# Šachy

## MainWindow.xaml.cs

### Přehled

Třída MainWindow je hlavní třídou aplikace, která:

- Inicializuje grafické rozhraní šachovnice.
- Reaguje na vstupy z klávesnice a myši.
- Aktualizuje grafiku dle změn ve hře.

---

### Konstruktor MainWindow()

- Inicializuje komponenty UI (*InitializeComponent()*).
- Nastavuje obsluhu události pro klávesnici.
- Vytváří vizuální mřížku (8x8 polí).
- Volá *SetupBoard()* pro inicializaci figur.
- Vykresluje každé pole a přiřazuje k němu obrázek dle typu a barvy figurky.

---

### Klávesové zkratky (MainWindow_KeyDown)

- **S**: Krátká rošáda (bílý nebo černý hráč).
- **L**: Dlouhá rošáda (bílý nebo černý hráč).

Po provedení rošády se UI zaktualizuje voláním *UpdateBoard()*.

---

### Událost kliknutí na políčko (Tile_MouseLeftButtonDown)

1. Uloží první kliknutí (výběr figurky).
2. Po druhém kliknutí se provede pokus o tah.
3. Ověří se legálnost tahu pomocí *IsMoveLegal()*.
4. Pokud je tah platný:
    - Provádí se tah figurky.
    - Provádí se případná proměna pěšce (*TryPromote()*).
    - Mění se hráč na tahu.

- Volá se *UpdateBoard()*.

5. Pokud dojde k matu, zobrazí se MessageBox a aplikace se ukončí.

---

## Metoda UpdateBoard()

- Znovu vykreslí celou šachovnici a figury podle aktuálního stavu GameBoard.
- Nastaví obrázky figurek dle jejich typu a barvy.

---

## Cesty k obrázkům figur

Používají URI ve formátu:

*pack://application:,,,/Pictures/wk.png*

kde např. *wk.png* označuje bílého krále (white king).

---

## Typy figur (enum ChessPieceName)

- Pawn (pěšec)
- Knight (jezdec)
- Bishop (střelec)
- Rook (věž)
- Queen (dáma)
- King (král)

---

## Barvy hráčů

Používá se enum Color s hodnotami:

- Color.White
- Color.Black

---

## Poznámky

- Všechny figury jsou vykreslovány pomocí WPF Image komponenty.
- Herní logika je oddělena do třídy Board, která musí obsahovat metody jako IsMoveLegal, Move, TryPromote, ShortCastling, LongCastling, FindCheckmate atd.

---

# ChessPiece.cs

Abstraktní třída reprezentující šachovou figuru:

```
abstract class ChessPiece
{
    public abstract ChessPieceName Name { get; }
    public abstract Color Color { get; }
    public bool WasAlreadyMoved { get; set; }
    public abstract bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board);
    public abstract bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board,
    Color ToMove, bool ignoreCheck = false);
    public abstract ChessPiece Clone();
}
```

## Popis členů

- **Name**: Název figurky (typ).
- **Color**: Barva figurky.
- **WasAlreadyMoved**: Indikuje, zda se figurka již pohnula (např. důležité pro rošádu).
- **Move(...)**: Provede tah, pokud je platný.
- **IsMoveLegal(...)**: Ověřuje, zda je daný tah legální.
- **Clone()**: Vrátí kopii dané figurky.

# Board.cs

Tato třída reprezentuje šachovnici a poskytuje funkce pro inicializaci, zjištění šachové situace, rošády a kontrolu matu.

---

## Vlastnosti

*public ChessPiece[,] GameBoard { get; set; }*

- 2D pole představující aktuální stav šachovnice (8x8).
- Každý prvek může být null nebo instancí potomka ChessPiece.

*public Color ToMove { get; set; } = Color.White;*

- Určuje, kdo je na tahu (Color.White nebo Color.Black).

---

## Konstruktory

*public Board()*

- Inicializuje prázdnou šachovnici (8x8).

---

## Setup metody

*public void SetupBoard()*

- Inicializuje výchozí rozmístění bílých a černých figur.

*public void SetupWhite()*

- Umístí bílé figury na šachovnici dle standardního rozestavení.

*public void SetupBlack()*

- Umístí černé figury na šachovnici dle standardního rozestavení.

---

## Pomocné metody

*public ChessPiece[,] Copy2DArray(ChessPiece[,] original)*

- Vrací hlubokou kopii 2D pole figur.
- Užitečné pro simulaci tahů.

---

## Kontrola šachu

*public bool IsWhiteKingInCheck(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Ověří, zda bílý král bude v šachu po simulovaném tahu z (fx, fy) na (sx, sy).
- board: aktuální stav desky.

*public bool IsBlackKingInCheck(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Totéž pro černého krále.

*private bool IsKingInCheck(Color color, ChessPiece[,] board)*

- Obecná verze pro zjištění, zda je král dané barvy v šachu.

---

## Rošáda

*public void LongCastling(Color c)*

- Provede dlouhou rošádu pro barvu c, pokud jsou splněny podmínky.
- Kontroluje, zda se král a věž již pohnuli a zda jsou mezi nimi prázdná pole.

*public void ShortCastling(Color c)*

- Provede krátkou rošádu pro barvu c.

## Kontrola matu

*public Color? FindCheckmate(Color color)*

- Zjistí, zda je král dané barvy v matové situaci.
- Prohledá všechny možné tahy daného hráče a kontroluje, zda existuje tah, který vyvede krále z šachu.
- Vrací null, pokud není mat; jinak barvu poraženého hráče.

# Pawn.cs

Tato třída reprezentuje pěšce v šachové hře. Dědí z abstraktní třídy ChessPiece a implementuje její specifické chování – pohyb, proměnu (promoci) a ověřování legality tahu.

## Konstruktor

*public Pawn(Color color)*

- Inicializuje pěšce s danou barvou (Color.White nebo Color.Black).

## Vlastnosti

*public override ChessPieceName Name { get => ChessPieceName.Pawn; }*

- Vrací název typu figurky – v tomto případě Pawn.

*public override Color Color => _color;*

- Vrací barvu pěšce.

*public bool WasAlreadyMoved = false;*

- Určuje, zda se pěšec již někdy pohnul (důležité pro možnost dvojitého tahu při prvním pohybu).

## Move(...)

*public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Jednoduše provede tah figurky na nové souřadnice, bez kontroly legality (to zajišťuje IsMoveLegal).
- Vrací vždy true.

---

## Clone()

*public override ChessPiece Clone()*

- Vrací klon pěšce se stejnou barvou a hodnotou WasAlreadyMoved.

---

## TryPromote(...)

*public void TryPromote(int x, int y, ChessPiece[,] board)*

- Pokusí se pěšce na dané pozici povýšit na dámu (Queen), pokud dosáhne poslední řady:
    - White: řada 7
    - Black: řada 0

---

## IsMoveLegal(...)

*public override bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board, Color ToMove, bool ignoreCheck)*

- Ověřuje legalitu tahu:
    - Kontroluje, jestli tah odpovídá barvě hráče na tahu.
    - Simuluje tah a zjišťuje, zda by hráč nenechal svého krále v šachu.
    - Umožňuje:
        - klasický pohyb o 1 nebo 2 pole vpřed (pokud ještě nebyl pohybován),
        - braní figurky diagonálně,
        - blokování pohybu jinou figurkou.
- Pokud je ignoreCheck == true, ignoruje kontrolu šachu (např. pro simulace).

# Knight.cs

Třída Knight reprezentuje jezdce v šachové hře. Dědí z abstraktní třídy ChessPiece a implementuje specifické chování pro pohyb jezdce.

## Konstruktor

*public Knight(Color color)*

- Inicializuje jezdce s danou barvou (White nebo Black).

## Vlastnosti

*public override ChessPieceName Name { get => ChessPieceName.Knight; }*

- Vrací název figurky – Knight.

*public override Color Color => _color;*

- Vrací barvu jezdce.

## Move(...)

public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)

- Přesune figurku na nové souřadnice (bez validace legality tahu).
- Nastaví board[sx, sy] na jezdce a board[fx, fy] na null.
- Vždy vrací true.

## Clone()

public override ChessPiece Clone()

- Vrací nový objekt Knight se stejnou barvou.
- Nepřenáší žádné další stavy, protože jezdec nemá atributy jako WasAlreadyMoved.

## IsMoveLegal(...)

*public override bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board, Color ToMove, bool ignoreCheck)*

- Ověřuje legalitu tahu:
    - Kontroluje, zda figurka patří hráči na tahu.

    - Ověří, že tah nezpůsobí šach vlastního krále (pokud ignoreCheck == false).
    - Nepovolí tah na políčko, kde je figurka stejné barvy.
    - Povolené pohyby odpovídají tvaru písmene "L":
        - 2 pole jedním směrem a 1 pole kolmo (např. 2 nahoru + 1 doprava).
- Vrací true, pokud je tah legální podle pravidel jezdce, jinak false.

# Bishop.cs

Třída Bishop reprezentuje střelce a dědí z abstraktní třídy ChessPiece. Implementuje pohyb po diagonálách podle šachových pravidel.

---

## Konstruktor

*public Bishop(Color color)*

- Inicializuje střelce s barvou (White nebo Black).

---

## Vlastnosti

*public override ChessPieceName Name { get => ChessPieceName.Bishop; }*

- Vrací název figurky Bishop.

*public override Color Color => _color;*

- Vrací barvu střelce.

---

## Move(...)

*public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Provádí přesun střelce na cílové pole (bez kontroly legality).
- Nastaví board[sx, sy] na figurku a board[fx, fy] na null.
- Vždy vrací true.

---

## Clone()

*public override ChessPiece Clone()*

- Vytvoří a vrátí nového střelce se stejnou barvou.
- Vhodné pro kopírování stavu šachovnice.

---

## IsMoveLegal(...)

*public override bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board, Color ToMove, bool ignoreCheck)*

- Ověří, zda hráč táhne svojí figurkou.
- Volitelně kontroluje, zda tah nezpůsobí šach:
    - Pomocí Board.IsWhiteKingInCheck nebo IsBlackKingInCheck.
- Ověřuje diagonální tah:
    - Platný tah musí splňovat:
      |fx - sx| == |fy - sy|
    - Nesmí být žádná jiná figurka mezi startem a cílem.
- Nesmí být na cílovém poli figurka stejné barvy.

# Rook.cs

Třída Rook dědí z ChessPiece a reprezentuje věž. Implementuje logiku pro legální tahy podle šachových pravidel – tj. pohyb vodorovně nebo svisle – a sleduje, zda se věž už pohnula (např. kvůli rošádě).

---

## Konstruktor

*public Rook(Color color)*

- Vytváří novou instanci věže dané barvy (White nebo Black).

---

## Vlastnosti

*public override ChessPieceName Name => ChessPieceName.Rook;*

- Vrací jméno figurky Rook.

*public override Color Color => _color;*

- Vrací barvu věže.

*public new bool WasAlreadyMoved { get; set; } = false;*

- Označuje, zda se věž už pohnula. Důležité pro **rošádu**.

---

## Move(...)

*public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Přesune věž z počáteční pozice (fx, fy) na cílovou (sx, sy).
- Nastaví board[sx, sy] na věž, board[fx, fy] na null.
- Vždy vrací true.

---

## Clone()

*public override ChessPiece Clone()*

- Vytváří novou věž se stejnou barvou.

---

## IsMoveLegal(...)

*public override bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board, Color ToMove, bool ignoreCheck)*

Provádí kontrolu legality tahu:

1. **Hráč může táhnout pouze vlastní figurkou:**

*if ((ToMove == Color.White && this.Color == Color.Black) || ...)*

2. **Nepřípustný tah, pokud ohrožuje vlastního krále (pokud ignoreCheck == false)**:
    - Využívá Board.IsWhiteKingInCheck() nebo IsBlackKingInCheck().
3. **Směr tahu**:
    - Povolen je **pouze pohyb ve stejné řadě nebo sloupci**:

if (!((fx == sx && fy != sy) || (fy == sy && fx != sx))) return false;

4. **Cesta mezi startem a cílem musí být volná**:
    - Vodorovný tah: ověřuje všechna pole mezi fy a sy.
    - Svislý tah: ověřuje všechna pole mezi fx a sx.
5. **Nesmí táhnout na pole s figurkou stejné barvy**:

if (board[sx, sy] != null && board[sx, sy].Color == this.Color) return false;

6. **Po legálním tahu se nastaví WasAlreadyMoved = true**, což je důležité pro rošádu.

# Queen.cs

Třída Queen (dáma) dědí z ChessPiece. Kombinuje logiku **věže** a **střelce** – tedy umožňuje tahy vodorovné, svislé a diagonální.

---

## Konstruktor

*public Queen(Color color)*

- Vytváří instanci dámy s danou barvou (White nebo Black).

---

## Vlastnosti

*public override ChessPieceName Name => ChessPieceName.Queen;*

- Vrací název figurky Queen.

*public override Color Color => _color;*

- Vrací barvu dámy.

---

## Move(...)

*public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Jednoduše přesune dámu z (fx, fy) na (sx, sy) a na původní pozici nastaví null.

---

## Clone()

public override ChessPiece Clone()

- Vytváří kopii dámy se stejnou barvou.

---

## IsMoveLegal(...)

*public override bool IsMoveLegal(int fx, int fy, int sx, int sy, ChessPiece[,] board, Color ToMove, bool ignoreCheck)*

Metoda ověřuje, zda je tah platný:

1. **Zabraňuje hraní s cizí figurkou:**

*if ((ToMove == Color.White && this.Color == Color.Black) || ...)*

*return false;*

2. **Zamezuje ohrožení vlastního krále (pokud není ignoreCheck)**:
    - Pomocí Board.IsWhiteKingInCheck(...) a IsBlackKingInCheck(...).
3. **Zamezuje braní vlastní figurky:**

*if (destinationPiece != null && destinationPiece.Color == this.Color)*

*return false;*

4. **Rovné tahy jako věž:**
    - Vodorovné: fx == sx && fy != sy
    - Svislé: fy == sy && fx != sx
    - Ověřuje volnost cesty mezi počáteční a cílovou pozicí.
5. **Diagonální tahy jako střelec:**
    - Ověřuje volnost cesty diagonálně.
6. Pokud žádný z výše uvedených tahů není splněn, metoda vrací false.

# King.cs

Třída reprezentuje chování krále na šachovnici.

## Konstruktor

*public King(Color color)*

- Inicializuje krále s danou barvou.

## Vlastnosti

*public override ChessPieceName Name => ChessPieceName.King;*

*public override Color Color => _color;*

*public new bool WasAlreadyMoved = false;*

## Move(...)

*public override bool Move(int fx, int fy, int sx, int sy, ChessPiece[,] board)*

- Prostý přesun figurky.
- Nastavuje WasAlreadyMoved = true.

## Clone()

*public override ChessPiece Clone()*

# IsMoveLegal(...)

Ověřuje platnost tahu:

1. **Tah s cizí figurkou**:

```
if ((ToMove == Color.White && this.Color == Color.Black) || ...)
   return false;
```

2. **Kontrola, zda by král nebyl v šachu po tahu:**

```
if (!ignoreCheck)
{
   Board b = new Board();
   if (...) b.IsWhiteKingInCheck(...) / b.IsBlackKingInCheck(...)
      return false;
}
```

3. **Zákaz braní vlastní figurky:**

```
if (board[sx, sy] != null && board[sx, sy].Color == this.Color)
   return false;
```

4. **Legální pohyb o jedno pole všemi směry:**

```
if (Math.Abs(fx - sx) <= 1 && Math.Abs(fy - sy) <= 1)
   return true;
```