

q01

me:

```
void turn(char grid[GRID_SIZE][GRID_SIZE], int* playerrow, int* playercol, int* moves, int*
treasuresFound){ char move; scanf("%c", &move); // check for invalid input switch (move){ case 'D': if
(*playerrow < GRID_SIZE - 1){ // cant move down if at bottom of grid *playerrow++; *moves++; }
break; case 'U': if (*playerrow > 0){ // cant move up if at top of grid *playerrow--; *moves++; } break;
case 'L': if (*playercol > 0){ // cant move left if at left of grid *playercol--; *moves++; } break; case 'R':
if (*playercol < GRID_SIZE - 1){ // cant move right if at right of grid *playercol++; *moves++; } break;
default: printf("Invalid move\n"); break; } if (grid[*playerrow][*playercol] == 'T'){ *treasuresFound++;
grid[*playerrow][*playercol] = '_'; } } *treasureFound++not increasing
```

github\_copilot:

The issue might be due to the fact that the `*treasuresFound++` line is only executed when the current grid cell contains a 'T'. If the player never moves to a cell containing a 'T', `*treasuresFound` will never increase.

If `*treasuresFound` is not increasing even when the player moves to a cell containing a 'T', then there might be an issue with how the grid is initialized or how the player's moves are being processed.

Here are a few things you can check:

Check the grid initialization: Make sure that there are cells in the grid that are initialized with 'T'. You can do this by printing the grid before the game starts.

Check the player's moves: Make sure that the player's row and column are being updated correctly when they make a move. You can do this by printing the player's row and column after each move.

Check the condition for increasing `*treasuresFound`: Make sure that the condition `grid[*playerrow][*playercol] == 'T'` is true when it should be. You can do this by adding a print statement inside the if statement, like this:

```
if (grid[*playerrow][*playercol] == 'T'){
    printf("Treasure found!\n");
    * treasuresFound++;
    grid[*playerrow][*playercol]='_'; }
```

```
{
```

q02

me:

or :scanf("%d", &nextFloor); what is the value of nextFloor if the input is A

github\_copilot:

The scanf function in C attempts to read and convert input according to the format specifier provided. In this case, the %d format specifier is used, which expects an integer.

If you input a non-integer value like 'A' when scanf("%d", &nextFloor); is expecting an integer, the function will fail to convert the input and nextFloor will not be updated. This means nextFloor will retain its previous value, or be uninitialized if it hasn't been assigned a value before.

Additionally, the non-integer input will remain in the input buffer, which could cause issues with subsequent input operations. It's generally a good idea to validate and clear the input buffer after a scanf operation to prevent such issues.

me:

How can I clear the input buffer after a failed scanf operation in C?

github\_copilot:

You can clear the input buffer in C using a simple loop that reads and discards characters until a newline character or EOF (End of File) is encountered. Here's how you can do it:

```
int c;
```

```
while ((c = getchar()) != '\n' && c != EOF){ }
```

This loop uses getchar() to read characters one by one from the input buffer. If scanf fails to read an integer, the loop will remove the invalid input from the buffer.