Parsa Aghaali 400521072

Hw5 – Compiler

The goal was to enhance the Minesweeper DSL compiler to handle a **hint** option that determines whether the number of bombs adjacent to a cell should be displayed. Specifically:

- If **hint** is **True**, the output should display the number of adjacent bombs for each cell that is not a bomb.
- If **hint** is **False**, the output should use a placeholder character (e.g., **#**) for cells that are not bombs.

The work I did was to update the **CustomExampleDSLCodeGenerator** class to correctly handle the **hint** option for the Minesweeper DSL. This involves:

1. Displaying the number of bombs adjacent to each cell if **hint** is **True**.
2. Using a placeholder character (**#**) for cells without bombs if **hint** is **False**.

Class Overview

The **CustomExampleDSLCodeGenerator** class is responsible for generating the code based on the parsed abstract syntax tree (AST) of the Minesweeper DSL. It maintains stacks for operands and generated code, and processes nodes based on their type.

Key Changes Made

1. **Added Hint Option Handling**
    - Introduced a class attribute **hint_option** to store the value of the **hint** option.
    - Implemented a method **set_hint_option** to capture and set the hint option value.
2. **Modified Code Generation Logic**
    - Updated the **generate_program** method to generate different output logic based on the **hint** option value.
    - Included two versions of the board display logic:
        - One for **hint: True** that displays the number of adjacent bombs.
        - One for **hint: False** that uses a placeholder character (**#**).

3. **Count Adjacent Bombs Function**
    - Added a helper function **count_adjacent_bombs** within the generated code to count the number of bombs surrounding a given cell.

Detailed Code Changes

Initialization:

- The **hint_option** attribute is initialized with the default value **"False"**.

Method is_operand:

- Checks if an item is an operand by ensuring it is not in the list of non-operands.

Method generate_code:

- Iterates through the post-order array, generating code based on whether items are operands or non-operands.

Method generate_code_based_on_non_operand:

- Calls the appropriate code generation method based on the type of non-operand item.

Method generate_program:

- Constructs the final program code based on the **hint** option and output type.
- Includes logic for counting adjacent bombs and displaying the board appropriately.

Method generate_initiate_game:

- Initializes the game board with the given dimensions.

Method generate_bomb:

- Places bombs at specified locations on the board.

Method set_output_type:

- Sets the output type (e.g., **console**).

Method set_hint_option:

- Sets the value of the **hint** option based on the parsed input.

Method generate_bomb_placements:

- Handles bomb placements within a scope.

Methods generate_begin_scope_operator and generate_end_scope_operator:

- Manage scope boundaries within the generated code.

And for the **CustomExampleDSLListener :**

The **CustomExampleDSLListener** class is responsible for listening to events triggered by the parsing process of ExampleDSL code. It captures the parsed information and constructs an abstract syntax tree (AST) using an **AST** object provided by the **default_codes.ast** module.

Key Components

1. **Constructor (__init__):**
    - Initializes the listener with a set of overridden rules and a default hint option value.
    - Creates an instance of the **AST** class to build the abstract syntax tree.
2. **Method exitEveryRule:**
    - Invoked after parsing each rule in the DSL grammar.
    - Checks if the current rule is not overridden and constructs the AST node accordingly.
3. **Method exitProgram, exitInitiate_game, exitBomb_placements, exitOutput, exitShowhint:**
    - Invoked when exiting specific rules defined in the DSL grammar.
    - Constructs AST nodes for the corresponding rules and adds them to the AST.
    - In the case of **exitShowhint**, captures the value of the **hint** option.

Detailed Analysis

- **Rule Overriding:**
    - The class overrides specific rules (**program**, **initiate_game**, **output**, **showhint**) while allowing other rules to be handled by the superclass.
- **AST Construction:**

    - The listener constructs AST nodes for each relevant rule encountered during parsing.
    - AST nodes are created using the **make_ast_subtree** function provided by the **default_codes.make_ast_subtree** module.
- **Hint Option Handling:**
    - The **hint** option value is captured and stored in the **hint_option** attribute.
    - The value is extracted from the parsed context using the **getText()** method.

Conclusion

The **CustomExampleDSLListener** class effectively captures and processes parsed information from ExampleDSL code, constructing an abstract syntax tree that reflects the structure of the code. By overriding specific rules and handling the **hint** option, it enables customization and flexibility in handling different aspects of ExampleDSL code. This implementation serves as a crucial component in the parsing and analysis of ExampleDSL programs.