

**Gra Linker**

Hubert Mazur, Marcin Moskała, Aleksandra Nowak, Paweł Stugient

Politechnika Warszawska

Wydział Elektryczny

Projekt realizowany w ramach przedmiotu Języki i Metodyka Programowania w roku akademickim 2019/20 semestr 2.

**Spis treści**

[**1. Definicje** 3](#_Toc40730709)

[**2. Opis aplikacji** 4](#_Toc40730710)

[Cel gry 4](#_Toc40730711)

[Reguły oraz założenia 4](#_Toc40730712)

[Sterowanie 5](#_Toc40730713)

[**3. Opis interfejsu graficznego** 6](#_Toc40730714)

[Okno aplikacji 6](#_Toc40730715)

[Menu 6](#_Toc40730716)

[Rozgrywka (panel) 7](#_Toc40730717)

[**4. Funkcjonalność** 8](#_Toc40730718)

[Nowa gra 8](#_Toc40730719)

[Przechodzenie gry 8](#_Toc40730720)

[**5. Deliverables** 9](#_Toc40730721)

[**6. Oprogramowanie gry** 10](#_Toc40730722)

[Schemat logiki gry 10](#_Toc40730723)

[Diagram klas 11](#_Toc40730724)

[Klasy 12](#_Toc40730725)

# **1. Definicje**

**Gracz** - Użytkownik aplikacji, w kontekście oprogramowania jest to ostatnia pozycja, na której zakończono rysowanie łamanej.

**Panel** - Plansza, na której rozgrywany jest poziom. Zawiera kafelki, ścieżki, wejście i wyjście oraz elementy logiczne.

**Łamana** - Linia łamana utworzona przez kolejne ruchy gracza.

**Ścieżka** - Element poziomu, po którym gracz może się przemieszczać.

**Kafelek** - Kwadrat ograniczony przez cztery ścieżki od każdej strony.

**Wejście** - Punkt początkowy (start), z którego gracz rozpoczyna rysowanie łamanej.

**Wyjście** - Punkt docelowy (meta), do którego gracz musi doprowadzić łamaną.

**Element logiczny** - Element znajdujący się na ścieżce lub w kafelku narzucający dodatkowe wymagania na poziom.

# **2. Opis aplikacji**

Linker to jednoosobowa minimalistyczna gra logiczna polegająca na przeprowadzaniu linii łamanej z punktu startowego do punktu końcowego przy spełnieniu warunków narzuconych przez dany poziom. Wraz z przechodzeniem kolejnych poziomów, wprowadzane są nowe mechaniki urozmaicające rozgrywkę oraz podnoszące poziom trudności. Dzięki poziomom, które pełnią funkcję samouczków, gracz samodzielnie odkrywa w jaki sposób działają poszczególne elementy gry.

## **Cel gry**

Celem gry jest ukończenie po kolei wszystkich poziomów. Każdy poziom to zagadka logiczna, rozwiązaniem której jest doprowadzenie linii z punktu początkowego (startu) do punktu końcowego (mety). Dodatkowo, aby rozwiązanie było uznane za poprawne, narysowana linia musi spełniać wszystkie warunki narzucone przez dany poziom.

## **Reguły oraz założenia**

* Rozgrywkę prowadzi jeden gracz.
* Rozgrywka rozpoczyna się w punkcie początkowym(wejście na panelu),z którego gracz prowadzi linię łamaną do punktu końcowego (wyjścia z panelu).
* Gracz może poruszać się jedynie po wyznaczonych przez poziom ścieżkach.
* Ruch odbywa się w jednym z czterech kierunków: góra, dół, lewo i prawo.
* Linia łamana nie może się przecinać ani zawracać (niedozwolone jest korzystanie dwa razy z tej samej ścieżki).
* Gracz ma możliwość cofnięcia się na ostatnią pozycję, poruszając się w kierunku przeciwnym do ostatniego wykonanego ruchu. Operację cofania można powtarzać do momentu, gdy gracz znajdzie się z powrotem w wejściu.
* Warunkiem koniecznym dla prawidłowego rozwiązania jest końcowa pozycja gracza na wyjściu z panelu oraz spełnienie wszystkich warunków narzuconych przez elementy logiczne znajdujące się na panelu.
* Aby rozwiązanie zostało uznane za prawidłowe, spełnione muszą być dwa warunki.
  + Gracz doprowadził łamaną do wyjścia z panelu.
  + Narysowana linia spełnia wszystkie warunki dodatkowe, narzucone przez elementy logiczne znajdujące się w danym poziomie.
* Gracz samodzielnie wysuwa wnioski o działaniu mechanik gry i wykorzystuje zdobytą wiedzę do rozwiązywania coraz trudniejszych poziomów.
* Gra kończy się, gdy wszystkie poziomy zostaną poprawnie rozwiązane.

Więcej informacji o warunkach nakładanych przez elementy logiczne znajduje się w dalszej części dokumentacji.

## **Sterowanie**

Gracz nawiguje po menu oraz po panelu gry przyciskami W, S, A oraz D. Wybór opcji w menu lub wprowadzenie rozwiązania w trakcie rozgrywki odbywa się poprzez wciśnięcie przycisku ENTER. Powrót do menu z poziomu rozgrywki odbywa się poprzez wciśnięcie przycisku ESC. Myszka oraz strzałki nie są obsługiwane.

**W trakcie rozgrywki:**

* '**W**' odpowiada za ruch w górę o połowę długości jednego kafelka.
* '**S**' odpowiada za ruch w dół o połowę długości jednego kafelka.
* '**A**' odpowiada za ruch w lewo o połowę długości jednego kafelka.
* '**D**' odpowiada za ruch w prawo o połowę długości jednego kafelka.

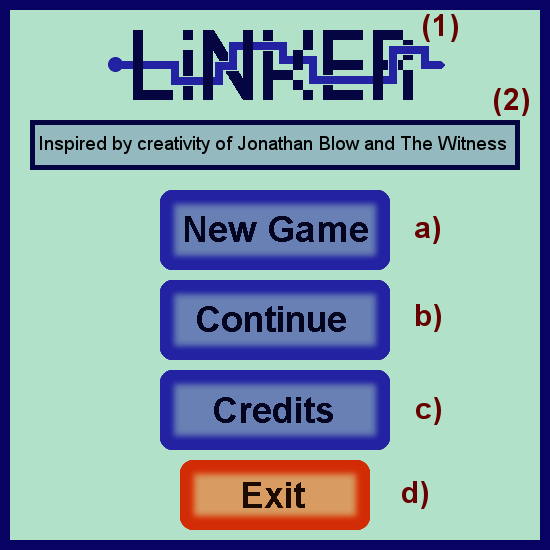
# **3. Opis interfejsu graficznego**

## **Okno aplikacji**

Gra uruchamiana jest w kwadratowym oknie o nazwie "Linker" z paskiem nawigacyjnym. Okno ma rozmiary 550 na 550 pikseli.

## **Menu**

Na samej górze menu znajduje się logo gry z jej nazwą (1). Poniżej logo widoczne jest okienko dialogowe (2), które zawiera komentarz do obecnie wybranej pozycji. Menu posiada również tryb neutralny, który jest widoczny jedynie przy włączeniu gry, zanim zaczniemy nawigować między opcjami.



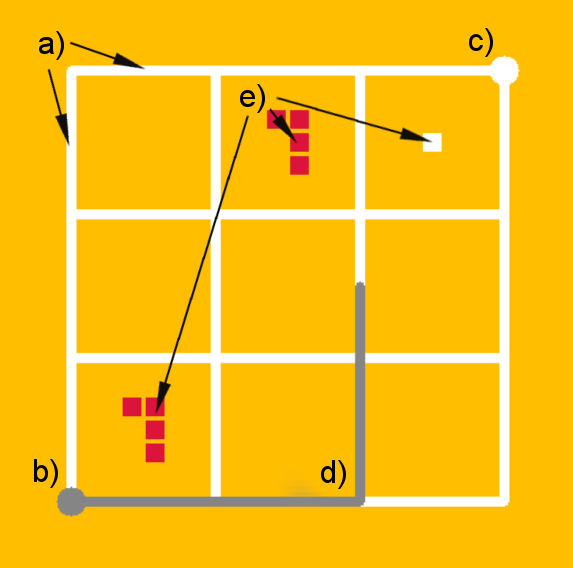
Menu gry posiada następujące opcje:

* **New Game** - rozpoczyna nową rozgrywkę poprzez załadowanie pierwszego poziomu oraz usunięcie poprzednich postępów. Przed wyświetleniem poziomu pokazany jest schemat sterowania w grze.
* **Continue** - wznawia niedokończoną rozgrywkę na poziomie, w którym gra została ostatnio zamknięta.
* **Credits** - wybranie tej opcji wyświetli autorów gry oraz specjalne podziękowania.
* **Exit** - zamyka aplikację.

## **Rozgrywka (panel)**

Każdy poziom posiada tło w kolorze charakterystycznym dla danej kategorii zagadek oraz panel, który znajduje się na środku okna. Na panel składają się:

1. **ścieżki** - linie tworzące kształt siatki, po których może poruszać się łamana.
2. **wejście** - kółko w kolorze łamanej rysowanej przez gracza, znajdujące się w lewym dolnym rogu panelu
3. **wyjście** - kółko w takim samym kolorze co ścieżki, znajdujące się prawym górnym rogu panelu.
4. **łamana** - szara linia łamana wychodząca z pozycji startowej. Na załączonym obrazku widoczny jest kształt łamanej po wykonaniu siedmiu ruchów: 4 razy w prawo i 3 razy w górę.
5. **elementy logiczne** - symbole, które znajdują się wewnątrz kafelków bądź bezpośrednio na ścieżkach lub skrzyżowaniach tych ścieżek.



Kolor oraz położenie wyżej wymienionych elementów panelu zostały podane na podstawie przykładu. Te własności nie są wiążące. Na przykład, wejście i wyjście mogą znaleźć się w dowolnym miejscu na panelu, dopóki istnieje między nimi ścieżka umożliwiająca graczowi rozwiązanie poziomu.

Sam panel może mieć dowolny kształt składający się z kafelków, których maksymalna ilość w osi poziomej i pionowej nie może przekroczyć pięciu. Ograniczenie to jest spowodowane metodą skalowania poziomów do rozmiarów okna.

# **4. Funkcjonalność**

## **Nowa gra**

Aby rozpocząć rozgrywkę, użytkownik musi wybrać opcję “Nowa Gra” w menu gry. Po wybraniu tej opcji aplikacja tworzy na podstawie danych w pliku poziom i wyświetla go.

## **Przechodzenie gry**

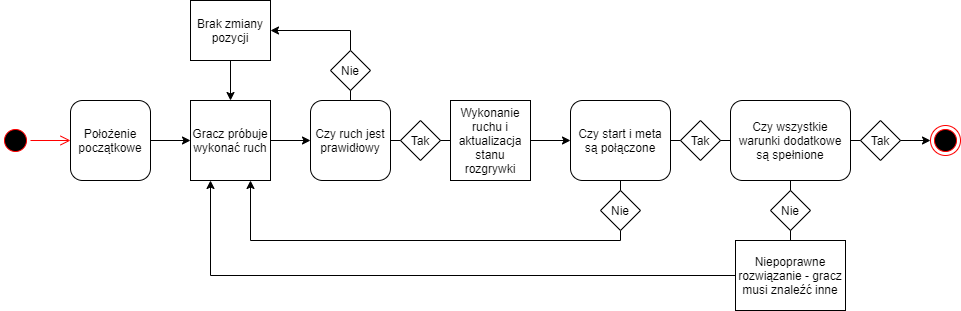
Gracz przechodzi poziomy doprowadzając linię ze startu do mety. Gdy linia połączy punkt początkowy z punktem końcowym program upewnia się, czy wszystkie warunki dodatkowe są spełnione. Jeśli tak, to ładowany jest następny poziom, a jeśli nie gracz musi znaleźć inne rozwiązanie.

Proces jest kontynuowany, dopóki wszystkie poziomy nie zostaną rozwiązane lub gracz samodzielnie nie zakończy rozgrywki wciskając klawisz ESC.

Schemat działania gry:



Schemat przechodzenia jednego poziomu:

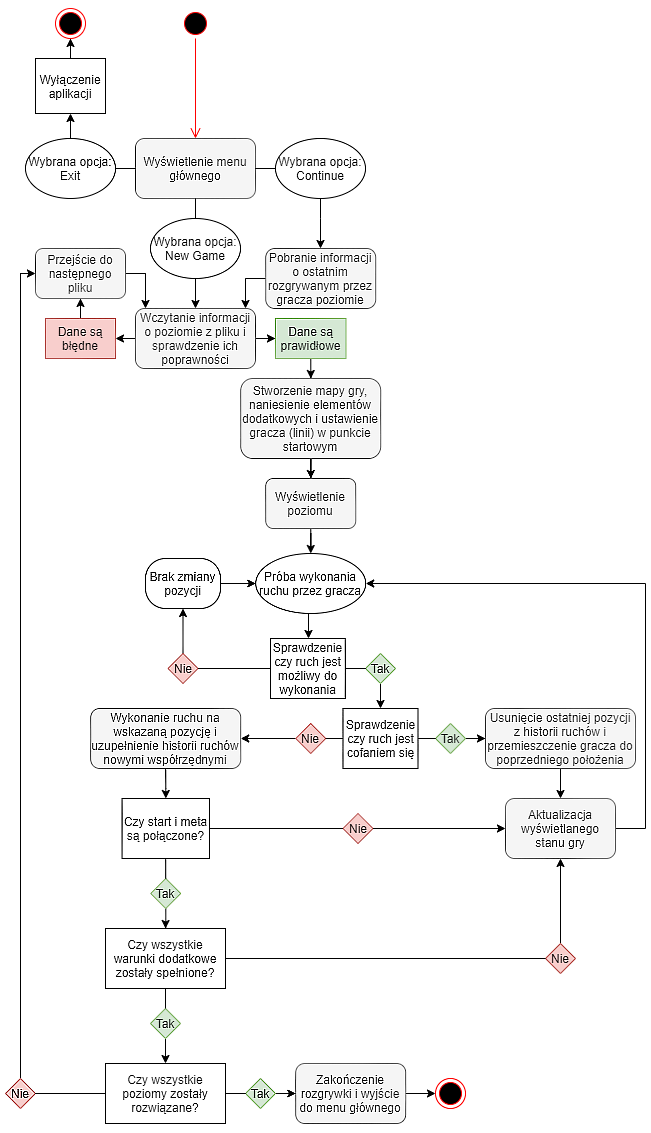


# **5. Deliverables**

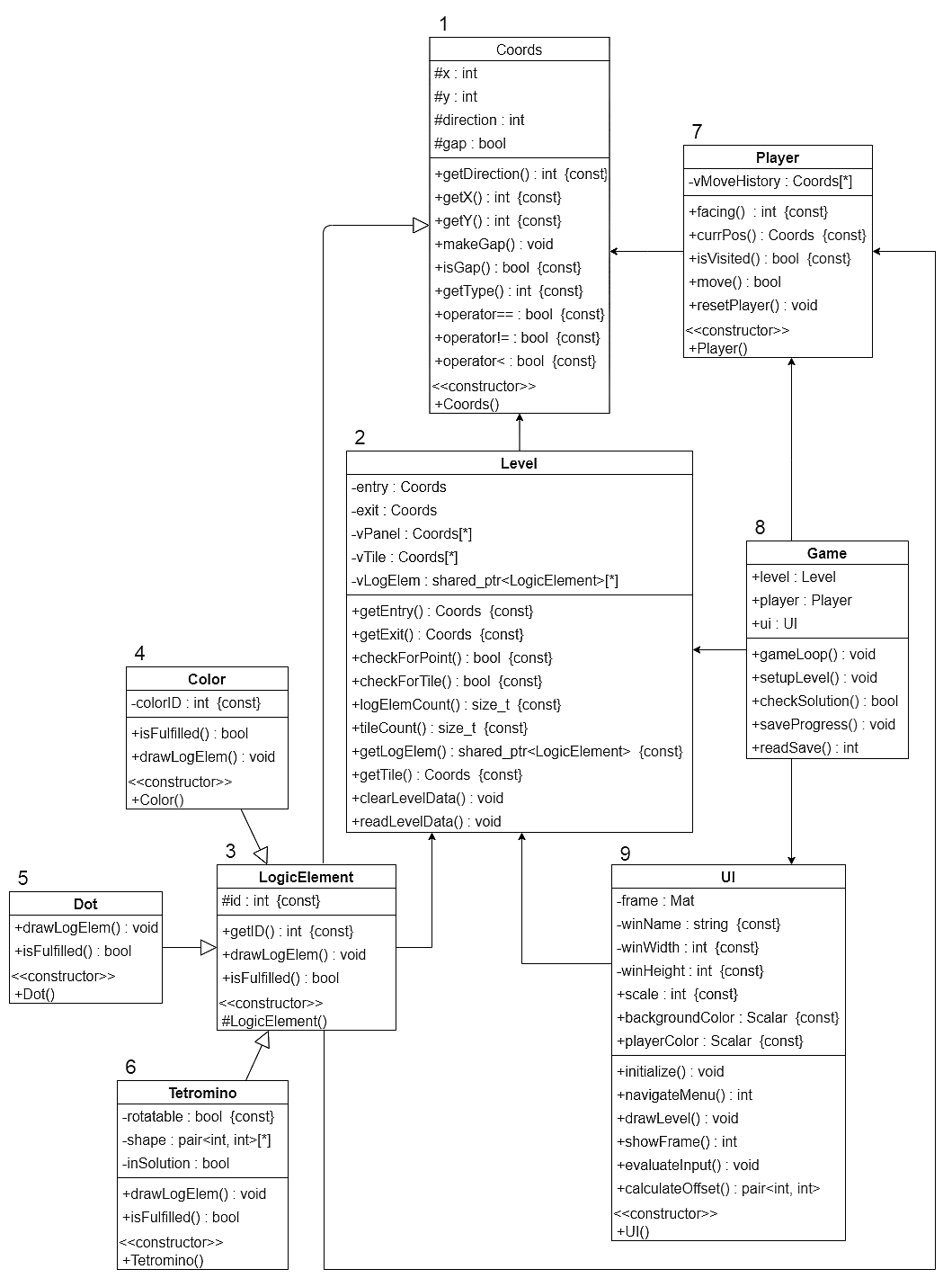
* Moduł odpowiedzialny za generowanie poziomu
  + Dekoduje informacje zawarte w pliku i zapisuje je w klasie Level.
* Moduł logiki poruszania się gracza
  + Sprawdza poprawność ruchów według założeń gry.
  + Wykrywa akcję próby cofnięcia ostatnio wykonanego ruchu.
  + Wprowadza zmiany w położeniu końca linii.
  + Prowadzi historię ruchów gracza w celu wyznaczenia kształtu linii.
* Moduł sprawdzania poprawności rozwiązania
  + Wykrywa czy linia łączy początek i koniec poziomu.
  + Sprawdza czy spełnione zostały dodatkowe warunki logiczne poziomu.
  + Zmienia stan rozgrywki adekwatnie do poprawności rozwiązania:
    - jeśli rozwiązanie jest prawidłowe pozwala załadować następny poziom.
    - jeśli rozwiązanie jest niepoprawne cofa gracza do początkowej pozycji, resetuje stan poziomu.
* Graficzny interfejs użytkownika
  + Wyświetla menu główne gry przy uruchomieniu gry oraz po wyjściu z poziomu.
  + Wyświetla wszystkie elementy składowe poziomu.
  + Wyświetla linię, którą porusza gracz oraz aktualizuje jej kształt na mapie po każdym wykonanym ruchu.
* Moduł interpretujący dane wejściowe z klawiatury
  + Rozpoznaje wciśnięcie klawiszy odpowiedzialnych za ruch, akceptację rozwiązania, wyjście z poziomu.
* Działająca gra
  + Moduły współpracują ze sobą w sposób umożliwiający prowadzenie rozgrywki w wyświetlonym po uruchomieniu aplikacji oknie.

# **6. Implementacja**

## **Schemat logiki gry**



## **Diagram klas**



## **Klasy**

1. **Coords**

Pola

* protected int **x**
* protected int **y**

Zmienne przechowujące wartości współrzędnych wszelkich elementów tj. obecnej pozycji gracza, przeszłych/odwiedzonych przez gracza pozycji, wszystkich elementów składowych panelu.

* protected int **direction**

W przypadku punktów opisujących przeszłą bądź teraźniejszą pozycję gracza, wskazuje kierunek, w którym gracz ostatnio wykonał ruch przemieszczając się na tę pozycję poprzez zawarcie wartości ASCII znaku z listy: w, a, s, d. Wartość w polu startowym to 0.

* protected bool **gap**

Prawdziwa wartość tego pola sprawia, że dana krawędź przestaje być dostępna dla gracza i jest specjalnie oznaczona na panelu. Domyślna wartość: false.

Metody

* public int **getDirection()**

Zwraca wartość pola **direction**.

* public int **getX()**
* public int **getY()**

Zwracają wartości pól odpowiednio **x** i **y**.

* public void **makeGap()**

Zmienia wartość pola **gap** na prawdziwą. Wykorzystywane tylko przy wczytywaniu danych poziomu.

* public bool **isGap()**

Zwraca wartość pola **gap**.

* public int **getType()**

Zwraca zakodowaną informację o typie kształtu reprezentowanego przez dany punkt w zbiorze punktów charakteryzującym kształt panelu. Wartość 0 oznacza skrzyżowanie krawędzi, 1 - pionową krawędź, 2 - poziomą krawędź.

* public bool**operator==**
* public bool**operator!=**
* public bool**operator<**

Przeciążenia operatorów.

* public **Coords(int x = -1, int y = -1, int dir = -1)**

Inicjalizuje wszystkie pola dla tworzonego obiektu.

1. **Level**

Pola

* private Coords **entry**

Obiekt przechowujący informacje o współrzędnych wejścia do danego poziomu.

* private Coords **exit**

Obiekt przechowujący informacje o współrzędnych wyjścia z danego poziomu.

* private vector<Coords> **vPanel**

Zbiór punktów opisujący kształt danego poziomu.

* private vector<Coords> **vTile**

Zbiór punktów opisujący położenia kafelków dla danego poziomu. Wykorzystywany w algorytmach sprawdzających poprawność rozwiązań.

* private vector<shared\_ptr<LogicElement>> **vLogElem**

Wektor wskaźników na elementy logiczne poziomu.

Metody

* public Coords **getEntry()**
* public Coords **getExit()**

Metody pobierające współrzędne wejścia i wyjścia danego poziomu.

* public bool **checkForPoint(const Coords &p)**

Sprawdza czy punkt p znajduje się w wektorze **vPanel**. Jeżeli tak, zwraca true, w przeciwnym razie - false.

* public bool **checkForTile(const Coords &p)**

Sprawdza czy punkt p znajduje się w wektorze **vTile**. Jeżeli tak, zwraca true, w przeciwnym razie - false.

* public size\_t **logElemCount()**

Metoda zwracająca ilość elementów logicznych na obecnym poziomie.

* public size\_t **tileCount()**

Metoda zwracająca ilość kafelków, z których składa się obecny panel.

* public shared\_ptr<LogicElement> **getLogElem(int i)**

Zwraca wskaźnik na i-tej pozycji w wektorze **vLogElem**.

* public Coords **getTile(int i)**

Zwraca punkt na i-tej pozycji w wektorze **vTile**.

* public void **clearLevelData()**

Zwalnia zarezerwowaną pamięć, usuwa zawartość wektorów **vTile**, **vLogElem**, **vPanel**. Nadpisuje wejście oraz wyjście domyślnymi obiektami klasy **Coords**.

* public void **readLevelData()**

Wczytuje zawartość plików tekstowych, a następnie uzupełnia wszystkie pola klasy o nowe dane dekodując informacje o poziomie w nich zawarte.

1. **LogicElement**

Zmienne

* protected int **id**

Zmienna przechowująca numer odpowiadający danemu typowi elementu logicznego.

Metody

* public int **getID()**

Metoda pobierająca numer id danego elementu logicznego.

* public void **drawLogElem()**

Metoda odpowiedzialna za naniesienie elementu logicznego na panel.

* public bool **isFulfilled()**

Metoda sprawdzająca czy wymuszony przez dany element logiczny warunek został spełniony.

* public **LogicElement()**

Konstruktor klasy LogicElement.

1. **Color**

Zmienne

* private int **colorID**

Zmienna przechowująca liczbę odpowiadającą danemu kolorowi.

Metody

* public bool**isFulfilled()**

W przypadku elementu logicznego "Color" metoda isFulfilled() ma sprawdzić czy pola zawierające różne kolory są od siebie oddzielone łamaną.

* public void**drawLogElem()**

W przypadku elementu logicznego "Color" metoda drawLogElem() ma narysować wewnątrz pola określony kształt o danym kolorze.

* public**Color()**

Konstruktor klasy Color.

1. **Dot**

Metody

* public bool **isFulfilled()**

W przypadku elementu logicznego "Dot" metoda isFulfilled() sprawdza czy punktzawierający kropkę został odwiedzony przez gracza.

* public void **drawLogElem()**

W przypadku elementu logicznego "Dot" metoda drawLogElem() rysuje w danym punkcie kropkę.

* public **Dot()**

Konstruktor klasy Dot.

1. **Tetromino**

Zmienne

* private bool**rotatable**

Zmienna przechowująca informacje o tym, czy danetetromino może być obrócone w rozwiązaniu.

* private vector<pair<int, int>>**shape**

Wektor przechowujący informacje o kształcie tetromina.

* private bool **inSolution**

Zmienna przechowująca informacje o tym, czy symbol tetromino znajduje się wewnątrz obszaru określonego narysowaną przez gracza łamaną.

Metody

* public bool **isFulfilled()**

W przypadku elementu logicznego "Tetromino" metoda isFulfilled() sprawdza czy symbol tetromina znajduje się wewnątrz obszaru określonego narysowaną przez gracza łamaną.

* public void **drawLogElem()**

W przypadku elementu logicznego "Tetromino" metoda drawLogElem() rysuje w danym polu symbol jakiegoś konkretnego tetromina.

* public**Tetromino()**

Konstruktor klasy Tetromino.

1. **Player**

Zmienne

* private vector<Coords>**vMoveHistory**

Wektor przechowujący informacje o wszystkich punktach panelu, które odwiedził gracz (historia ruchów).

Metody

* public int**facing()**

Metoda zwracająca ostatni znany kierunek patrzenia gracza.

* public Coords**currPos()**

Metoda zwracająca współrzędne obecnego położenia gracza.

* public bool**isVisited()**

Metoda odpowiedzialna za określenie czy punkt został odwiedzony przez gracza. Jeśli współrzędne danego punktu znajdują się w historii ruchów zwraca prawdę, a jeśli nie - fałsz.

* public bool**move()**

Metoda sprawdzająca czy ruch, który chce wykonać gracz jest możliwy oraz czy będzie on cofaniem czy nie. Jeśli określone przez gracza działanie jest prawidłowe to wykonuje ruch - ustawia nowe współrzędne i kierunek oraz uzupełnia historię ruchów.

* public void**resetPlayer()**

Metoda odpowiedzialna za wykonywanie ruchu cofania. Usuwa najnowszy element z wektora historii ruchów, a następnie ustawia pozycję i kierunek patrzenia gracza na te z ruchu poprzedniego.

* public**Player()**

Konstruktor klasy Player.

1. **Game**

Zmienne

* public Level**level**

Obiekt przechowujący informacje o poziomie.

* public Player**player**

Obiekt przechowujący informacje o stanie gracza.

* public UI**ui**

Obiekt przechowujący informacje o interfejsie graficznym.

Metody

* public void**gameLoop()**

Metoda prowadząca główną pętlę programu wewnątrz menu głównego lub podczas rozgrywki.

* public void**setupLevel()**

Metoda odpowiedzialna za wczytanie danych o poziomie z pliku, ustawienie gracza w położeniu początkowym i nadpisanie postępów użytkownika.

* public bool**checkSolution()**

Metoda kontrolująca poprawność rozwiązania poprzez sprawdzenie czy start i meta zostały połączone oraz czy wszystkie warunki dodatkowe zostały spełnione.

* public void**saveProgress()**

Metoda, która przechowuje i uzupełnia informacje o ostatnim poziomie, do którego doterł gracz. Pozwala na kontynuowanie rozgrywki od momentu, w którym została ostatnio zakończona.

* public int **readSave()**

Metoda, wczytująca ostatni rozgrywany przez gracza poziom, jeśli wybrana została opcja Continue w menu głównym.

1. **UI**

Zmienne

* private Mat **frame**
* private string**winName**
* private int**winWidth**
* private int**winHeight**

Zmienne przechowujące informacje o rodzaju, rozmiarach i kształcie okna, w którym uruchamiana jest aplikacja.

* public int**scale**
* public Scalar**backgroundColor**
* public Scalar**playerColor**

Zmienne przechowujące informacje o tym jak mają wyglądać tworzone obiekty, pola, ścieżki, rysowana przez gracza łamana oraz jaki kolor ma mieć tło poziomu.

Metody

* public void**initialize()**

Metoda, która wyświetla menu gry w opcji neutralnej.

* public int **navigateMenu()**

Metoda, która w zależności od wybranej opcji wyświetla odpowiedni stan menu.

* public void**drawLevel()**

Metoda odpowiedzialna za narysowanie i wyświetlenie poziomu na podstawie informacji zawartych w pliku.

* public int**showFrame()**

Metoda wyświetlająca framerate podczas działania aplikacji.

* public void**evaluateInput()**

Metoda interpretująca informacje wprowadzone z klawiatury jako próbę wykonania jakiejś akcji. Na przykład wciśnięcie klawisza W - próba wykonania ruchu w górę.

* public pair<int, int>**calculateOffset()**

Metoda odpowiedzialna za prawidłowe ustawienie panelu danego poziomu wewnątrz okna aplikacji.

* public**UI()**

Konstruktor klasy UI.