Šachová výuková aplikace

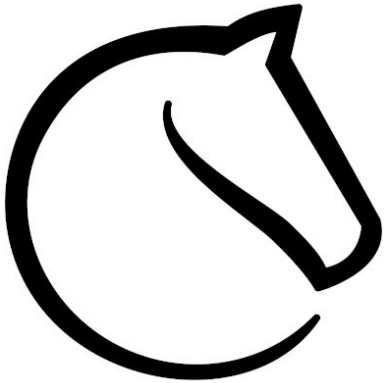
Michal Špriňar, XB

Cíle projektu

- vytvořit základ pro šachovou aplikaci
- možnost volně tvořit šachové lekce

Příklady jiných aplikací

- chess.com
- Chessable
- Lichess
- a jiné...

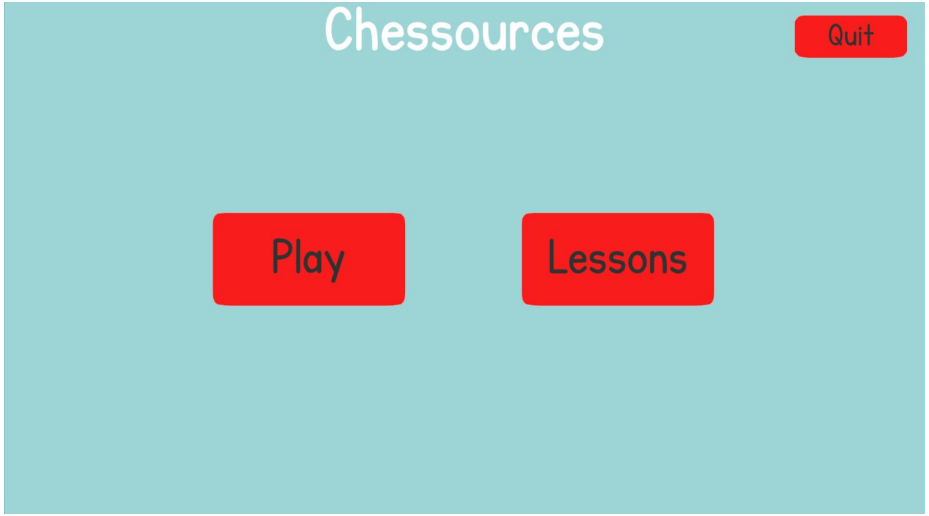


Proč tedy další šachová aplikace?

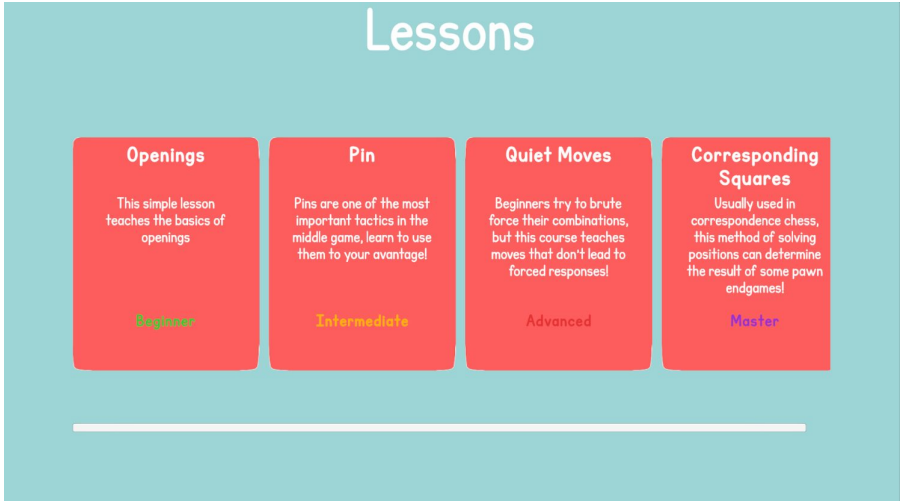
- obsah buď není plně přístupný zadarmo
- lekce jsou neúplné
- nebo jsou appky nepřehledné

Jak zlepšit nedostatky?

- přehlednost - vše potřebné je vidět hned v menu



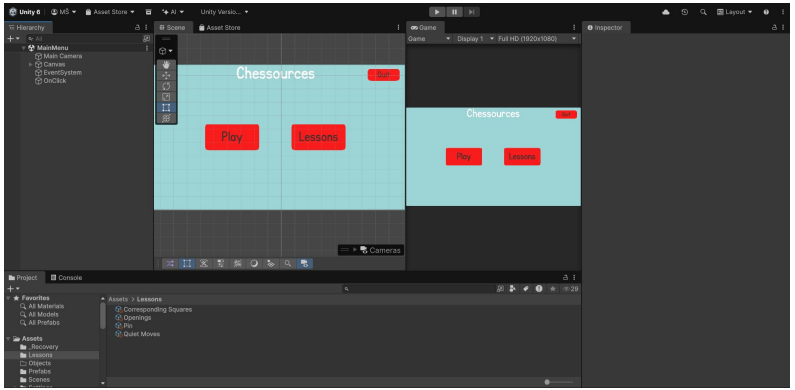
- paywall - lekce jsou plně přístupné bez jakékoliv platby



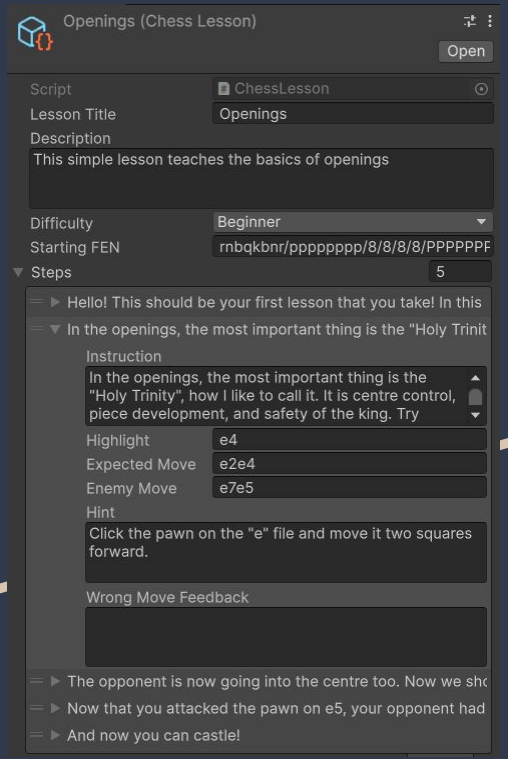
- množství - lekce budou přidávat hlavně uživatelé (jediné, co je nutné a chybí)

Implementace

- Unity
- C#



Struktura lekce



- Starting FEN - FEN string ukládá pozici, hra string přečte a postaví figurky
- je možné dělat “prázdné kroky”, tedy jen text bez tahu
- Highlight - zvýraznění políčka, lze i více najednou
- Expected Move a Enemy Move potřebuje jen políčka odkud táhnout kam

Jak funguje pohyb?

```
public void TryMoveTo(souřadnice targetTile)
{
    //získáme políčko, ze kterého chceme táhnout
    souřadnice fromTile = selectedPiece.currentTile;
    //a potom jak táhnout
    Vector2Int target = new Vector2Int(targetTile.x, targetTile.y);
    //kontrola, jestli je tah podle pravidel
    if (!currentLegalMoves.Contains(target)) return;
    //zaznamenání tahu; důležité pro vracení (hlavně v lekcích, musí se vrátit nesprávný tah)
    RegisterMove(new Move
    {
        piece = selectedPiece,
        from = fromTile,
        to = targetTile,
        capturedPiece = targetTile.occupiedPiece
    });
    //dále názvy metod mluví za sebe
    MovePiece(selectedPiece, targetTile);
    //special moves = proměna pěšce, rošáda a en passant
    HandleSpecialMoves(selectedPiece, fromTile, targetTile);
    FinalizeMove(selectedPiece, fromTile, targetTile);
    SwitchTurn();
    CheckGameState();
}
```

Co pravidla?

```
static List<Vector2Int> RookMoves(Piece piece, souřadnice[,] board)
    List<Vector2Int> moves = new();

    int x = piece.currentTile.x;
    int y = piece.currentTile.y;

    Vector2Int[] directions = {
        new Vector2Int(0, 1),
        new Vector2Int(0, -1),
        new Vector2Int(1, 0),
        new Vector2Int(-1, 0)
    };

    foreach (Vector2Int dir in directions)
    {
        int checkX = x + dir.x;
        int checkY = y + dir.y;

        while (IsInsideBoard(checkX, checkY))
        {
            Piece target = board[checkX, checkY].occupiedPiece;

            if (target != null)
            {
                if (target.color != piece.color)
                    moves.Add(new Vector2Int(checkX, checkY));
                break;
            }

            moves.Add(new Vector2Int(checkX, checkY));
            checkX += dir.x;
            checkY += dir.y;
        }
    }

    return moves;
```

- jednoduše řečeno se používá vektorů
- dokud jsou tahy na šachovnici, použijí se vektory ve všech směrech a do seznamu tahů se přidávají tahy, dokud “nenarazí” na nějakou figurku
- příklad - tahy věže

```
static bool IsInsideBoard(int x, int y)
{
    return x >= 0 && x < 8 && y >=0 && y < 8;
}
```

```
public static bool IsInCheck(string color, souřadnice[,] board)
{
    Vector2Int kingPos = Vector2Int.zero;
    for (int x = 0; x < 8; x++)
        for (int y = 0; y < 8; y++)
            if (board[x, y].occupiedPiece != null &&
            board[x, y].occupiedPiece.type == "King" &&
            board[x, y].occupiedPiece.color == color)
            kingPos = new Vector2Int(x, y);

    string enemyColor = color == "White" ? "Black" : "White";
    for (int x = 0; x < 8; x++)
        for (int y = 0; y < 8; y++)
        {
            Piece piece = board[x, y].occupiedPiece;
            if (piece != null && piece.color == enemyColor)
            {
                List<Vector2Int> moves = GetLegalMovesRaw(piece, board);
                if (moves.Contains(kingPos))
                    return true;
            }
        }
    return false;
}
```

- u všech figur se ještě musí kontrolovat, jestli náhodou nejsou v šachu nebo dané tahy nejsou v šachu

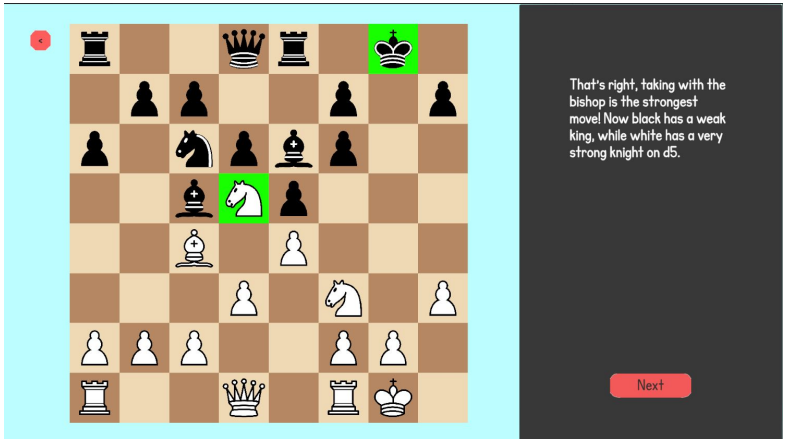
```
public static List<Vector2Int> GetLegalMoves(Piece piece, souřadnice[,] board)
{
    List<Vector2Int> raw = GetLegalMovesRaw(piece, board);
    List<Vector2Int> legal = new();

    foreach (Vector2Int move in raw)
    {
        Piece originalOccupant = board[move.x, move.y].occupiedPiece;
        souřadnice fromTile = board[piece.currentTile.x, piece.currentTile.y];

        board[move.x, move.y].occupiedPiece = piece;
        fromTile.occupiedPiece = null;
        Vector2Int originalTile = new Vector2Int(piece.currentTile.x, piece.currentTile.y);
        piece.currentTile = board[move.x, move.y];

        if (!IsInCheck(piece.color, board))
            legal.Add(move);

        piece.currentTile = board[originalTile.x, originalTile.y];
        fromTile.occupiedPiece = piece;
        board[move.x, move.y].occupiedPiece = originalOccupant;
    }
    if (piece.type == "King")
        legal.AddRange(GetCastlingMoves(piece,board));
    return legal;
}
```

Příklad jedné lekce


Now that it's white's turn, find a move to create more pressure on the pin!
Hint


Great! Now the knight is defended only by the pawn on g7 and queen on d8. Let's see what happens after a common move from black.
Next


The bishop on e6 does absolutely nothing against us taking the knight. What piece would you take with to create a huge advantage?
Hint


That's right, taking with the bishop is the strongest move! Now black has a weak king, while white has a very strong knight on d5.
Next


Lesson complete!
Proceed

Problémy na vyřešení

- pokrytí celého políčka figurkou a zároveň stejné velikosti (povedlo se vždy jen jedno…)
- custom highlight - důležité zejména pro lekci korespondující pole

Co do budoucna?

- lepší grafika
- editor lekcí
- nahrávání a stahování lekcí
- hraní s počítačem nebo s reálným soupeřem
  - ideálně na čas
- a další...