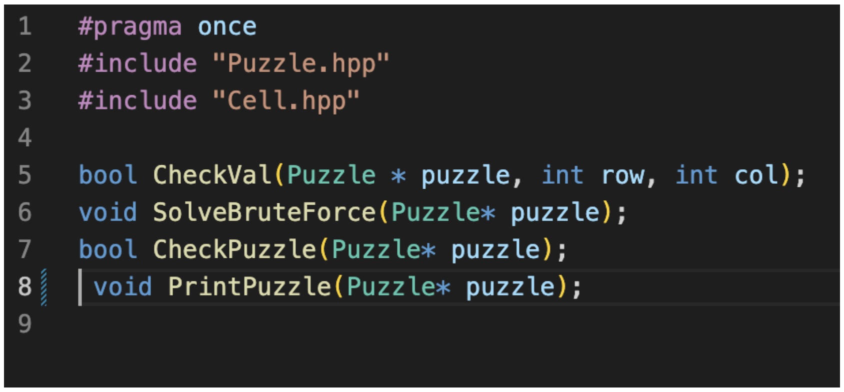
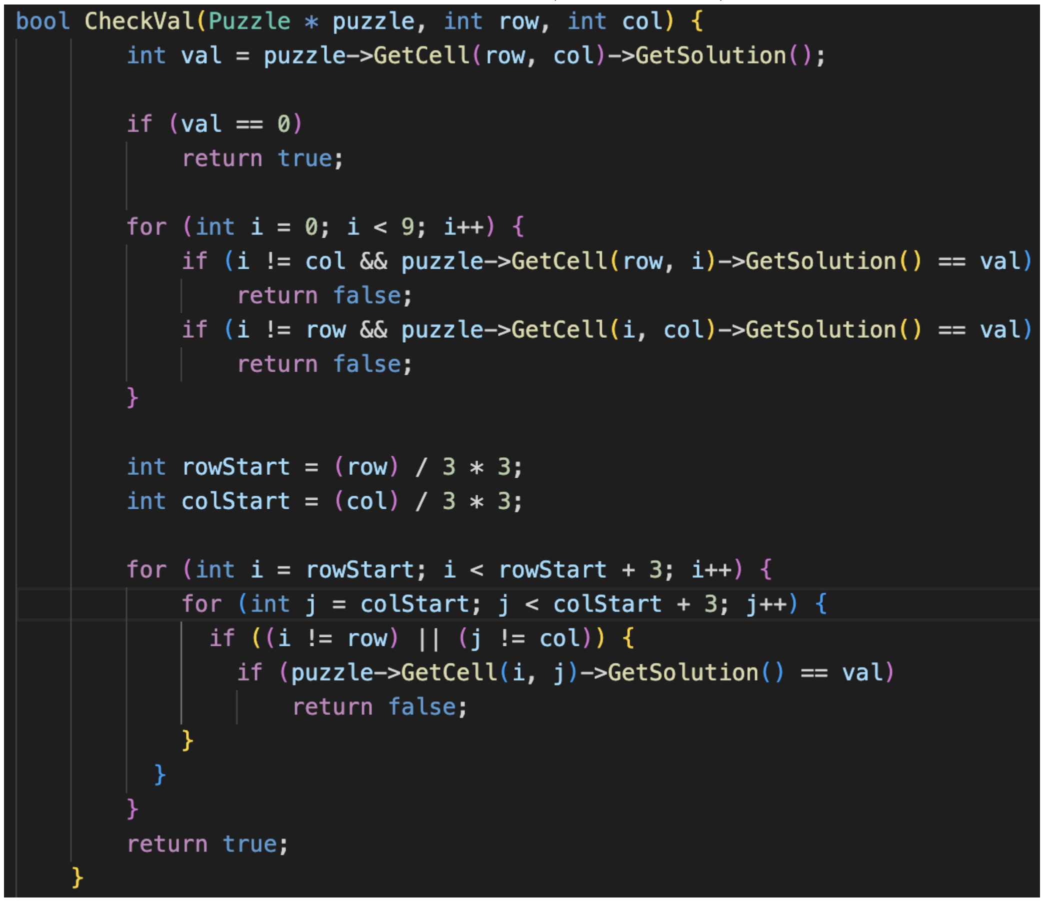
API Documentation: Class Algorithms



```cpp
#pragma once
#include "Puzzle.hpp"
#include "Cell.hpp"

bool CheckVal(Puzzle * puzzle, int row, int col);
void SolveBruteForce(Puzzle* puzzle);
bool CheckPuzzle(Puzzle* puzzle);
void PrintPuzzle(Puzzle* puzzle);

```

CheckVal(Puzzle * puzzle , int row, int col):

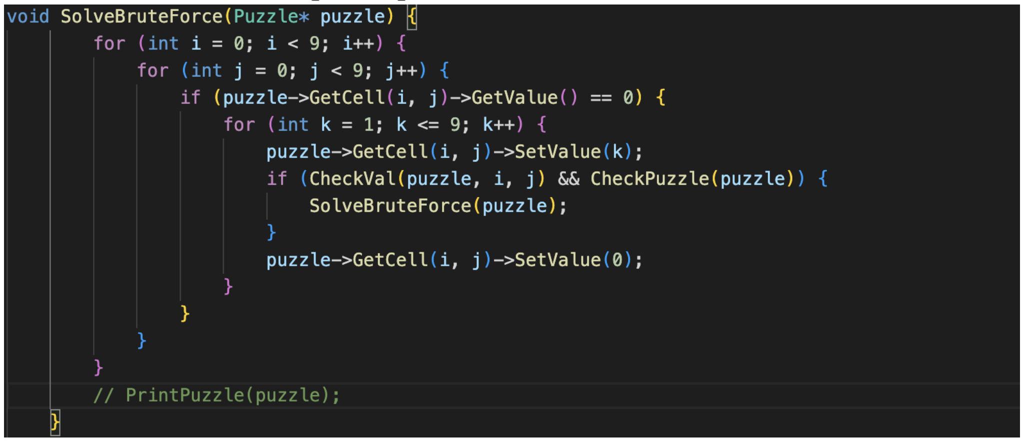
Description: Gets the current puzzle and checks if every new cell update is a valid entry. Helper function of SolveBruteForce
Parameters: The current puzzle object, row, and column that the SolveBruteForce method is solving
Exceptions: The method is unable to get the current puzzle
Returns: Boolean if current value is valid (True or false)



```cpp
bool CheckVal(Puzzle * puzzle, int row, int col) {
        int val = puzzle->GetCell(row, col)->GetSolution();

        if (val == 0)
            return true;

        for (int i = 0; i < 9; i++) {
            if (i != col && puzzle->GetCell(row, i)->GetSolution() == val)
                return false;
            if (i != row && puzzle->GetCell(i, col)->GetSolution() == val)
                return false;
        }

        int rowStart = (row) / 3 * 3;
        int colStart = (col) / 3 * 3;

        for (int i = rowStart; i < rowStart + 3; i++) {
            for (int j = colStart; j < colStart + 3; j++) {
              if ((i != row) || (j != col)) {
                if (puzzle->GetCell(i, j)->GetSolution() == val)
                    return false;
            }
          }
        }
        return true;
    }
```

SolveBruteForce(Puzzle* puzzle):

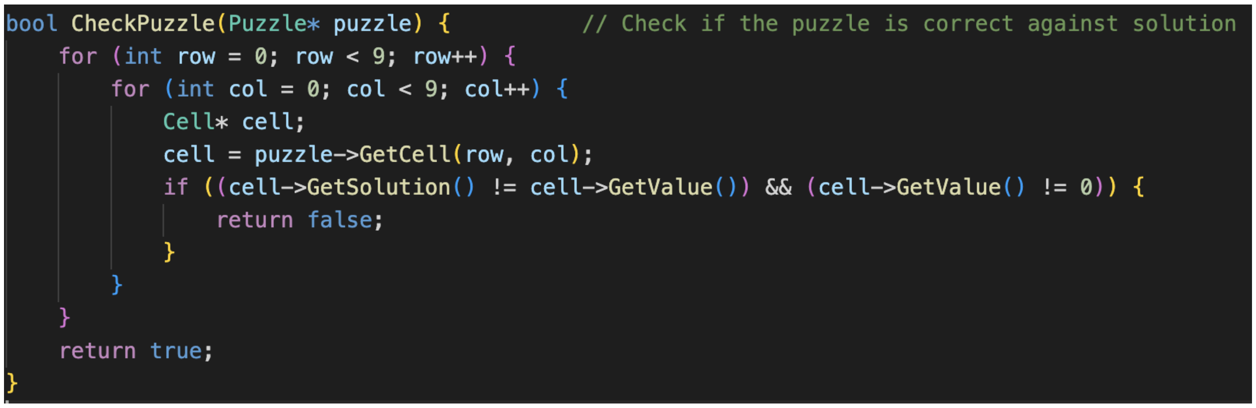
Description: Solves the inputted puzzle by backtracking.
Parameters: The current puzzle object
Exceptions: The method is unable to get the current puzzle
Returns: Prints the completed puzzle and returns true if the Puzzle was



```
void SolveBruteForce(Puzzle* puzzle) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (puzzle->GetCell(i, j)->GetValue() == 0) {
                for (int k = 1; k <= 9; k++) {
                    puzzle->GetCell(i, j)->SetValue(k);
                    if (CheckVal(puzzle, i, j) && CheckPuzzle(puzzle)) {
                        SolveBruteForce(puzzle);
                    }
                    puzzle->GetCell(i, j)->SetValue(0);
                }
            }
        }
    }
    // PrintPuzzle(puzzle);
}
```

CheckPuzzle (Puzzle * puzzle):

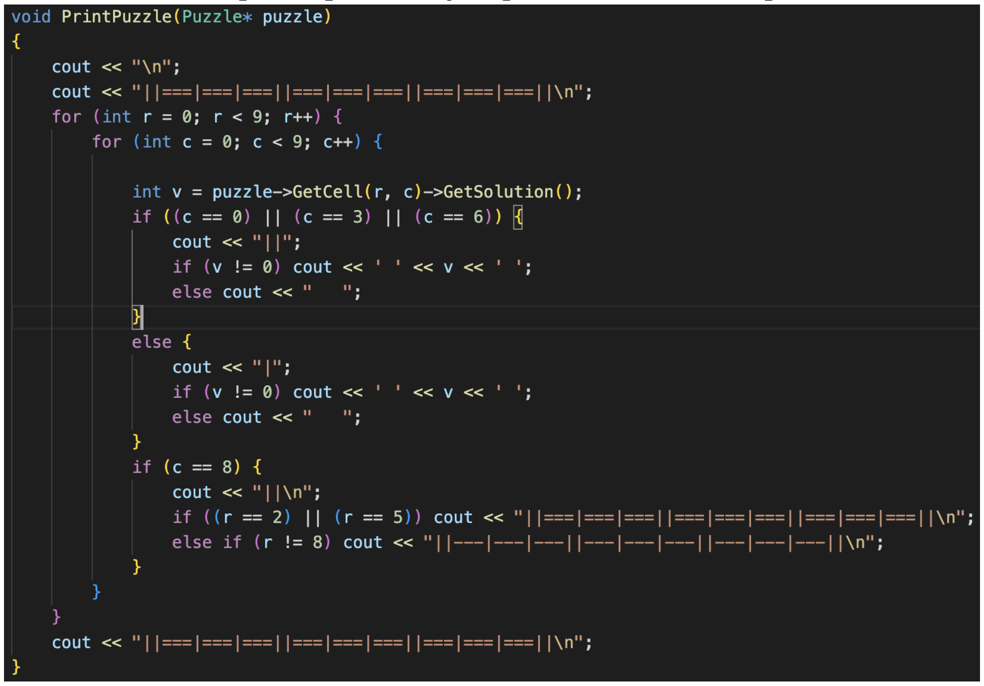
Description: Intermittently checks if the current board state is a solved puzzle
Parameters: The current puzzle object and the user input
Exceptions: The method is unable to get the current puzzle
Returns: The completed puzzle object



```
bool CheckPuzzle(Puzzle* puzzle) {          // Check if the puzzle is correct against solution
    for (int row = 0; row < 9; row++) {
        for (int col = 0; col < 9; col++) {
            Cell* cell;
            cell = puzzle->GetCell(row, col);
            if ((cell->GetSolution() != cell->GetValue()) && (cell->GetValue() != 0)) {
                return false;
            }
        }
    }
    return true;
}
```

PrintPuzzle(Puzzle * puzzle)

Description: Print the puzzle
Parameters: The current puzzle object
Exceptions: The method is unable to get the current puzzle
Returns: The completed puzzle object printed as a board representation



```cpp
void PrintPuzzle(Puzzle* puzzle)
{
    cout << "\n";
    cout << "||===|===|===||===|===|===||===|===|===||\n";
    for (int r = 0; r < 9; r++) {
        for (int c = 0; c < 9; c++) {

            int v = puzzle->GetCell(r, c)->GetSolution();
            if ((c == 0) || (c == 3) || (c == 6)) {
                cout << "||";
                if (v != 0) cout << ' ' << v << ' ';
                else cout << "   ";
            }
            else {
                cout << "|";
                if (v != 0) cout << ' ' << v << ' ';
                else cout << "   ";
            }
            if (c == 8) {
                cout << "||\n";
                if ((r == 2) || (r == 5)) cout << "||===|===|===||===|===|===||===|===|===||\n";
                else if (r != 8) cout << "||---|---|---||---|---|---||---|---|---||\n";
            }
        }
    }
    cout << "||===|===|===||===|===|===||===|===|===||\n";
}
```