به نام خدا

علی شیخ عطار ۹۹۵۴۲۲۲۲

```
program: output? hint? initiate_game bomb_placements;

output:'output:' output_types ;

output_types: 'html' | 'console';

hint: 'hint:' bool;

bool : 'True' | 'False';
```

در قدم اول hint را به گرامر اضافه میکنیم تا parser آن را بشناسد.
بعد از output باید hint استفاده شود که با توجه به علامت سوال یعنی الزامی به وجودش نیست. و به صورت عبارت hint: Bool می آید که خود bool میتواند true یا false باشد.
در گام بعدی به rul های listener آن را اضافه میکنیم

```
2 usages
class CustomExampleDSLListener(ExampleDSLListener):
    def __init__(self, rule_names):
        self.overridden_rules = ['program', 'initiate_game', 'output', 'hint']
```

و همچنین تابع exithint را override میکنیم واسم نود را hint قرار میدهیم و keep_node را مانند نود های دیگر true قرار میدهیم.

```
def exitHint(self, ctx):
    make_ast_subtree(self.ast, ctx, node_value="hint", keep_node=True)
```

در مرحله ی بعدی کد برنامه ی dsl_code_generator را تغییر میدهیم و کلاس CustomExampleDSLCodeGenerator را تغییر میدهیم و به پراپرتی no_operands استرینگ hint را اضافه میکنیم تا آن را به عنوان یک no_operand بشناسد و توابع لازم را اجرا

کنید

```
class CustomExampleDSLCodeGenerator:
    def __init__(self):
        self.non_operands = ['program', 'initiate_game',
                             'bomb_location', 'output', 'hint',
                             'bomb_placements', 'begin_scope_operator',
                             'end_scope_operator']
        self.operand_stack = []
        self.code_stack = []
        self.hint = False
        self.width = 0
        self.height = 0
```

همچنین دو متغیر width و height را اضافه میکنیم تا به صورت گلوبال بتوانیم به آن ها در مراحل مختلف دسترسی داشته باشیم

و فیلد hint را به عنوان فلگی تعریف می کنیم تا در صورت لزوم کد های مربوطه را به برنامه اضافه کنیم.

```
1 usage
def generate_code_based_on_non_operand(self, item):
    if item == "program":
        self.generate_program()

    elif item == "output":
        self.set_output_type()

    elif item == "initiate_game":
        self.generate_initiate_game()

    elif item == "bomb_location":
        self.generate_bomb()

    elif item == "bomb_placements":
        self.generate_bomb_placements()

    elif item == "begin_scope_operator":
        self.generate_begin_scope_operator()

    elif item == "end_scope_operator":
        self.generate_end_scope_operator()

    elif item == "hint":
        self.set_hint()
```

حال داخل تابع generate_code_based_on_non_operand حالتی که به hint میرسیم را اضافه میکنیم تا تابع مربوطه اجرا شود(تصویر بالا)

با اجرای set_hint فلگ hint کلاس در صورتی که مقدار hint خوانده شده از کاربر درست باشد true ست میشود و در ادامه بسته به مقدار این فلگ عملیات های اضافه میشوند.

```
1 usage
def set_hint(self):
    flag = self.operand_stack.pop()
    if flag == 'True':
        self.hint = True
```

در تابع generate_initiate_game از دو پراپرتی width و height برای تعریف آرایه ی bombs استفاده میشود و در ادامه اگر فلگ hint درست بود آنگاه متغیری با نام hint و مقدار true در برنامه نوشته میشود و آرایه ای با سایز bombs ایجاد میشود

```
def generate_initiate_game(self):
    self.height = int(self.operand_stack.pop())
    self.width = int(self.operand_stack.pop())
    code_string = f"bombs = [[False for y in range({self.height})] for x in range({self.width})]\n"
    self.code_stack.append(code_string)
    if self.hint:
        hint_string = "hint = True\n"
        self.code_stack.append(hint_string)
        hints = f"hints = [[0 for _ in range({self.height})] for __ in range({self.width})]\n"
        self.code_stack.append(hints)
```

```
self.code_stack.append(hints)
hint_function = "\ndef hint_func(x, y):\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x - 1 >= {0} and {self.height} > y - 1 >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x - 1][y - 1] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x - 1 >= {0} and {self.height} > y >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x - 1][y] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x - 1 >= {0} and {self.height} > y + 1>= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x - 1][y + 1] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x >= {0} and {self.height} > y + 1 >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x][y + 1] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x + 1 >= {0} and {self.height} > y + 1 >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x + 1][y + 1] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x + 1 >= {0} and {self.height} > y >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x + 1][y] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x + 1 >= {0} and {self.height} > y - 1 >= {0}:\n"
self.code_stack.append(hint_function)
hint_function = "\t\thints[x + 1][y - 1] += 1\n"
self.code_stack.append(hint_function)
hint_function = f"\tif {self.width} > x >= {0} and {self.height} > y - 1 >= {0}:\n"
```

در ادامه تابعی در کد برنامه ی خروجی تعریف میشود که با دریافت مختصات بمب هشت همسایه ی آن را در صورتی که بمب نباشند در ارایه ی متناظر در hints اپدیت میکند (یکی اضافه میکند)(تصویر بالا)

در صورتی که hint false باشد تنها متغیر hint با مقدار false و آرایه ی خالی تعریف می شود که از خطاهای runtime جلوگیری کند

```
    else:
        hint_string = "hint = False\n"
        self.code_stack.append(hint_string)
        hints = "hints = []\n"
        self.code_stack.append(hints)
```

در تابع generate_bomb به ازای هر بمبی که تعریف می شود تابع hint_func متناظر آن (در صورت درست بودن hint) صدا زده میشود تا هینت های متناظر آن اپدیت شود.

```
def generate_bomb(self):
    y = int(self.operand_stack.pop())
    x = int(self.operand_stack.pop())
    code_string = f"bombs[{x - 1}][{y - 1}] = True\n"
    self.code_stack.append(code_string)
    if self.hint:
        call_hint_function = f"hint_func({x - 1}, {y - 1})\n\n"
        self.code_stack.append(call_hint_function)
```

و در نهایت در تابع generate_program در pragram_code شرط جدیدی را اضافه میکنیم که در صورتی که خانه بمب نبود و مقدار hint ورودی true بود و مقدار خانه ی متناظر این خانه در hints مقداری بزرگ تر از صفر داشت (hint وجود داشت برای آن) مقدار hint چاپ شود و در غیر این دو حالت مقدار دیفالت '#' چاپ شود

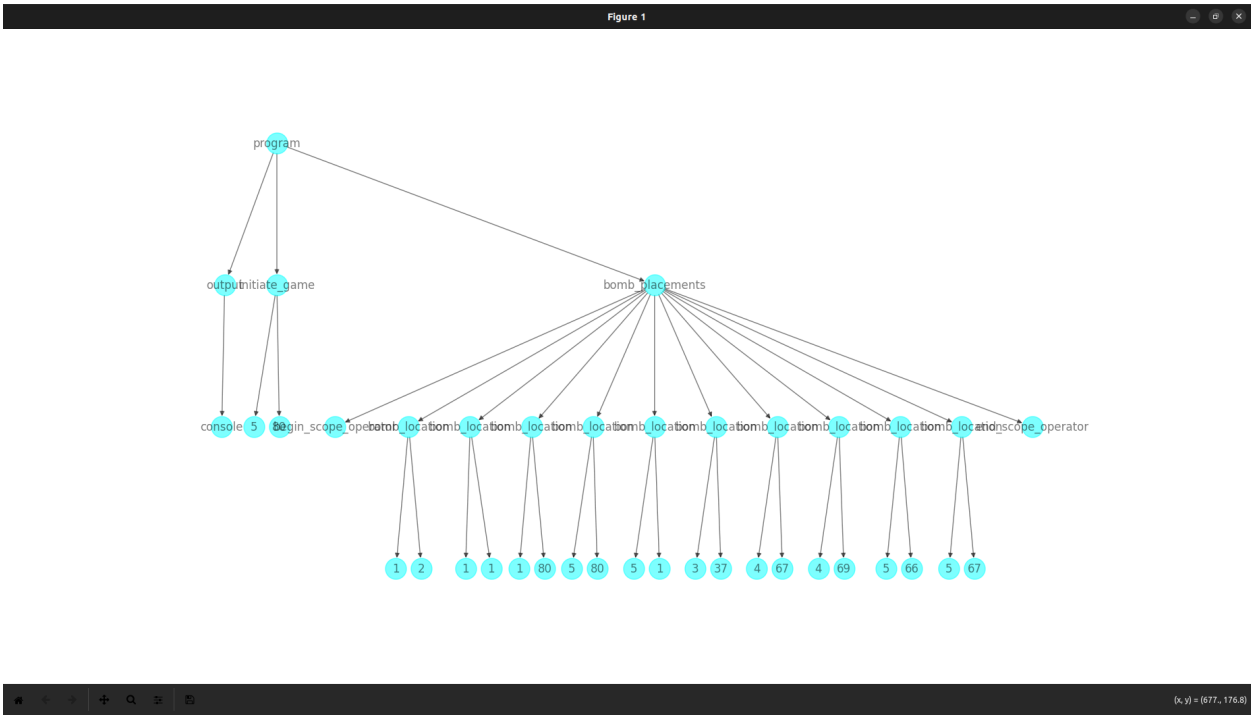
```
    if output_type == 'console':
        program_code = (initiate_code + placements_code
                        + f"\nfor i in range({self.width}):\n" +
                        f"\tfor j in range({self.height}):\n" +
                        "\t\tif bombs[i][j]:\n" +
                        "\t\t\tprint('*', end ='')\n" +
                        "\t\telif hint and hints[i][j] > 0:\n" +
                        "\t\t\tprint(hints[i][j], end ='')\n" +
                        "\t\telse:\n" +
                        "\t\t\tprint('#', end ='')\n" +
                        "\tprint()"
                        )
        self.code_stack.append(program_code)
```

در ادامه با دو نمونه برنامه را تست میکنیم.

custom_example_dsl_code_generator.py | main.py | test_output.py | test.minesweeper

```
output: console
game: 5 X 80
bomb: 1 , 2
bomb: 1, 1
bomb: 1, 80
bomb: 5, 80
bomb: 5, 1
bomb: 3, 37
bomb: 4, 67
bomb: 4, 69
bomb: 5, 66
bomb: 5, 67
```

به برنامه hint را ورودی نمیدهیم



همانطور که میبینیم در درخت آن نود hint وجود ندارد

custom_example_dsl_code_generator.py | main.py | test_output.py

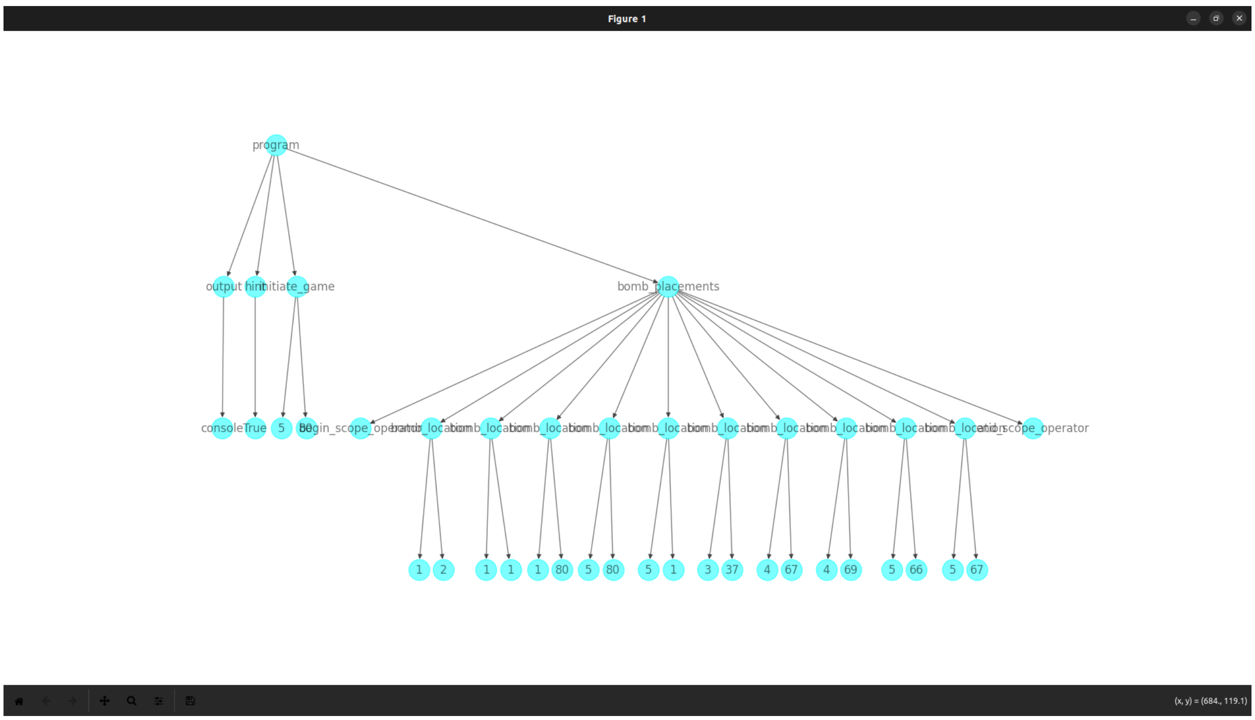
```python
##COMPILER_PARAM:::output_type:::console
bombs = [[False for y in range(80)] for x in range(5)]
hint = False
hints = []
bombs[0][1] = True
bombs[0][0] = True
bombs[0][79] = True
bombs[4][79] = True
bombs[4][0] = True
bombs[2][36] = True
bombs[3][66] = True
bombs[3][68] = True
bombs[4][65] = True
bombs[4][66] = True

for i in range(5):
    for j in range(80):
        if bombs[i][j]:
            print('*', end ='')
        elif hint and hints[i][j] > 0:
            print(hints[i][j], end ='')
        else:
            print('#', end ='')
    print()
```

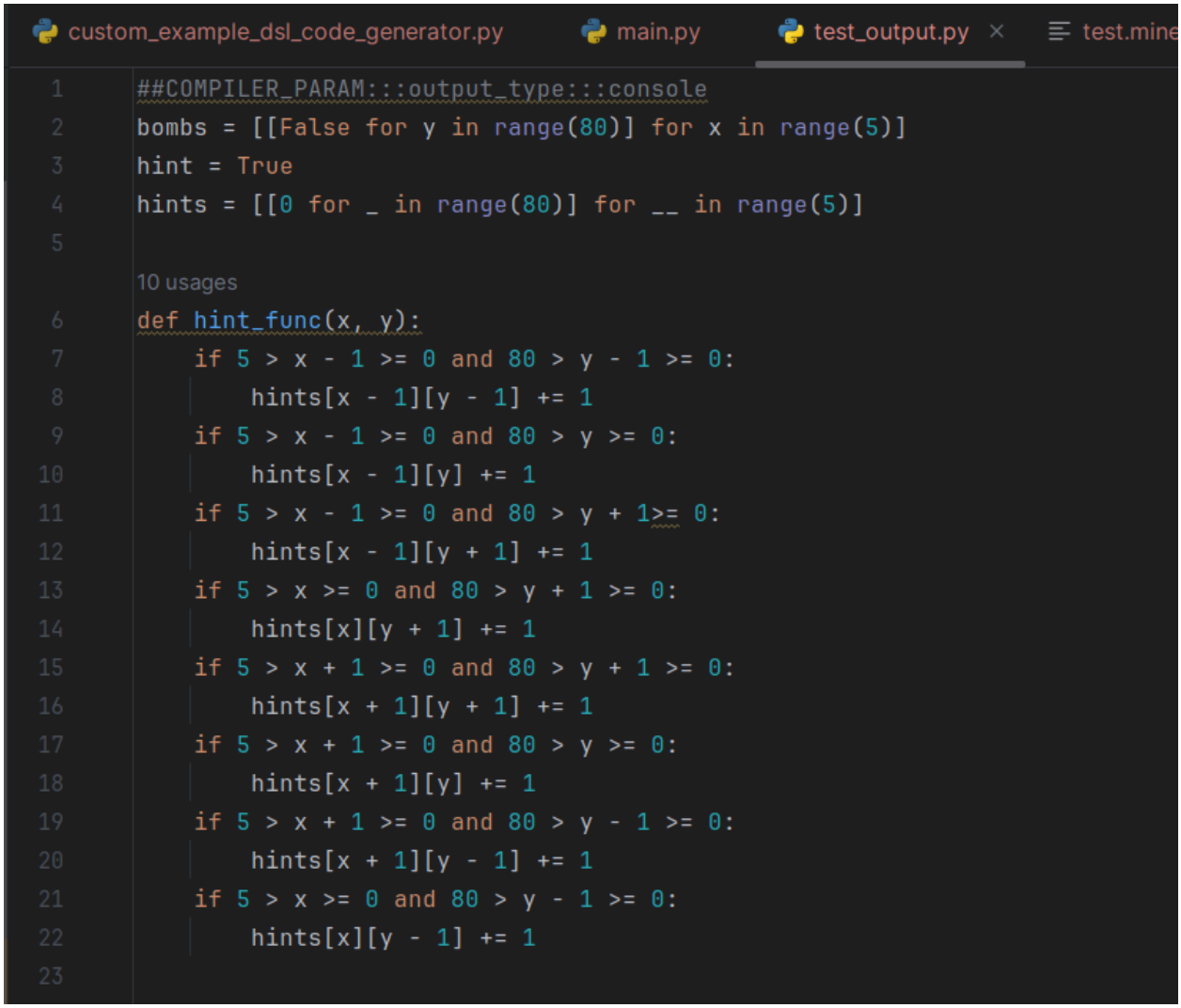
خروجی کد به صورت بالا میباشد که متغیر hint با مقدار False تعریف شده و hints به عنوان یک ارایه ی خالی تعریف میشود و خروجی به صورت زیر میشود.

```
/usr/bin/python3.10 /home/asa/code/Git/Compiler_Design/minesweeper/test_output.py
**#############################################################################*
################################################################################
####################################*###########################################
##################################################################*#*###########
*################################################################**############*
```

این بار به hint مقدار True میدهیم تا از راهنمایی برخوردار شویم



همانطور که میبینیم نود hint (دومی از چپ) با مقدار True اضافه شده و کد برنامه به صورت زیر نوشته می شود



همانظور که میبینیم مقدار متغیر hint True میشود و تابع hint_fuc نیز تعریف میشود

```
bombs[0][1] = True
hint_func(x: 0, y: 1)

bombs[0][0] = True
hint_func(x: 0, y: 0)

bombs[0][79] = True
hint_func(x: 0, y: 79)

bombs[4][79] = True
hint_func(x: 4, y: 79)

bombs[4][0] = True
hint_func(x: 4, y: 0)

bombs[2][36] = True
hint_func(x: 2, y: 36)

bombs[3][66] = True
hint_func(x: 3, y: 66)

bombs[3][68] = True
hint_func(x: 3, y: 68)

bombs[4][65] = True
hint_func(x: 4, y: 65)

bombs[4][66] = True
hint_func(x: 4, y: 66)
```

همچنین به ازای تعریف هر بمب تابع hint_func متقابل آن نیز صدا زده میشود و خروجی در نهایت به شکل زیر تبدیل میشود.

```
**1###########################################################################1*
221################################111########################################11
###################################1*1###########################11211##########
11#################################111##########################13*3*1########11
*1##############################################################1**311########1*
```