `void restartGame()`**

Restartuje grę:

* Czyści wybrane litery i współrzędne za pomocą `clearSelection()`.
* Opróżnia listę znalezionych słów (`found\_words`).
* Uruchamia nową grę za pomocą `startGame()`.

**9. `string getSelectedWord()`**

* Generuje słowo z aktualnie wybranych liter (`selected\_letters`), łącząc je w ciąg znaków.

**10. `int getSelectedLength()`**

* Zwraca liczbę aktualnie wybranych liter, obliczaną na podstawie długości wektora `selected\_coordinates`.

**11. `bool isFoundWord(string word)`**

* Sprawdza, czy dane słowo zostało już znalezione, iterując przez listę `found\_words`.

**12. `void clearSelection()`**

* Czyści wszystkie dane związane z aktualnym wyborem:
* Wektor współrzędnych (`selected\_coordinates`).
* Wektor liter (`selected\_letters`).

**13. `vector<struct coordinates> getSelectedCoordinates()`**

* Zwraca wektor współrzędnych wszystkich aktualnie wybranych liter.

**14.`int calculateScore()`**

* Oblicza wynik gracza na podstawie długości znalezionych słów:

- 1 punkt za słowa o długości 3–4.

- 2 punkty za słowa o długości 5.

- 3 punkty za słowa o długości 6.

- 5 punktów za słowa o długości 7.

- 11 punktów za słowa o długości 8+.

* Iteruje przez listę `found\_words`, sumując punkty zgodnie z powyższymi zasadami.

BoardAnalyzer

**1. `BoardAnalyzer::BoardAnalyzer(string board[RZEDY][KOLUMNY], string dictionary\_file\_path)`**

* Konstruktor klasy.
* Inicjalizuje planszę (`board`) i ładuje słownik za pomocą obiektu `Dictionary`.
* Kopiuje wartości z podanej planszy do wewnętrznej tablicy obiektu `BoardAnalyzer`.

**2. `void fillNeighborList()`**

* Tworzy graf planszy w postaci listy sąsiedztwa (`neighbor\_list`).
* Dla każdego wierzchołka (komórki planszy) oblicza jego sąsiadów:
* Sąsiadami są wszystkie komórki w odległości maksymalnie 1 pola w pionie, poziomie i na ukos, z wyjątkiem samego wierzchołka.
* Każdy wierzchołek jest reprezentowany przez pojedynczy numer (`rows \* RZEDY + cols`), a jego sąsiedzi są przechowywani w wektorze.

**3. `void findAllWords()`**

* Znajduje wszystkie możliwe słowa na planszy, zaczynając od każdego wierzchołka.
* Dla każdej komórki planszy wywołuje rekurencyjną funkcję `searchNode()`.
* Po zakończeniu usuwa duplikaty słów z listy za pomocą `removeDuplicates()`.

**4. `vector<string> getAllWords()`**

* Zwraca listę wszystkich znalezionych słów (`all\_words`).
* Jeśli lista słów jest pusta, generuje ją, wywołując `fillNeighborList()` i `findAllWords()`.

**5. `void removeDuplicates()`**

Usuwa duplikaty z wektora `all\_words`:

* Sortuje listę słów.
* Używa funkcji `unique()` do przeniesienia duplikatów na koniec listy.
* Usuwa duplikaty za pomocą `erase()`.

**6. `void searchNode(int node\_number)`**

Rekurencyjnie przeszukuje graf w poszukiwaniu słów:

* Dodaje bieżący wierzchołek do listy odwiedzonych (`visited`).
* Tworzy aktualne słowo na podstawie odwiedzonych wierzchołków za pomocą `visitedLettersAsString()`.
* Wyszukuje w słowniku słowo, które zaczyna się od tego ciągu znaków, oraz kolejne słowo:
* Jeśli aktualne słowo jest pełnym słowem w słowniku, dodaje je do `all\_words`.
* Jeśli jest prefiksem innych słów w słowniku, kontynuuje przeszukiwanie dla sąsiadów bieżącego wierzchołka.
* Po zakończeniu odwiedza kolejnych sąsiadów.
* Usuwa bieżący wierzchołek z listy odwiedzonych po zakończeniu analizy.

**7. `string visitedLettersAsString()`**

* Tworzy ciąg znaków (słowo) na podstawie liter w odwiedzonych wierzchołkach (`visited`).
* Wykorzystuje indeksy wierzchołków, aby pobrać odpowiednie litery z tablicy `board`.

Dictionary

**1. `Dictionary::Dictionary(string file\_path)`**

* Konstruktor klasy `Dictionary`.
* Inicjalizuje obiekt `Dictionary`, przypisując ścieżkę pliku (`file\_path`) i wczytując listę słów do wektora `word\_list` za pomocą funkcji `loadWordList()`.

**2. `bool Dictionary::loadWordList(string file\_path)`**

* Wczytuje słowa z pliku o podanej ścieżce (`file\_path`) do wektora `word\_list`.
* Otwiera plik za pomocą `ifstream` i odczytuje go linia po linii.
* Usuwa zbędne znaki końca linii (`\r`) po każdym wczytaniu linii.
* Jeśli plik jest otwarty i dane zostały poprawnie wczytane, zwraca `true`, w przeciwnym razie zwraca `false`.
* Na końcu zamyka plik.

**3. `int Dictionary::findWord(string word)`**

* Wyszukuje słowo (`word`) w słowniku za pomocą algorytmu wyszukiwania binarnego.
* Inicjuje dwa wskaźniki `l` i `r` (lewy i prawy), które wskazują na początek i koniec zakresu w wektorze `word\_list`.
* Wykonuje iteracje, dzieląc zakres na połowy, aż znajdzie słowo lub stwierdzi, że nie istnieje w słowniku.
* Zwraca indeks słowa w wektorze, jeśli je znajdzie, w przeciwnym razie zwraca `-1`.

**4. `int Dictionary::findWordThatStartsWith(string prefix)`**

* Szuka pierwszego słowa w słowniku, które zaczyna się od danego prefiksu (`prefix`).
* Zastosowano wyszukiwanie binarne, zakładając, że każde słowo jest także prefiksem swojego wystąpienia.
* Jeśli prefiks jest dokładnym słowem w słowniku, zwróci je jako pierwsze.
* Zwraca indeks pierwszego słowa, które zaczyna się od podanego prefiksu, lub indeks pierwszego słowa, które jest większe niż podany prefiks.

**5. `string Dictionary::getWord(int index)`**

* Zwraca słowo z listy `word\_list` znajdujące się pod podanym indeksem `index`.
* Jeśli indeks jest poza zakresem, zwraca pusty ciąg znaków (`""`).

**6. `bool Dictionary::hasWord(string word)`**

* Sprawdza, czy dane słowo (`word`) istnieje w słowniku, używając algorytmu wyszukiwania binarnego.
* Działa podobnie jak `findWord()`, ale zamiast zwracać indeks, zwraca `true`, jeśli słowo zostało znalezione, lub `false`, jeśli go nie ma.

Leaderboard

**1. `Leaderboard::Leaderboard(int max\_length)`**

* Konstruktor klasy `Leaderboard`.
* Inicjalizuje obiekt `Leaderboard`, przypisując maksymalną długość listy wyników (`max\_length`) oraz inicjalizując `score\_buffer` na 0.
* `score\_buffer` przechowuje tymczasowy wynik, który może być używany w trakcie gry przed jego zapisaniem na liście wyników.

**2. `bool Leaderboard::doesChangeLeaderboard(int new\_score)`**

* Sprawdza, czy nowy wynik (`new\_score`) powinien zmienić listę wyników (czy powinien być dodany do leaderboardu).
* Jeśli liczba elementów w `items` (liście wyników) jest mniejsza niż maksymalna długość (`max\_length`), zwraca `true`, ponieważ można dodać nowy wynik.
* Następnie iteruje przez listę wyników i sprawdza, czy któryś wynik w `items` jest mniejszy niż nowy wynik. Jeśli tak, zwraca `true`.
* W przeciwnym razie zwraca `false`, jeśli nowy wynik nie zmienia leaderboardu.

**3. `void Leaderboard::insertScore(std::string name, int new\_score)`**

* Wstawia nowy wynik do listy wyników (`items`) w odpowiednie miejsce, zachowując porządek wyników malejąco.
* Tworzy nowy element `LeaderboardItem` z nazwą gracza (`name`) i wynikiem (`new\_score`).
* Następnie przeszukuje listę `items` i znajduje odpowiednią pozycję, wstawiając nowy wynik przed pierwszym wynikiem, który jest mniejszy od nowego wyniku.
* Jeśli liczba wyników przekroczy maksymalną dozwoloną długość (`max\_length`), usuwa ostatni element w `items` (najniższy wynik).
* Jeśli liczba wyników jest mniejsza niż `max\_length`, dodaje nowy wynik na koniec listy.

**4. `std::vector<struct LeaderboardItem> Leaderboard::getItemsList()`**

* Zwraca wektor `items`, który zawiera listę wyników graczy w grze.
* Lista ta jest posortowana malejąco, z najlepszym wynikiem na początku.

**5. `int Leaderboard::getMaxLength()`**

* Zwraca maksymalną długość listy wyników (`max\_length`), która jest ustawiona podczas tworzenia obiektu `Leaderboard`.

**6. `void Leaderboard::setScoreBuffer(int new\_score)`**

* Ustawia wartość `score\_buffer` na nowy wynik (`new\_score`).
* `score\_buffer` przechowuje tymczasowy wynik, który może być wykorzystany do dalszych operacji, np. do zapisania na liście wyników.

**7. `int Leaderboard::getScoreBuffer()`**

* Zwraca wartość tymczasowego wyniku przechowywanego w `score\_buffer`.

LeaderboardFunctions

**1. `void displayList(Leaderboard \*lb)`**

* Funkcja odpowiedzialna za wyświetlenie listy wyników na ekranie.
* Pobiera listę wyników (`items`) z obiektu `Leaderboard` za pomocą funkcji `lb->getItemsList()`.
* Iteruje po wszystkich wynikach i za pomocą funkcji `mvprintw` wypisuje je na ekranie w postaci: `numer. imię - wynik: wartość`.
* Dla każdego elementu w `items` wyświetla numer, nazwisko gracza (`items[i].name`) oraz wynik (`items[i].score`).
* Jeśli liczba wyników jest mniejsza niż maksymalna długość (`lb->getMaxLength()`), wypisuje dodatkowe linie z informacją, że dane pole jest puste, aż do osiągnięcia pełnej listy wyników.

**2. `ScreenChoice leaderboard(Leaderboard \*lb)`**

* Funkcja wyświetlająca ekran leaderboardu.
* Pierwsza linia funkcji czyści ekran (funkcja `clear()`), a następnie odświeża ekran (funkcja `refresh()`), aby przygotować go do rysowania.
* Następnie wywołuje funkcję `displayList(lb)` w celu wyświetlenia wyników na ekranie.
* Po wyświetleniu listy wyników, na ekranie pojawia się informacja o możliwości powrotu do menu: `"Aby wrocic do menu, nacisnij dowolny przycisk."`, umieszczona w linii tuż po ostatnim wpisie na liście wyników.
* Funkcja czeka na naciśnięcie dowolnego przycisku za pomocą funkcji `getch()`, a po jego naciśnięciu zwraca wartość `MENU`, co może sygnalizować powrót do głównego menu lub wcześniejszy ekran.

LeaderboardUpdateFunctions

**1.`string getName()`**

* Funkcja odpowiedzialna za pobranie imienia gracza z wejścia w trybie tekstowym.
* Używa funkcji `nocbreak()` i `echo()`, aby umożliwić graczowi wprowadzanie tekstu bez przerwania, a także wyświetlanie wprowadzanego tekstu na ekranie.
* Funkcja czeka na naciśnięcie klawisza za pomocą `getch()`, zbierając znaki wprowadzane przez użytkownika, które są dodawane do zmiennej `name`.
* Pętla trwa do momentu, kiedy gracz naciśnie klawisz Enter (kod ASCII 10). Każdy naciśnięty znak jest dodawany do łańcucha `name`.
* Po zakończeniu wprowadzania imienia funkcja przełącza terminal z powrotem do trybu `cbreak()` i `noecho()`, aby nie wyświetlać już kolejnych wprowadzanych znaków na ekranie.
* Na końcu zwraca wprowadzone imię jako string.

**2. `ScreenChoice leaderboardUpdate(Leaderboard \*lb)`**

* Funkcja odpowiadająca za wyświetlenie ekranu aktualizacji leaderboardu po uzyskaniu przez gracza wysokiego wyniku.
* Najpierw czyści ekran (funkcja `clear()`) i odświeża go (funkcja `refresh()`).
* Następnie pobiera aktualny wynik gracza z bufora wyników (`lb->getScoreBuffer()`).
* Wyświetla komunikat gratulacyjny na ekranie za pomocą funkcji `mvprintw()`, informując gracza o uzyskaniu jednego z najlepszych wyników, wyświetlając jego wynik oraz prosząc o podanie imienia.
* Po wyświetleniu instrukcji funkcja przechodzi do wczytania imienia gracza. Używa funkcji `getName()`, która zbiera tekst z wejścia.
* Po wprowadzeniu imienia przez gracza, wynik i imię są dodawane do leaderboardu za pomocą `lb->insertScore(name, score)`.
* Na końcu funkcja zwraca wartość `LEADERBOARD`, co sugeruje powrót do ekranu leaderboardu, gdzie można zobaczyć zaktualizowaną listę wyników.

InstructionsFunctions

Funkcja instructions() wyświetla szczegółowe instrukcje dotyczące gry Boggle w terminalu i pozwala użytkownikowi zapoznać się z zasadami. Po przeczytaniu instrukcji i naciśnięciu dowolnego klawisza użytkownik wraca do głównego menu gry.