



DOKUMENTACJA TECHNICZNA

Gra „Chińczyk”

[O projekcie](#)

Projekt przedstawia grę komputerową „Chińczyk”

Stępień Kamil
GitHub: KamilStepien

Toczek Dominik
GitHub: Toczq

Rup Adam
GitHub: AdamRup

Spis treści

1. Opis Projektu	2
2. Zasady Gry	2
3. Charakterystyka Interfejsu Użytkownika.....	2
4. Charakterystyka Biblioteki Klas „ChinczykLib”	3
4.1 Diagram UML.....	3
4.2 Opis Diagramu UML	3
4.3 Klasa Dice.....	3
4.3.1 Pola i właściwości	3
4.3.2 Metody	3
4.4 Klasa Pawn.....	3
4.4.1 Pola i właściwości	3
4.4.2 Metody	4
4.5 Klasa Player	4
4.5.1 Pola i właściwości	4
4.5.2 Metody	4
4.6 Klasa HumanPlayer.....	5
4.6.1 Pola i właściwości	5
4.6.2 Metody	5
4.7 Klasa ComputerPlayer	5
4.7.1 Pola i właściwości	5
4.7.2 Metody	5
4.8 Klasa Point	6
4.8.1 Pola i właściwości	6
4.8.2 Metody	6
4.9 Klasa GameBoard	6
4.9.1 Pola i właściwości	6
4.9.2 Metody	6
4.10 Klasa GameModel	6
4.10.1 Pola i właściwości	6
4.10.2 Metody	6

1. Opis Projektu

2. Zasady Gry

Gra dla 2-4 osób, w której celem graczy jest przejechanie dookoła planszy czterema pionkami z pozycji początkowych na końcowe. Pierwszy gracz, który tego dokona, wygrywa.

Wygląd planszy z początkowym ustawieniem pionów:

[tu wstawić zdjęcie planszy]

Ruch w grze polega na przesunięciu na planszy dowolnego z własnych pionów o tyle pól, ile oczek zostanie wyrzuconych na kostce (1-6). Jeśli na docelowym polu znajduje się pion przeciwnika, to jest on usuwany i wraca na jedną z początkowych pozycji, a jeśli znajduje się tam własny pion, to taki ruch jest niedozwolony.

Wyrzucając 6 oczek na kostce, gracz wykonuje dodatkowy ruch, ale nie może wykonać pod rząd więcej niż 3 takie ruchy.

Aby wyjechać pionem z pozycji początkowej konieczne jest wyrzucenie „szóstki”. Dowolny pion jest wtedy umieszczany na pierwszym polu planszy oznaczonym w kolorze pionów gracza. Jeśli gracz posiada wszystkie swoje piony na pozycjach początkowych to przysługują mu trzy próby na wyrzucenie „szóstki” wymaganej do wyjechania dowolnym z pionów.

3. Charakterystyka Interfejsu Użytkownika

Interfejs Użytkownika składa się z 4 okien aplikacji WPF – okno główne, okno z zasadami gry, okno konfiguracji nowej gry oraz okno właściwe z planszą gry.

Po uruchomieniu aplikacji pojawia się okno startowe, w którym użytkownik ma do wyboru 3 opcje:

- Nowa gra
- Zasady
- Wyjście

Po wybraniu przycisku „Wyjście” program kończy swoje działanie i aplikacja zostaje zamknięta.

Opcja „Zasady” przenosi użytkownika do nowego okna, w którym jest opis zasad gry w „Chińczyka”. Po zaznajomieniu się z regułami gry, użytkownik może powrócić do głównego menu poprzez kliknięcie przycisku „powrót”.

Gdy użytkownik kliknie na przycisk „Nowa gra” aplikacja otworzy nowe okno, w którym użytkownik będzie mógł wybrać ilość graczy oraz poziom trudności, a następnie rozpocząć grę poprzez kliknięcie przycisku „Start”.

Po rozpoczęciu gry zostanie wyświetlona plansza gry, na której znajdują się pionki graczy. Na środku planszy znajduje się przycisk służący do „rzutu kostką”. W każdej fazie gry, użytkownik może przerwać swoją rozgrywkę poprzez wybranie przycisku „Zakończ”, który przeniesie użytkownika do okna głównego menu.

4. Charakterystyka Biblioteki Klas „ChinczykLib”

4.1 Diagram UML

4.2 Opis Diagramu UML

4.3 Klasa Dice

Klasa definiująca obiekt kostki do gry.

4.3.1 Pola i właściwości

- **ImgPath** – Właściwość przechowująca ścieżkę do zdjęcia kostki z odpowiednią wartością.
Typ danych: string
- **Value** – Właściwość przechowująca wartość na kostce do gry (liczba oczek).
Typ danych: int

4.3.2 Metody

- **SetImgPath()** – Metoda ustawiająca ścieżkę do zdjęcia z odpowiednią wartością na kostce.
Typ zwracany: void
- **Roll()** – Metoda symulująca rzut kostką (losowy wybór wartości kostki z zakresu 1-6). Przypisuje wylosowaną wartość do właściwości Value.
Typ zwracany: void

4.4 Klasa Pawn

Klasa definiująca obiekt pionka w grze.

4.4.1 Pola i właściwości

- **field**
Typ danych: int
- **path** – Zmienna przechowująca ścieżkę, po której pionek przemieszcza się po planszy
Typ danych: Point[]
- **startPosition** – Zmienna przechowująca punkt początkowy na planszy od którego rozpoczyna się rozgrywka
Typ danych: Point
- **IsLeaveTheBase**
Typ danych: bool
- **IsActive**
Typ danych: bool
- **ImgPath** – Właściwość przechowująca ścieżkę do zdjęcia pionka.
typ danych: string
- **Position** – Właściwość przechowująca pozycję pionka na planszy.
Typ danych: Point
- **PawnColor** – Właściwość przechowująca kolor pionka.
Typ danych: string
- **NumberPawn** – Właściwość przechowująca unikalny numer pionka.
Typ danych: int

4.4.2 Metody

- **Pawn(int, Point, Point[], Point, string, string)** – Konstruktor obiektu Pawn 6-argumentowy.
Parametry:
 - int_NumberPawn
 - Point_Position
 - Point[]_Path
 - Point_StartPosition
 - string_PawnColor
 - string_ImgePath
- **Move(int)** – Metoda przesuwająca pionek o daną liczbę pól na planszy.
Parametry:
 - int_NewPositionTyp zwracany: void
- **ReturnNewPosition(int)** – Metoda sprawdzająca, czy ruch pionka o daną liczbę pól jest możliwy i zwracająca nową pozycję pionka.
Parametry:
 - int_NewPositionTyp zwracany: Point
- **LeaveTheBase()** – Metoda przesuwająca pionek z pozycji bazowej na pozycję początkową(pierwsze pole na planszy).
Typ zwracany: void

4.5 Klasa Player

Klasa abstrakcyjna definiująca gracza.

4.5.1 Pola i właściwości

- **Name** – Właściwość przechowująca nazwę gracza.
Typ danych: string
- **Number** – Właściwość przechowująca unikalny numer gracza
Typ danych: int
- **pawns** – Zmienna przechowująca pionki gracza
Typ danych: Pawns[]
- **dice** – zmienna przechowująca kostkę do gry
Typ danych: Dice

4.5.2 Metody

- **MovePawn(Pawn)** – Metoda abstrakcyjna przesuwająca wskazany pionek gracza na planszy
Parametry:
 - Pawn pawnTyp zwracany: void
- **SetNumber(int)** – Metoda ustawiająca unikalny numer gracza
Parametry:
 - int_playerNumberTyp zwracany: void

- **SetName(string)** – Metoda ustawiająca imię/nick gracza
Parametry:
- string playerName
Typ zwracany: void

4.6 Klasa HumanPlayer

Klasa definiująca obiekt gracza użytkownika i dziedzicząca po klasie abstrakcyjnej Player.

4.6.1 Pola i właściwości

-

4.6.2 Metody

- **HumanPlayer(string, int, Point[], Point[], Point, Dice)** – Konstruktor obiektu HumanPlayer 6 argumentowy
Parametry:
- string playerName
- int playerNumber
- Point[] pawnPosition
- Point[] pawnPath
- Point pawnStart
- Dice dice
- **MovePawn(Pawn)** – Metoda przesuująca pionek gracza na planszy
Parametry:
- Pawn pawn
Typ zwracany: void
- **SelectPawn(int)** – Metoda zwracająca pionek, który ma zostać przesunięty
Parametry:
- int pawnNumber
Typ zwracany: Pawn

4.7 Klasa ComputerPlayer

Klasa definiująca obiekt gracza komputerowego

4.7.1 Pola i właściwości

-

4.7.2 Metody

- **ComputerPlayer(int, Point[], Point[], Point, Dice)** – Konstruktor obiektu ComputerPlayer 5-argumentowy
Parametry:
- int playerNumber
- Point[] pawnPosition
- Point[] pawnPath
- Point pawnStart
- Dice dice
- **Play()** – Metoda wywołująca inne metody dla obiektu ComputerPlayer i przyspieszająca realizację ruchu gracza komputerowego
Typ zwracany: void

- **MovePawn(Pawn)** – Metoda przesuująca pionek gracza komputerowego na planszy
Parametry:
- Pawn pawn
Typ zwracany: void
- **SelectPawn()** – Metoda automatycznie wybierająca pionek gracza komputerowego
Typ zwracany: Pawn

4.8 Klasa Point

Klasa definiująca pozycję na planszy.

4.8.1 Pola i właściwości

- **X** - Zmienna przechowująca współrzędną X na planszy
Typ danych: int
- **Y** – Zmienna przechowująca współrzędną Y na planszy
Typ danych: int

4.8.2 Metody

- **Point(int, int)** – Konstruktor obiektu Point 2-argumentowy
Parametry:
- int x
- int y
- **Equals** – Metoda porównująca 2 obiekty typu Point
Parametry:
- Point other
Typ zwracany: bool

4.9 Klasa GameBoard

Klasa definiująca obiekt planszy gry.

4.9.1 Pola i właściwości

-

4.9.2 Metody

-

4.10 Klasa GameModel

Klasa definiująca logikę gry.

4.10.1 Pola i właściwości

-

4.10.2 Metody

-