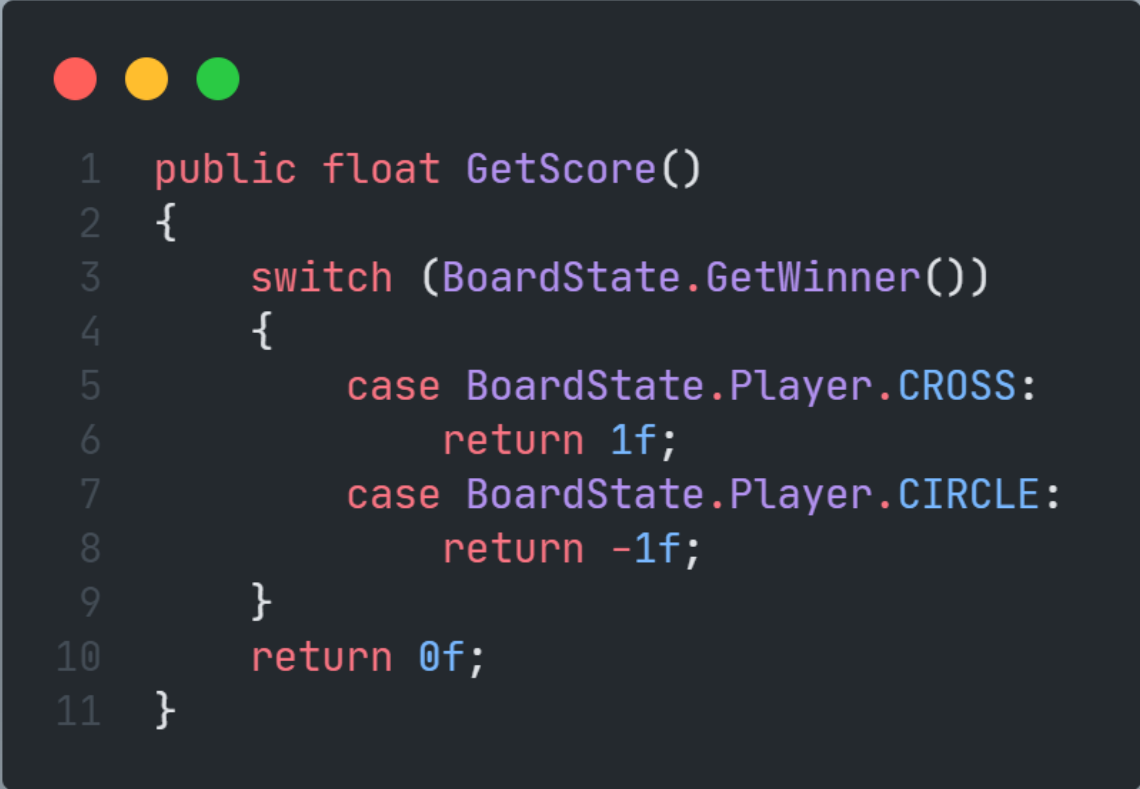


MVP do algoritmo de minimax

Para esse MVP, foi implementado o algoritmo de minimax de uma maneira simples em um jogo da velha.

Pontuação

O algoritmo nesse caso constrói a árvore de possibilidades inteira do jogo e atribui pontuação 1 para caso o "X" tenha vencido, e -1 caso o "O" tenha vencido, qualquer outro caso a pontuação é 0.

A screenshot of a code editor with a dark background and light-colored text. At the top left of the editor window, there are three colored circles: red, yellow, and green. The code is written in a C#-like syntax. It shows a method named GetScore() that returns a float. Inside the method, there is a switch statement that checks the result of BoardState.GetWinner(). If the winner is BoardState.Player.CROSS, it returns 1f. If the winner is BoardState.Player.CIRCLE, it returns -1f. If there is no winner, it returns 0f. The lines of code are numbered from 1 to 11 on the left side of the editor.

```
1 public float GetScore()
2 {
3     switch (BoardState.GetWinner())
4     {
5         case BoardState.Player.CROSS:
6             return 1f;
7         case BoardState.Player.CIRCLE:
8             return -1f;
9     }
10    return 0f;
11 }
```

Árvore

A árvore é construída a partir do trecho de código a seguir, a função é recursiva e chama a si mesma para cada um dos galhos da árvore, ela verifica se é possível fazer uma jogada em determinada casa e cria um objeto para o estado do tabuleiro se a jogada for feita

```
1 private BoardNode GetBoardNode(BoardState boardState, int move = -1, int depth = 0)
2 {
3     List<BoardNode> children = new();
4
5     if (depth < MaxDepth && !boardState.GameEnded())
6     {
7         for (int i = 0; i < 9; i++)
8         {
9             if (boardState[i] != CellState.FREE) continue;
10
11             var newBoardState = boardState.Clone();
12             newBoardState.Play(i);
13
14             children.Add(GetBoardNode(newBoardState, i, depth + 1));
15         }
16     }
17
18     return new BoardNode
19     {
20         BoardState = boardState,
21         Children = children,
22         move = move,
23         depth = depth
24     };
25 }
```

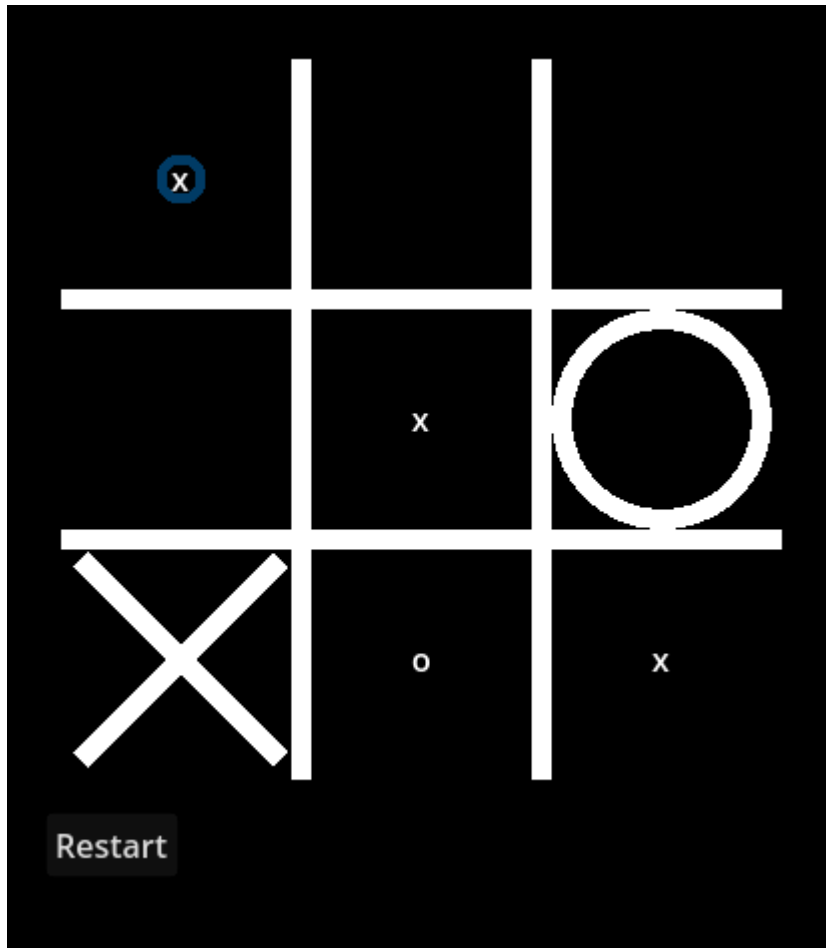
Melhor jogada

No final, a função GetScore percorre a árvore e procura a pontuação das folhas, depois, caso seja vez do jogador "X", ele escolhe o galho com maior pontuação, caso seja "O", o galho com menor pontuação

```
1 private int GetBestMove(BoardState boardState)
2 {
3     BoardNode rootNode = GetBoardNode(boardState);
4     if (rootNode.Children.Count == 0) return -1;
5
6
7     int scoreMultiplier = rootNode.BoardState.CurrentPlayer == BoardState.Player.CROSS ? 1 : -1;
8     var orderedNodes = rootNode.Children.OrderByDescending(child =>
9     {
10         float score = GetScore(child);
11         EmitSignal(SignalName.ScoreCalculated, child.move, score);
12         return score * scoreMultiplier;
13     });
14
15     return orderedNodes.First().move;
16 }
17
18 private float GetScore(BoardNode boardNode)
19 {
20     if (boardNode.Children.Count == 0) return boardNode.GetScore() / (float)boardNode.depth;
21
22     if (boardNode.BoardState.CurrentPlayer == BoardState.Player.CROSS)
23         return boardNode.Children.Max(node => GetScore(node));
24     else
25         return boardNode.Children.Min(node => GetScore(node));
26 }
```

Funcionamento

A partir disso, após definidas as regras do jogo, é possível ter uma previsão precisa de qual jogador está em vantagem em qualquer momento da partida



Nessa imagem, é a vez do jogador "X", as casas marcadas com "X" garantem a vitória, as casas marcadas com "O" garantem a derrota, e as casas sem marcação levam ao empate, considerando que cada jogador jogue perfeitamente