Politechnika Rzeszowska
Wydział Elektrotechniki i Informatyki

LUDO: Documentation

Wykonał:
Maciej Nabożny 173678

Player.cpp

Metody:

- `void Player::MovePiece(int player, int pawn, int steps, sf::RectangleShape board[15][15], sf::CircleShape circle[4][4])`: odpowiada za przemieszczanie pionka gracza na podstawie wyniku rzutu kostką. Sprawdza wartość wyrzuconą kostką i wykonuje odpowiednie akcje, takie jak umieszczenie pionka na planszy lub wykonanie określonych warunków, jeśli pionek jest już w grze.
- `bool Player::IsPawnInPlay(int player, int pawn, sf::CircleShape circle[4][4])`: sprawdza, czy pionek określonego gracza jest już w grze na planszy, uniemożliwiając ponowne wprowadzenie go, jeśli już się na niej znajduje.
- `bool Player::Conditions(int player, int pawn, int steps, sf::CircleShape circle[4][4])`: sprawdza różne warunki przed zezwoleniem na ruch pionka. Uwzględnia bieżącą pozycję pionka, wartość wyrzuconą kostką i zapewnia, że ruch pionka przestrzega zasad gry.
- `bool Player::SixesLimit()`: śledzi liczbę kolejnych rzutów kostką, które wyniosły sześć punktów, zwracając true, jeśli gracz wyrzucił trzy sześćki z rzędu, co ma specjalne znaczenie w grze.
- `bool Player::Checkmate(int player , int x, int y, sf::CircleShape circle[4][4])`: sprawdza, czy ruch pionka gracza prowadzi do złapania pionka przeciwnika. Jeśli tak, resetuje złapany pionek do jego pozycji początkowej.

- Metody: `void Player::getRedCords()`, `void Player::getGreenCords()`, `void Player::getYellowCords()`, `void Player::getBlueCords()`: inicjalizują początkowe współrzędne pionków dla każdego koloru (Czerwony, Zielony, Żółty, Niebieski) na planszy.
- `bool Player::IfCheckmate(int player, int x, int y, sf::CircleShape circle[4][4])`: jest podobna do metody **Checkmate**, ale jest zaprojektowana do niezależnego sprawdzania warunków złapania pionka, pozwalając na bardziej elastyczne sprawdzanie warunków łapania.
- Metody `void Player::SetSixes(int steps)` i `int Player::GetSixes() const`: zajmują się zarządzaniem licznikiem kolejnych sześciosekund gracza. **SetSixes** ustawia licznik, a **GetSixes** pobiera jego aktualną wartość.

Pola składowe klasy:

- `int sixes`: Pole to służy do śledzenia, ile razy gracz wyrzucił sześć oczek kostką podczas swojej tury. Jest wykorzystywane w metodzie **SixesLimit**, gdzie kontroluje się, czy gracz wyrzucił trzy sześćki z rzędu. W przypadku osiągnięcia tego warunku, może być podjęta odpowiednia akcja, zgodnie z zasadami gry, co zostało opisane w kodzie, którego fragment został dostarczony wcześniej.

Board.cpp

Metody:

- `void Board::InitializeBoard(sf::RectangleShape board[15][15], sf::CircleShape circle[4][4]):` inicjalizuje planszę gry, ustawiając kształty (koła) dla pionków graczy na odpowiednich pozycjach. W pętlach iterujących po graczach i pionkach ustawia właściwości graficzne, takie jak promień, grubość obramowania i kolor wypełnienia dla każdego pionka. Następnie definiuje konkretne pozycje startowe dla pionków graczy na planszy.
- `void Board::SetSpawnPosition(int player,int pawn, sf::CircleShape circle[4][4]):` ustawia pozycję startową dla określonego pionka gracza na planszy. Wykorzystuje indeksy gracza i pionka, aby pobrać odpowiednie współrzędne z wcześniej zdefiniowanej tablicy spawnPositions, a następnie przesuwa pionek na tę pozycję.
- `Cords Board::GetRespawnPoint(int player, int pawn):` zwraca współrzędne punktu respawnu dla określonego gracza i pionka, korzystając z tablicy **spawnPositions**.
- `bool Board::isPawnMovable(int player, int pawn, int steps, Cords px[61], sf::CircleShape circle[4][4]):` sprawdza, czy pionek gracza może być przesunięty o określoną liczbę kroków. Pobiera aktualną pozycję pionka oraz pozycję punktu respawnu. Następnie iteruje przez tablicę px reprezentującą pola na planszy i sprawdza, czy pionek znajduje się na danym polu. W zależności od liczby kroków i warunków gry decyduje, czy ruch jest możliwy.

Pola składowe:

- `Cords spawnPositions[4][4]:` to dwuwymiarowa tablica obiektów typu Cords, reprezentująca początkowe pozycje respawnu dla pionków graczy. Każdy gracz (indeks od 0 do 3) posiada cztery pionki (indeks od 0 do 3), a dla każdego pionka przechowywane są współrzędne jego początkowej pozycji.

Dice.cpp

Metody:

- `int Dice::Roll()`: jest odpowiedzialna za losowanie liczby całkowitej z zakresu od 1 do 6, symulując tym samym rzut kostką. Wewnątrz metody używana jest funkcja **rand()** z biblioteki **<cstdlib>**, która generuje pseudolosową liczbę całkowitą. Następnie, wynik losowania jest przeskalowany modulo 6, aby uzyskać liczbę z zakresu 0-5, a na koniec dodawana jest jedynka, aby liczby wynosiły od 1 do 6, zgodnie z kostką do gry.
  Metoda zwraca uzyskaną liczbę, która reprezentuje wynik rzutu kostką.

Game.cpp

Metody:

- `int Game::Init()`: stanowi kluczowy fragment kodu odpowiedzialny za inicjalizację i zarządzanie procesem rozgrywki w grze **LUDO**. W pierwszej kolejności, tworzy obiekt sf::RenderWindow, reprezentujący główne okno gry, o zdefiniowanych parametrach szerokości i wysokości. Następnie generuje planszę (board), pionki (circle), oraz obiekt klasy **Dice** do obsługi losowania liczb.
  Po zainicjowaniu elementów graficznych i logiki gry, metoda przechodzi do obsługi głównej pętli gry (**while (window.isOpen())**). Wewnątrz tej pętli, reaguje na zdarzenia klawiszowe, takie jak naciśnięcie przycisku ESC, umożliwiając gracza zamknięcie gry. Obsługuje także przyciski klawiatury, takie jak R (restart gry) czy Enter (potwierdzenie ruchu gracza).
  Dodatkowo, metoda prezentuje interfejs graficzny, uwzględniając ekran powitalny oraz aktualny stan rozgrywki. W trakcie gry, wyświetla informacje dotyczące wyniku, wartości wyrzuconej kostki, dostępnych ruchów dla danego gracza, czy informacje o ograniczeniach ruchu (np. limit sześciokrotnego rzutu "6" z rzędu). Dynamicznie reaguje na akcje graczy, takie jak wybór pionka do ruchu, przewijanie gracza, a także zamykanie gry.
  Metoda Init jest centralnym punktem obsługi interakcji z graczem i aktualizacji stanu gry, tworząc płynny i responsywny przebieg rozgrywki w grze LUDO.

main.cpp

Plik **main.cpp** zawiera kod źródłowy programu, zaczynając od inkluzji nagłówków, takich jak **Dice.h**, **Game.h**, **Player.h**, a także standardowych nagłówków jak **iostream**, **ctime**, **cstdlib**, i **windows.h**. Główna funkcja programu, **WinMain**, tworzy obiekt klasy **Game,** a następnie wywołuje na nim metodę **Init**, co stanowi punkt wejścia do logiki gry w LUDO.