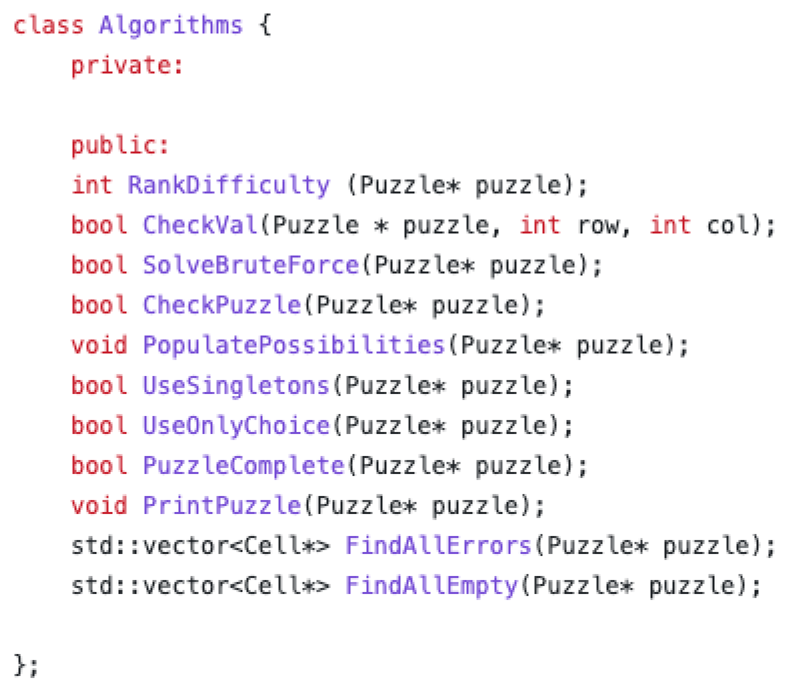
RankDifficulty(puzzle)

This call is using an algorithm that reads a puzzle board and determines the difficulty of the board. This call is not something the user or GameEngine really relies on.



```
class Algorithms {
    private:

    public:
    int RankDifficulty (Puzzle* puzzle);
    bool CheckVal(Puzzle * puzzle, int row, int col);
    bool SolveBruteForce(Puzzle* puzzle);
    bool CheckPuzzle(Puzzle* puzzle);
    void PopulatePossibilities(Puzzle* puzzle);
    bool UseSingletons(Puzzle* puzzle);
    bool UseOnlyChoice(Puzzle* puzzle);
    bool PuzzleComplete(Puzzle* puzzle);
    void PrintPuzzle(Puzzle* puzzle);
    std::vector<Cell*> FindAllErrors(Puzzle* puzzle);
    std::vector<Cell*> FindAllEmpty(Puzzle* puzzle);

};
```

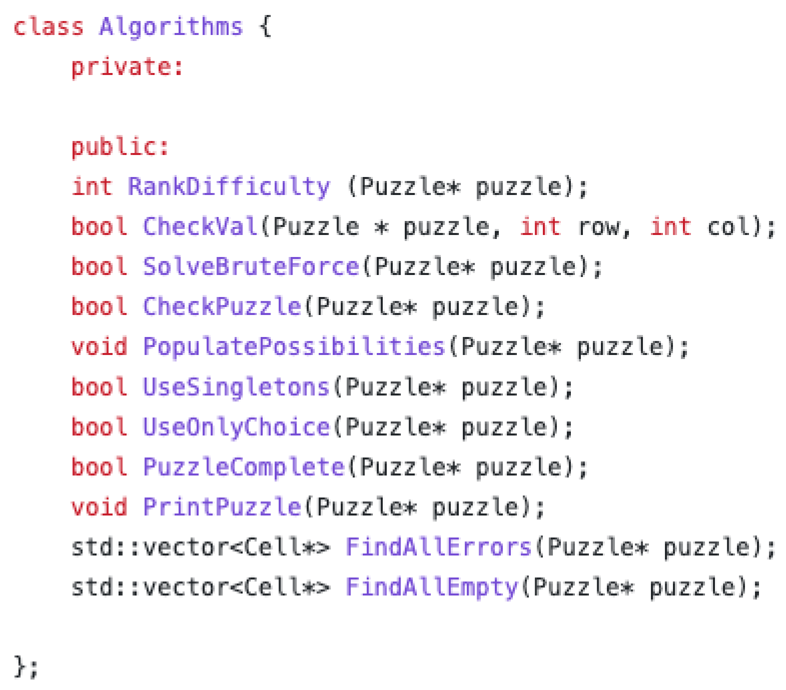
Parameters:
A puzzle
    The function will read in a puzzle as the parameter.

Returns:
An int
    The return value of the function will be an integer, that is the difficult level.

Exceptions:
If it is an invalid sudoku board, then the function will not be able to determine the difficulty level. It will halt the function call.

PuzzleComplete(puzzle)

This function reads in a puzzle board and checks to see if it is completed.



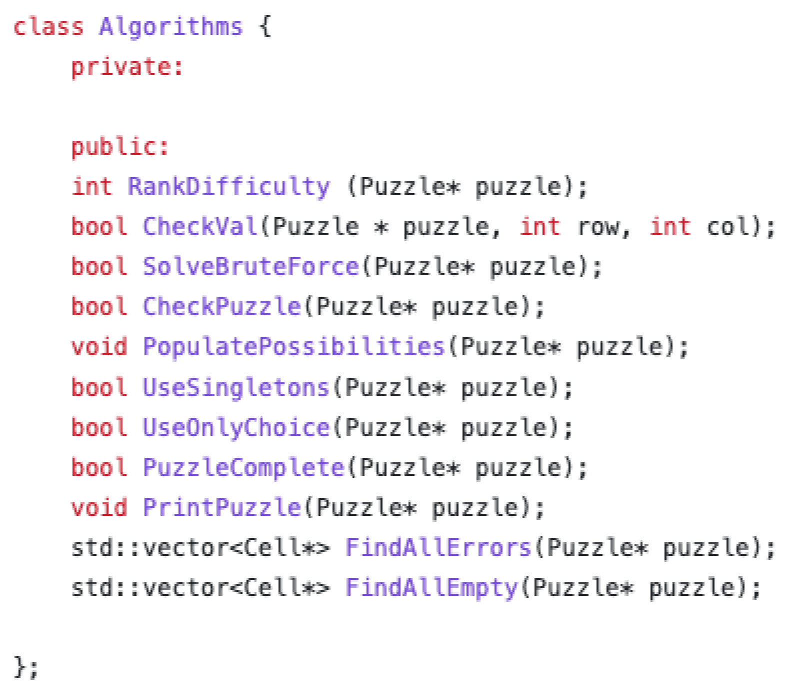
```
class Algorithms {
    private:

    public:
    int RankDifficulty (Puzzle* puzzle);
    bool CheckVal(Puzzle * puzzle, int row, int col);
    bool SolveBruteForce(Puzzle* puzzle);
    bool CheckPuzzle(Puzzle* puzzle);
    void PopulatePossibilities(Puzzle* puzzle);
    bool UseSingletons(Puzzle* puzzle);
    bool UseOnlyChoice(Puzzle* puzzle);
    bool PuzzleComplete(Puzzle* puzzle);
    void PrintPuzzle(Puzzle* puzzle);
    std::vector<Cell*> FindAllErrors(Puzzle* puzzle);
    std::vector<Cell*> FindAllEmpty(Puzzle* puzzle);

};
```

Parameters:
A puzzle
    The function will read in a puzzle as the parameter.

Returns:
A bool
    The return true if the puzzle is complete, and false if the puzzle is not complete

PopulatePuzzle(puzzle)

Populates values into notes array of each cell in the puzzle, all possible values that don't conflict along/in its row, column, and group



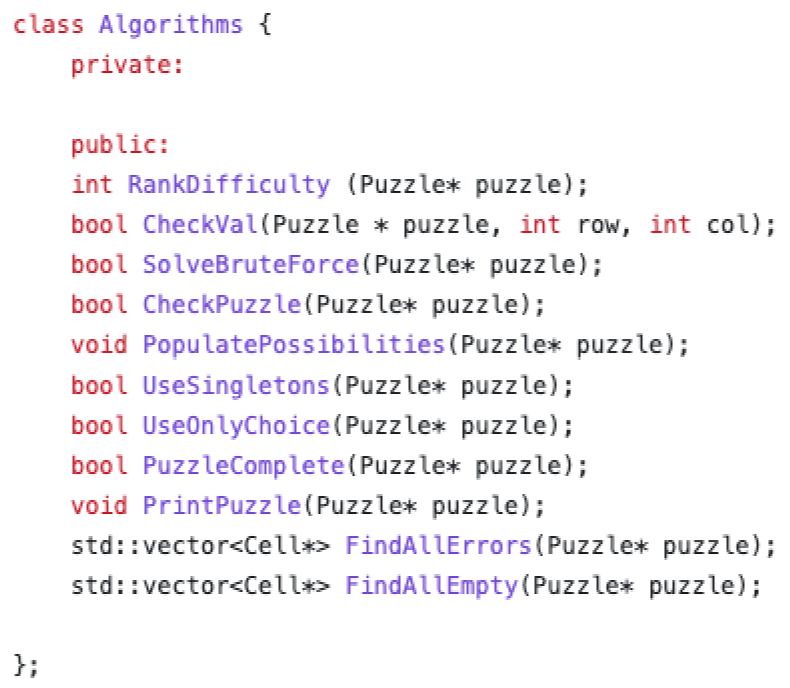
```
class Algorithms {
    private:

    public:
    int RankDifficulty (Puzzle* puzzle);
    bool CheckVal(Puzzle * puzzle, int row, int col);
    bool SolveBruteForce(Puzzle* puzzle);
    bool CheckPuzzle(Puzzle* puzzle);
    void PopulatePossibilities(Puzzle* puzzle);
    bool UseSingletons(Puzzle* puzzle);
    bool UseOnlyChoice(Puzzle* puzzle);
    bool PuzzleComplete(Puzzle* puzzle);
    void PrintPuzzle(Puzzle* puzzle);
    std::vector<Cell*> FindAllErrors(Puzzle* puzzle);
    std::vector<Cell*> FindAllEmpty(Puzzle* puzzle);

};
```

Parameters:
A puzzle
    The function will read in a puzzle as the parameter.

Returns:
Void
    This puzzle does not return a value.

UseSingletons(puzzle)

Traverses over the whole puzzle and looks in each cell's notes for any case where there is only one choice. If yes, it sets the solution with that value



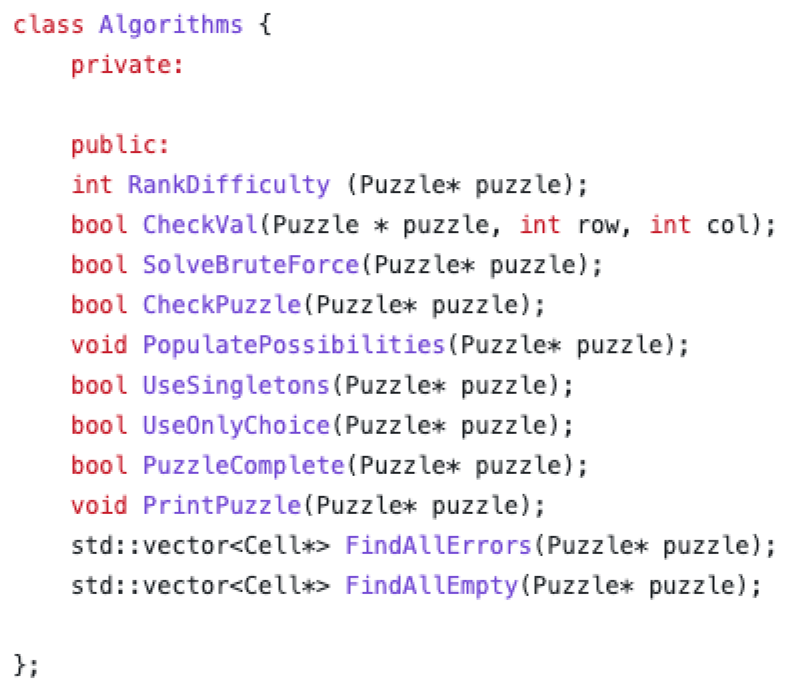
```
class Algorithms {
    private:

    public:
    int RankDifficulty (Puzzle* puzzle);
    bool CheckVal(Puzzle * puzzle, int row, int col);
    bool SolveBruteForce(Puzzle* puzzle);
    bool CheckPuzzle(Puzzle* puzzle);
    void PopulatePossibilities(Puzzle* puzzle);
    bool UseSingletons(Puzzle* puzzle);
    bool UseOnlyChoice(Puzzle* puzzle);
    bool PuzzleComplete(Puzzle* puzzle);
    void PrintPuzzle(Puzzle* puzzle);
    std::vector<Cell*> FindAllErrors(Puzzle* puzzle);
    std::vector<Cell*> FindAllEmpty(Puzzle* puzzle);

};
```

Parameters:
A puzzle
    The function will read in a puzzle as the parameter.

Returns:
Bool
    Returns a true value once the puzzle has been modified

UseOnlyChoice(puzzle)

Traverses over the puzzle three different ways: by row, by column, and by group looking for where a value occurs only once. When it does, it gets set in the puzzle's solution.



```
class Algorithms {
    private:

    public:
    int RankDifficulty (Puzzle* puzzle);
    bool CheckVal(Puzzle * puzzle, int row, int col);
    bool SolveBruteForce(Puzzle* puzzle);
    bool CheckPuzzle(Puzzle* puzzle);
    void PopulatePossibilities(Puzzle* puzzle);
    bool UseSingletons(Puzzle* puzzle);
    bool UseOnlyChoice(Puzzle* puzzle);
    bool PuzzleComplete(Puzzle* puzzle);
    void PrintPuzzle(Puzzle* puzzle);
    std::vector<Cell*> FindAllErrors(Puzzle* puzzle);
    std::vector<Cell*> FindAllEmpty(Puzzle* puzzle);

};
```

Parameters:
A puzzle
The function will read in a puzzle as the parameter.

Returns:
Bool
Returns a true value once the puzzle has been modified