Usage of GitHub Co-pilot in our AI course project

Ahmed Wael   Adham Hazem   Omar Brikaa
Mootaz Medhat   Ali Esmat

May 20, 2023

1 Introduction

Using Prolog as our programming language of choice for the AI course project, we have found some advantages and disadvantages of using GitHub Co-pilot to auto-generate code.

2 Advantages

We have mostly found GitHub Co-pilot[1] to suggest correct code completion in two cases: avoiding manually typing code that does not require cognitive effort (figures 1, 2), and helping in the completion of commonly typed utility code (figures 3, 4).

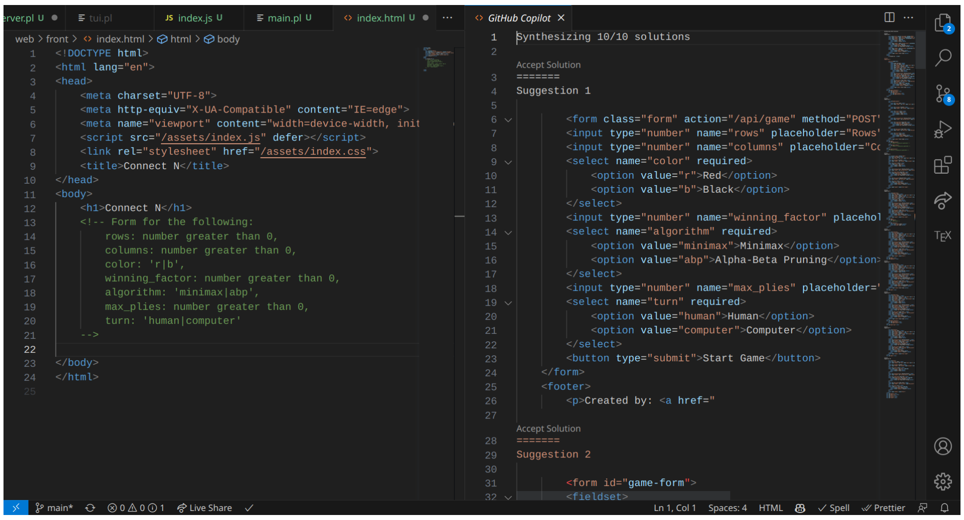
Even if the code that Co-pilot generated in these cases was not used as is, it was a good starting point.

3 Disadvantages

We have mostly found GitHub Co-pilot to generate inaccurate code in codebase-specific algorithms that aren't commonly used in other projects (figure 5).

Moreover, Co-pilot does not suggest code refactoring and would suggest duplicate code if the programmer does not exert effort to maintain the quality of their codebase.

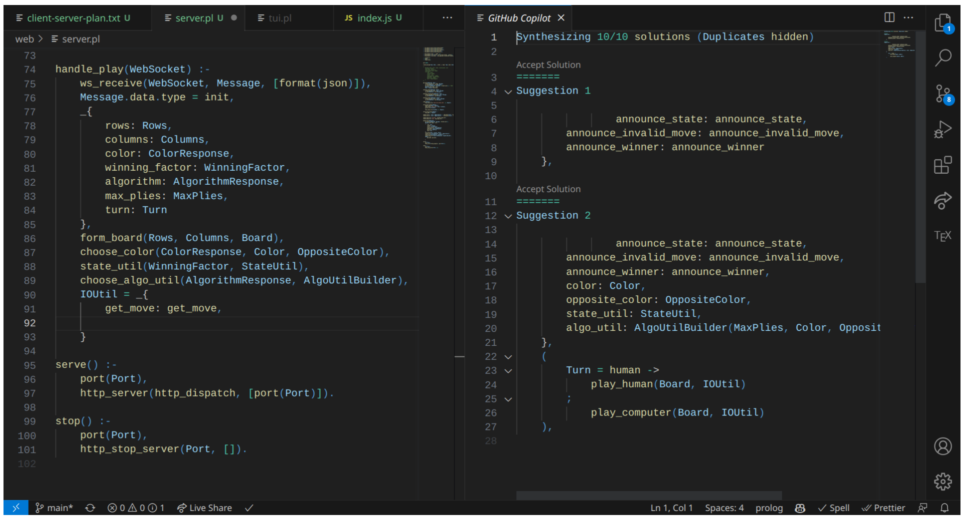
[1]https://github.com/features/copilot



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, init
    <script src="/assets/index.js" defer></script>
    <link rel="stylesheet" href="/assets/index.css">
    <title>Connect N</title>
</head>
<body>
    <h1>Connect N</h1>
    <!-- Form for the following:
        rows: number greater than 0,
        columns: number greater than 0,
        color: 'r|b',
        winning_factor: number greater than 0,
        algorithm: 'minimax|abp',
        max_plies: number greater than 0,
        turn: 'human|computer'
    -->

</body>
</html>
```

```
Synthesizing 10/10 solutions

Accept Solution
=======
Suggestion 1

        <form class="form" action="/api/game" method="POST'
        <input type="number" name="rows" placeholder="Rows'
        <input type="number" name="columns" placeholder="Cc
        <select name="color" required>
            <option value="r">Red</option>
            <option value="b">Black</option>
        </select>
        <input type="number" name="winning_factor" placehol
        <select name="algorithm" required>
            <option value="minimax">Minimax</option>
            <option value="abp">Alpha-Beta Pruning</option>
        </select>
        <input type="number" name="max_plies" placeholder='
        <select name="turn" required>
            <option value="human">Human</option>
            <option value="computer">Computer</option>
        </select>
        <button type="submit">Start Game</button>
    </form>
    <footer>
        <p>Created by: <a href="

Accept Solution
=======
Suggestion 2

        <form id="game-form">
        <fieldset>
```

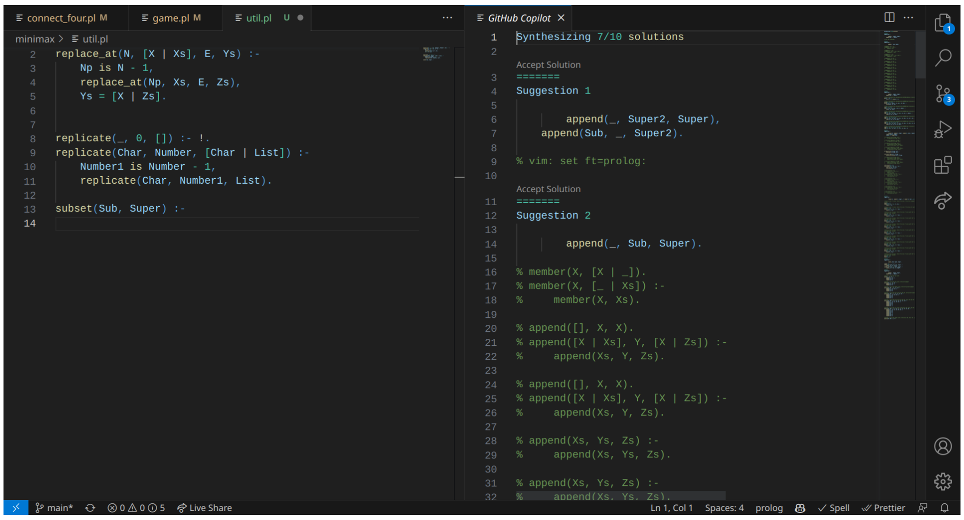
Figure 1: GitHub co-pilot auto-completing an HTML form based on a requirements comment (helpful)



```prolog

handle_play(WebSocket) :-
    ws_receive(WebSocket, Message, [format(json)]),
    Message.data.type = init,
    _{
        rows: Rows,
        columns: Columns,
        color: ColorResponse,
        winning_factor: WinningFactor,
        algorithm: AlgorithmResponse,
        max_plies: MaxPlies,
        turn: Turn
    },
    form_board(Rows, Columns, Board),
    choose_color(ColorResponse, Color, OppositeColor),
    state_util(WinningFactor, StateUtil),
    choose_algo_util(AlgorithmResponse, AlgoUtilBuilder),
    IOUtil = _{
        get_move: get_move,

    }

serve() :-
    port(Port),
    http_server(http_dispatch, [port(Port)]).

stop() :-
    port(Port),
    http_stop_server(Port, []).
```

```
Synthesizing 10/10 solutions (Duplicates hidden)

Accept Solution
=======
Suggestion 1

                announce_state: announce_state,
        announce_invalid_move: announce_invalid_move,
        announce_winner: announce_winner
    },

Accept Solution
=======
Suggestion 2

                announce_state: announce_state,
        announce_invalid_move: announce_invalid_move,
        announce_winner: announce_winner,
        color: Color,
        opposite_color: OppositeColor,
        state_util: StateUtil,
        algo_util: AlgoUtilBuilder(MaxPlies, Color, Opposit
    },
    (
        Turn = human ->
            play_human(Board, IOUtil)
        ;
            play_computer(Board, IOUtil)
    ),
```

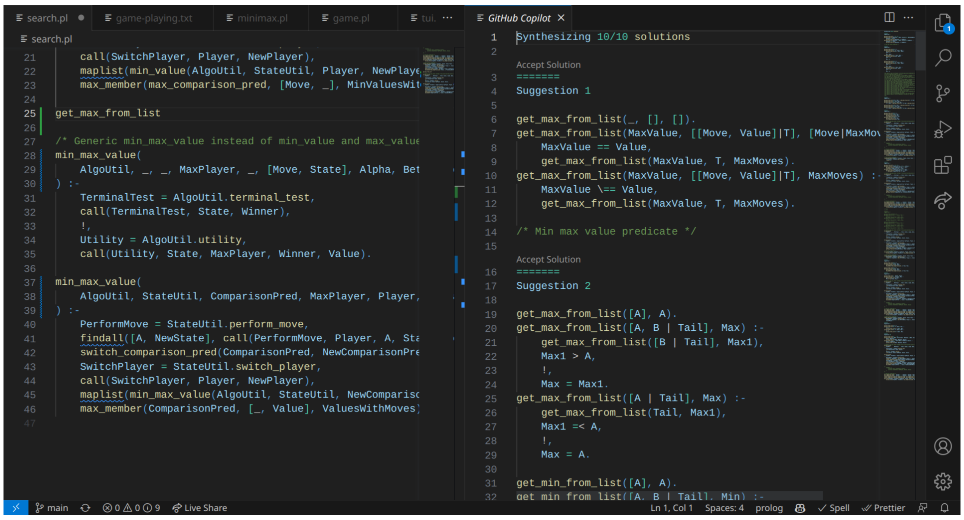
Figure 2: GitHub co-pilot auto-completing a configuration dictionary (helpful)



```
replace_at(N, [X | Xs], E, Ys) :-
    Np is N - 1,
    replace_at(Np, Xs, E, Zs),
    Ys = [X | Zs].


replicate(_, 0, []) :- !.
replicate(Char, Number, [Char | List]) :-
    Number1 is Number - 1,
    replicate(Char, Number1, List).

subset(Sub, Super) :-

```

```
Synthesizing 7/10 solutions

Accept Solution
=======
Suggestion 1

        append(_, Super2, Super),
    append(Sub, _, Super2).

% vim: set ft=prolog:

Accept Solution
=======
Suggestion 2

        append(_, Sub, Super).

% member(X, [X | _]).
% member(X, [_ | Xs]) :-
%     member(X, Xs).

% append([], X, X).
% append([X | Xs], Y, [X | Zs]) :-
%     append(Xs, Y, Zs).

% append([], X, X).
% append([X | Xs], Y, [X | Zs]) :-
%     append(Xs, Y, Zs).

% append(Xs, Ys, Zs) :-
%     append(Xs, Ys, Zs).

% append(Xs, Ys, Zs) :-
%     append(Xs, Ys, Zs).
```

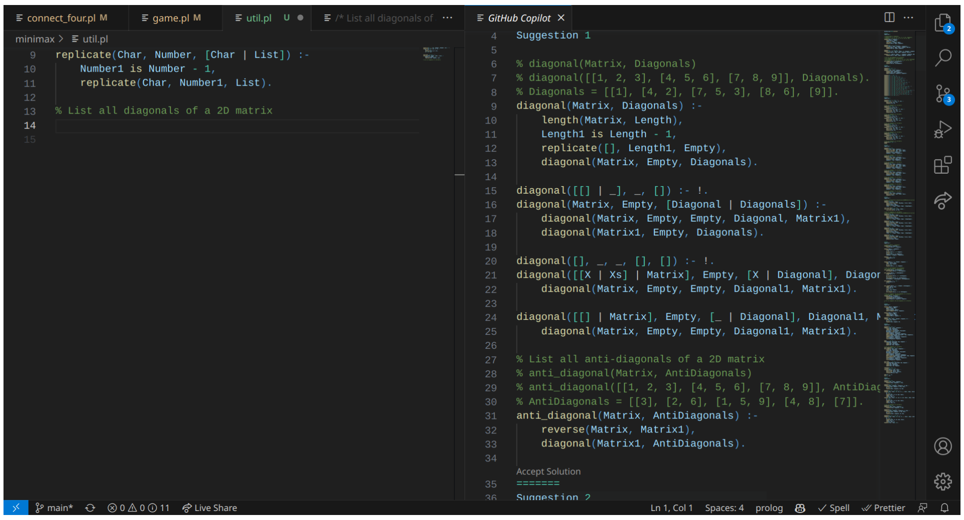
Figure 3: GitHub co-pilot auto-completing the commonly used subset utility predicate (helpful)



```
    call(SwitchPlayer, Player, NewPlayer),
    maplist(min_value(AlgoUtil, StateUtil, Player, NewPlaye
    max_member(max_comparison_pred, [Move, _], MinValuesWit

get_max_from_list

/* Generic min_max_value instead of min_value and max_value
min_max_value(
    AlgoUtil, _, _, MaxPlayer, _, [Move, State], Alpha, Bet
) :-
    TerminalTest = AlgoUtil.terminal_test,
    call(TerminalTest, State, Winner),
    !,
    Utility = AlgoUtil.utility,
    call(Utility, State, MaxPlayer, Winner, Value).

min_max_value(
    AlgoUtil, StateUtil, ComparisonPred, MaxPlayer, Player,
) :-
    PerformMove = StateUtil.perform_move,
    findall([A, NewState], call(PerformMove, Player, A, Sta
    switch_comparison_pred(ComparisonPred, NewComparisonPre
    SwitchPlayer = StateUtil.switch_player,
    call(SwitchPlayer, Player, NewPlayer),
    maplist(min_max_value(AlgoUtil, StateUtil, NewCompariso
    max_member(ComparisonPred, [_, Value], ValuesWithMoves)
```

```
Synthesizing 10/10 solutions

Accept Solution
=======
Suggestion 1

get_max_from_list(_, [], []).
get_max_from_list(MaxValue, [[Move, Value]|T], [Move|MaxMov
    MaxValue == Value,
    get_max_from_list(MaxValue, T, MaxMoves).
get_max_from_list(MaxValue, [[Move, Value]|T], MaxMoves) :-
    MaxValue \== Value,
    get_max_from_list(MaxValue, T, MaxMoves).

/* Min max value predicate */

Accept Solution
=======
Suggestion 2

get_max_from_list([A], A).
get_max_from_list([A, B | Tail], Max) :-
    get_max_from_list([B | Tail], Max1),
    Max1 > A,
    !,
    Max = Max1.
get_max_from_list([A | Tail], Max) :-
    get_max_from_list(Tail, Max1),
    Max1 =< A,
    !,
    Max = A.

get_min_from_list([A], A).
get_min_from_list([A, B | Tail], Min) :-
```

Figure 4: GitHub co-pilot auto-completing the commonly used maximum from list predicate (helpful)



```
replicate(Char, Number, [Char | List]) :-
    Number1 is Number - 1,
    replicate(Char, Number1, List).

% List all diagonals of a 2D matrix

```

```
Suggestion 1

% diagonal(Matrix, Diagonals)
% diagonal([[1, 2, 3], [4, 5, 6], [7, 8, 9]], Diagonals).
% Diagonals = [[1], [4, 2], [7, 5, 3], [8, 6], [9]].
diagonal(Matrix, Diagonals) :-
    length(Matrix, Length),
    Length1 is Length - 1,
    replicate([], Length1, Empty),
    diagonal(Matrix, Empty, Diagonals).

diagonal([[] | _], _, []) :- !.
diagonal(Matrix, Empty, [Diagonal | Diagonals]) :-
    diagonal(Matrix, Empty, Empty, Diagonal, Matrix1),
    diagonal(Matrix1, Empty, Diagonals).

diagonal([], _, _, [], []) :- !.
diagonal([[X | Xs] | Matrix], Empty, [X | Diagonal], Diagor
    diagonal(Matrix, Empty, Empty, Diagonal1, Matrix1).

diagonal([[] | Matrix], Empty, [_ | Diagonal], Diagonal1, M
    diagonal(Matrix, Empty, Empty, Diagonal1, Matrix1).

% List all anti-diagonals of a 2D matrix
% anti_diagonal(Matrix, AntiDiagonals)
% anti_diagonal([[1, 2, 3], [4, 5, 6], [7, 8, 9]], AntiDiag
% AntiDiagonals = [[3], [2, 6], [1, 5, 9], [4, 8], [7]].
anti_diagonal(Matrix, AntiDiagonals) :-
    reverse(Matrix, Matrix1),
    diagonal(Matrix1, AntiDiagonals).

Accept Solution
=======
Suggestion 2
```

Figure 5: GitHub co-pilot suggesting incorrect code to get the diagonals of a 2D matrix