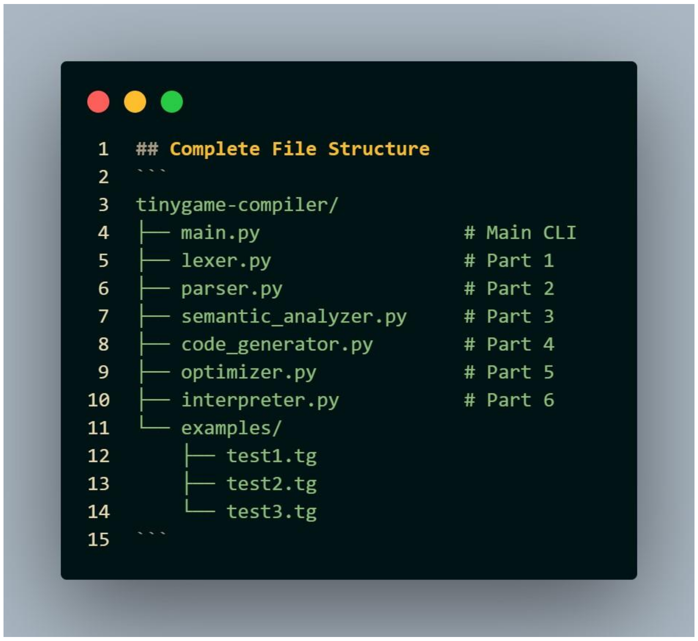
Compiler Construction Project **Part-3 (Implementation)**

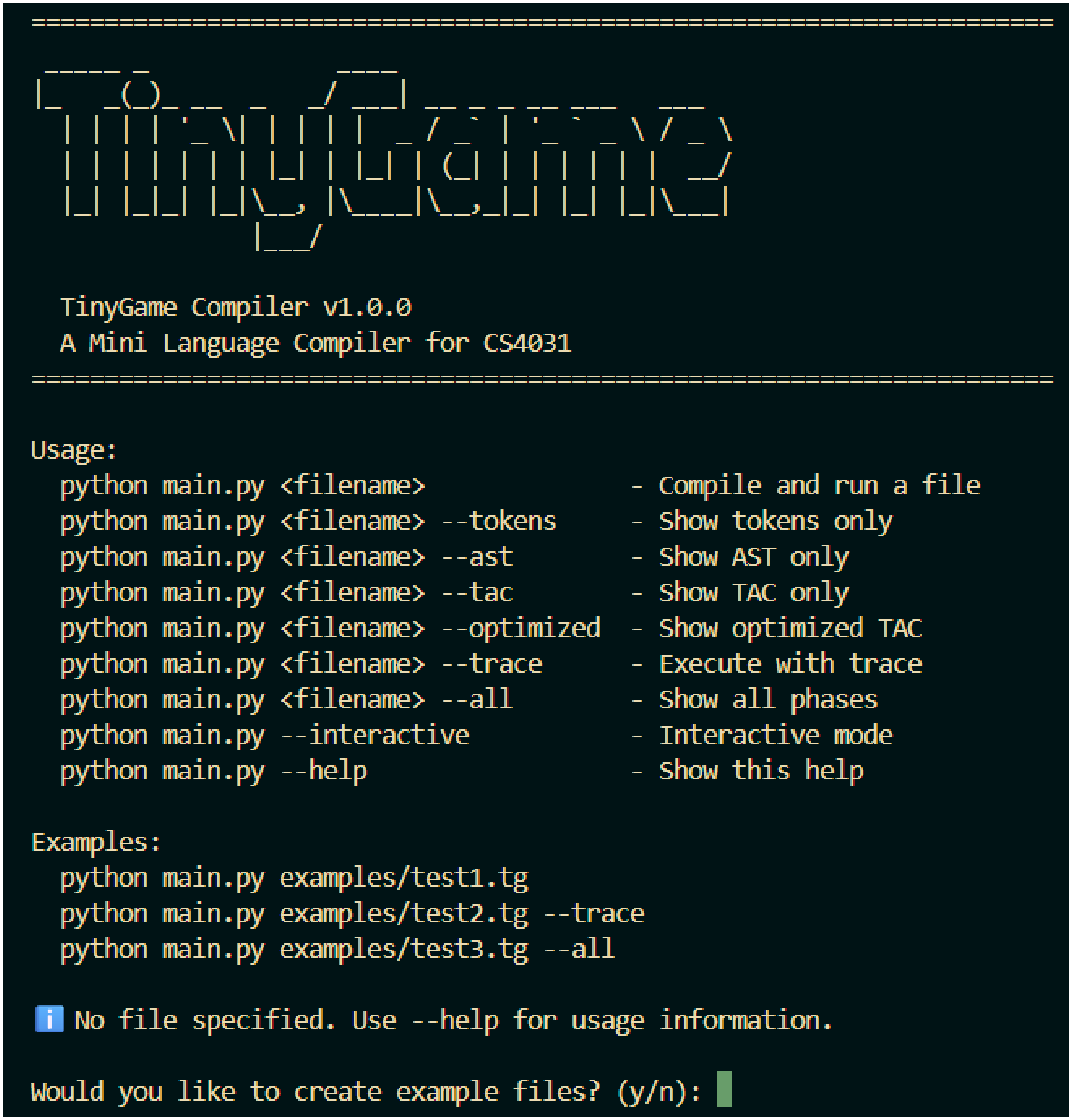
GitHub Repository:
https://github.com/Anasahmed-10/Compiler-Construction-Project.git

Code Structure



```
## Complete File Structure
```
tinygame-compiler/
├── main.py   # Main CLI
├── lexer.py   # Part 1
├── parser.py   # Part 2
├── semantic_analyzer.py   # Part 3
├── code_generator.py   # Part 4
├── optimizer.py   # Part 5
├── interpreter.py   # Part 6
└── examples/
    ├── test1.tg
    ├── test2.tg
    └── test3.tg
```
```

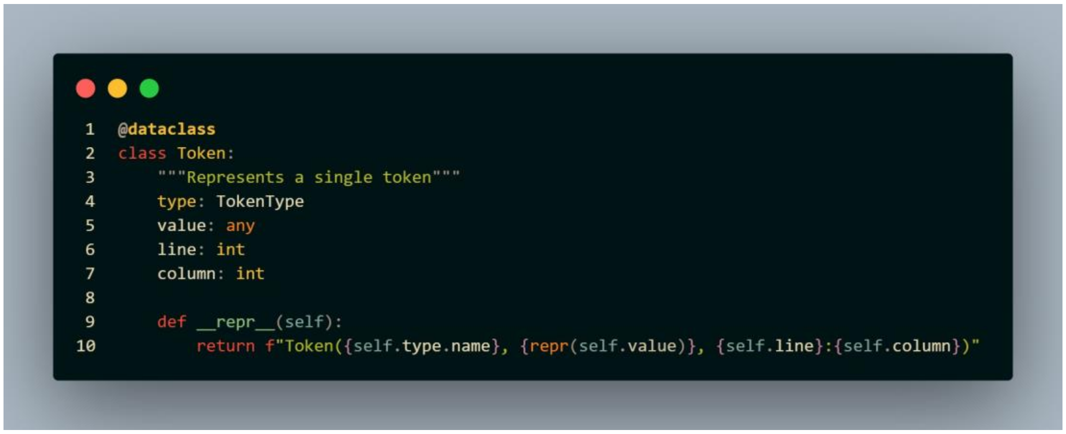
UI



```
======================================================================
 _____ _              ____
|_   _(_)_ __  _   _ / ___| __ _ _ __ ___   ___
  | | | | '_ \| | | | |  _ / _` | '_ ` _ \ / _ \
  | | | | | | | |_| | |_| | (_| | | | | | |  __/
  |_| |_|_| |_|\__, |\____|\__,_|_| |_| |_|\___|
               |___/

  TinyGame Compiler v1.0.0
  A Mini Language Compiler for CS4031
======================================================================

Usage:
  python main.py <filename>              - Compile and run a file
  python main.py <filename> --tokens     - Show tokens only
  python main.py <filename> --ast        - Show AST only
  python main.py <filename> --tac        - Show TAC only
  python main.py <filename> --optimized  - Show optimized TAC
  python main.py <filename> --trace      - Execute with trace
  python main.py <filename> --all        - Show all phases
  python main.py --interactive           - Interactive mode
  python main.py --help                  - Show this help

Examples:
  python main.py examples/test1.tg
  python main.py examples/test2.tg --trace
  python main.py examples/test3.tg --all

ℹ️ No file specified. Use --help for usage information.

Would you like to create example files? (y/n): 
```

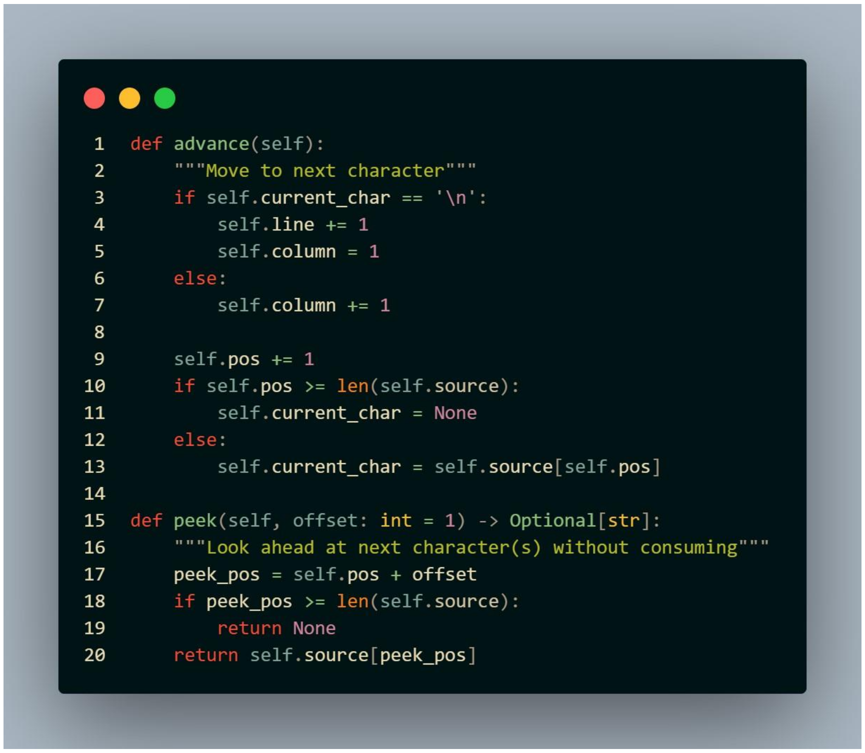
Stage 1 - Lexical Analysis

Token DataClass

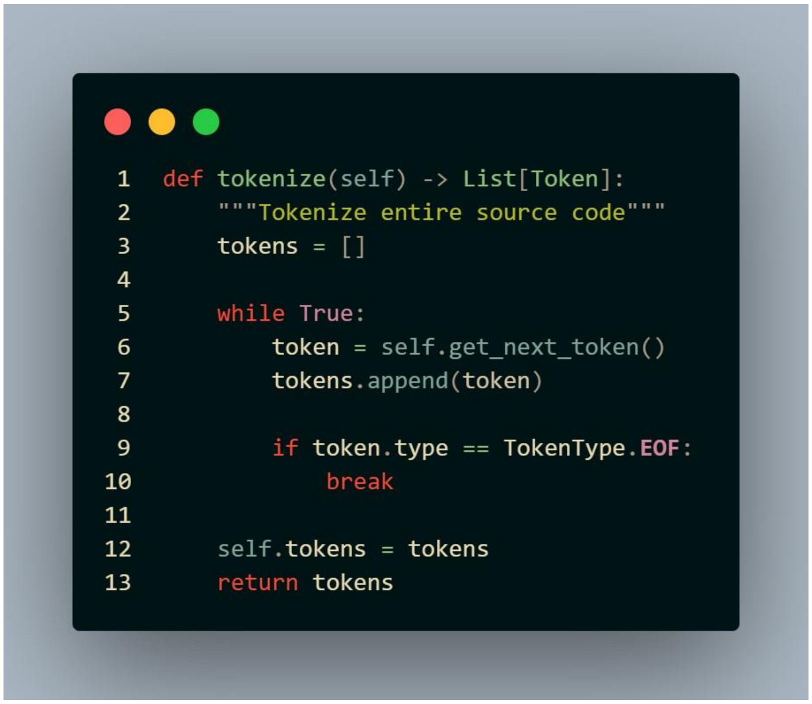


```python
@dataclass
class Token:
    """Represents a single token"""
    type: TokenType
    value: any
    line: int
    column: int

    def __repr__(self):
        return f"Token({self.type.name}, {repr(self.value)}, {self.line}:{self.column})"
```

Functions

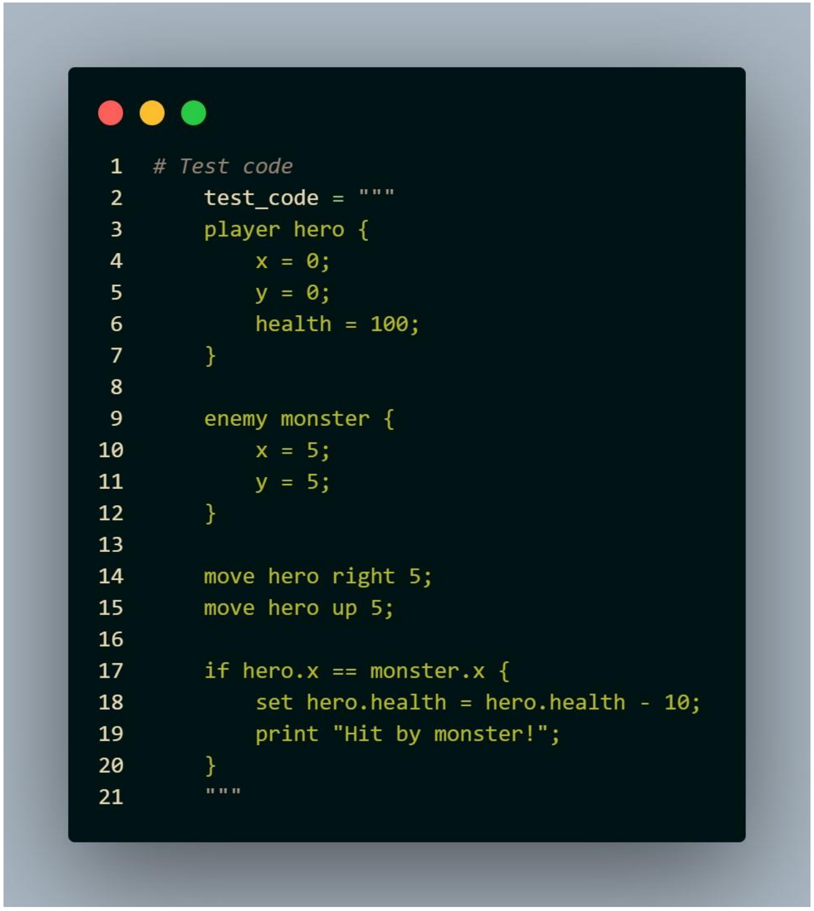


```python
def advance(self):
    """Move to next character"""
    if self.current_char == '\n':
        self.line += 1
        self.column = 1
    else:
        self.column += 1

    self.pos += 1
    if self.pos >= len(self.source):
        self.current_char = None
    else:
        self.current_char = self.source[self.pos]

def peek(self, offset: int = 1) -> Optional[str]:
    """Look ahead at next character(s) without consuming"""
    peek_pos = self.pos + offset
    if peek_pos >= len(self.source):
        return None
    return self.source[peek_pos]
```



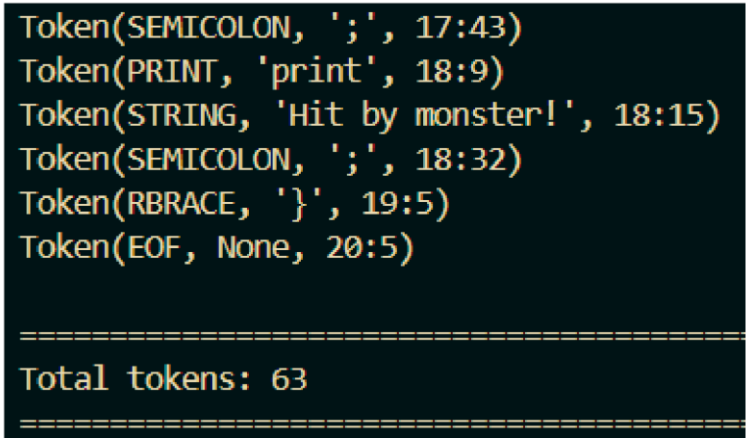
```python
def tokenize(self) -> List[Token]:
    """Tokenize entire source code"""
    tokens = []

    while True:
        token = self.get_next_token()
        tokens.append(token)

        if token.type == TokenType.EOF:
            break

    self.tokens = tokens
    return tokens
```

Input



```python
# Test code
    test_code = """
    player hero {
        x = 0;
        y = 0;
        health = 100;
    }

    enemy monster {
        x = 5;
        y = 5;
    }

    move hero right 5;
    move hero up 5;

    if hero.x == monster.x {
        set hero.health = hero.health - 10;
        print "Hit by monster!";
    }
    """
```

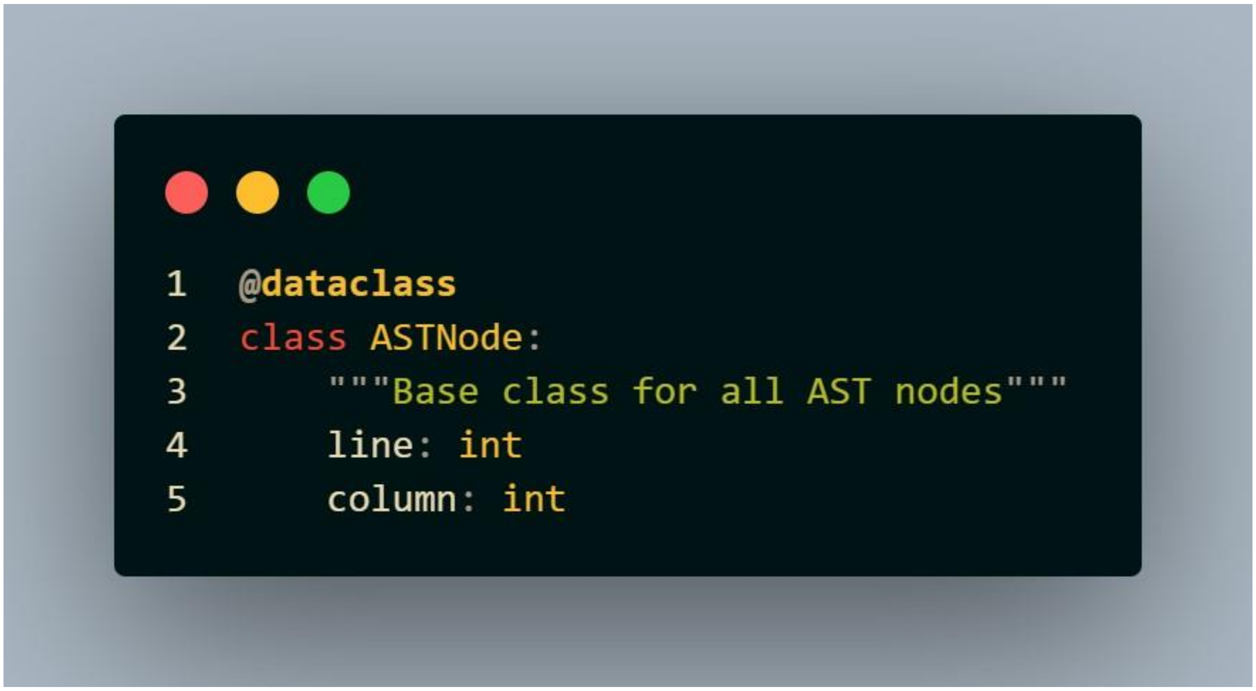
Output



```
Token(SEMICOLON, ';', 17:43)
Token(PRINT, 'print', 18:9)
Token(STRING, 'Hit by monster!', 18:15)
Token(SEMICOLON, ';', 18:32)
Token(RBRACE, '}', 19:5)
Token(EOF, None, 20:5)


=============================================
Total tokens: 63
=============================================
```

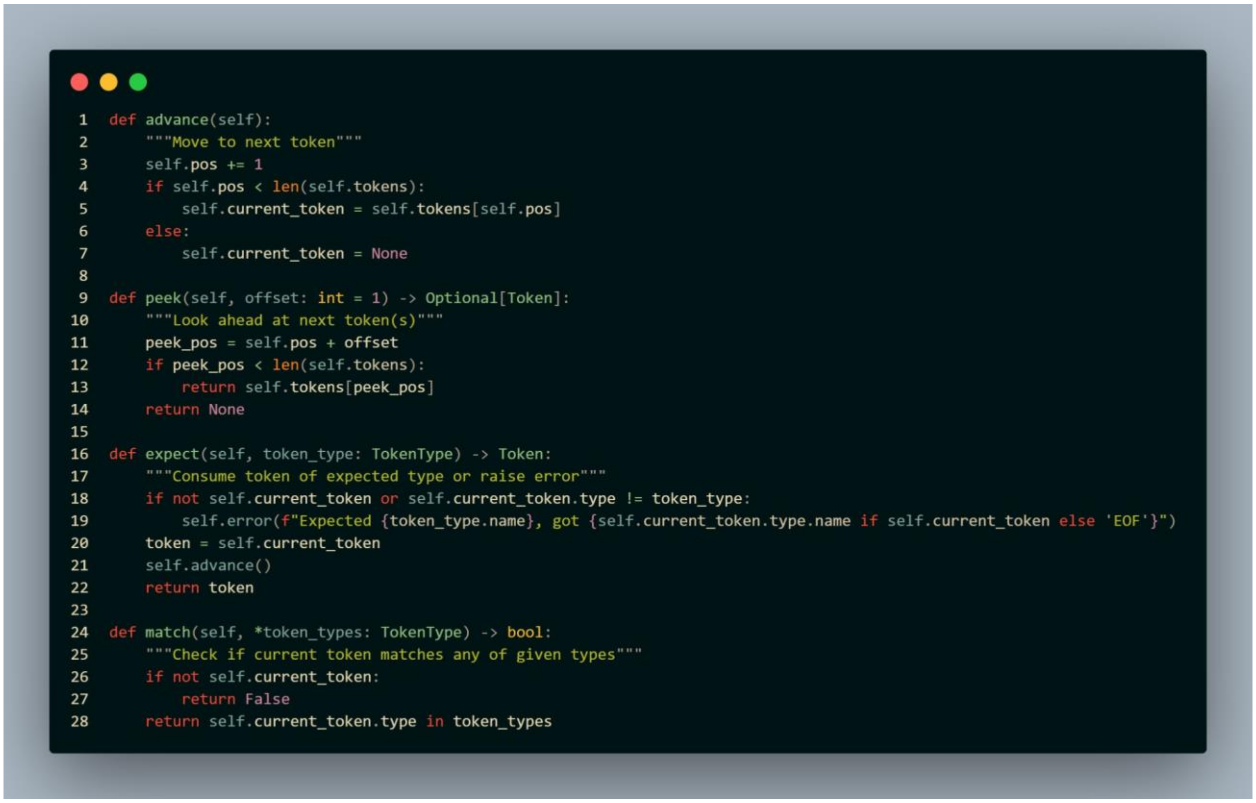
Stage 2 - Syntax Analysis

ASTNode DataClass



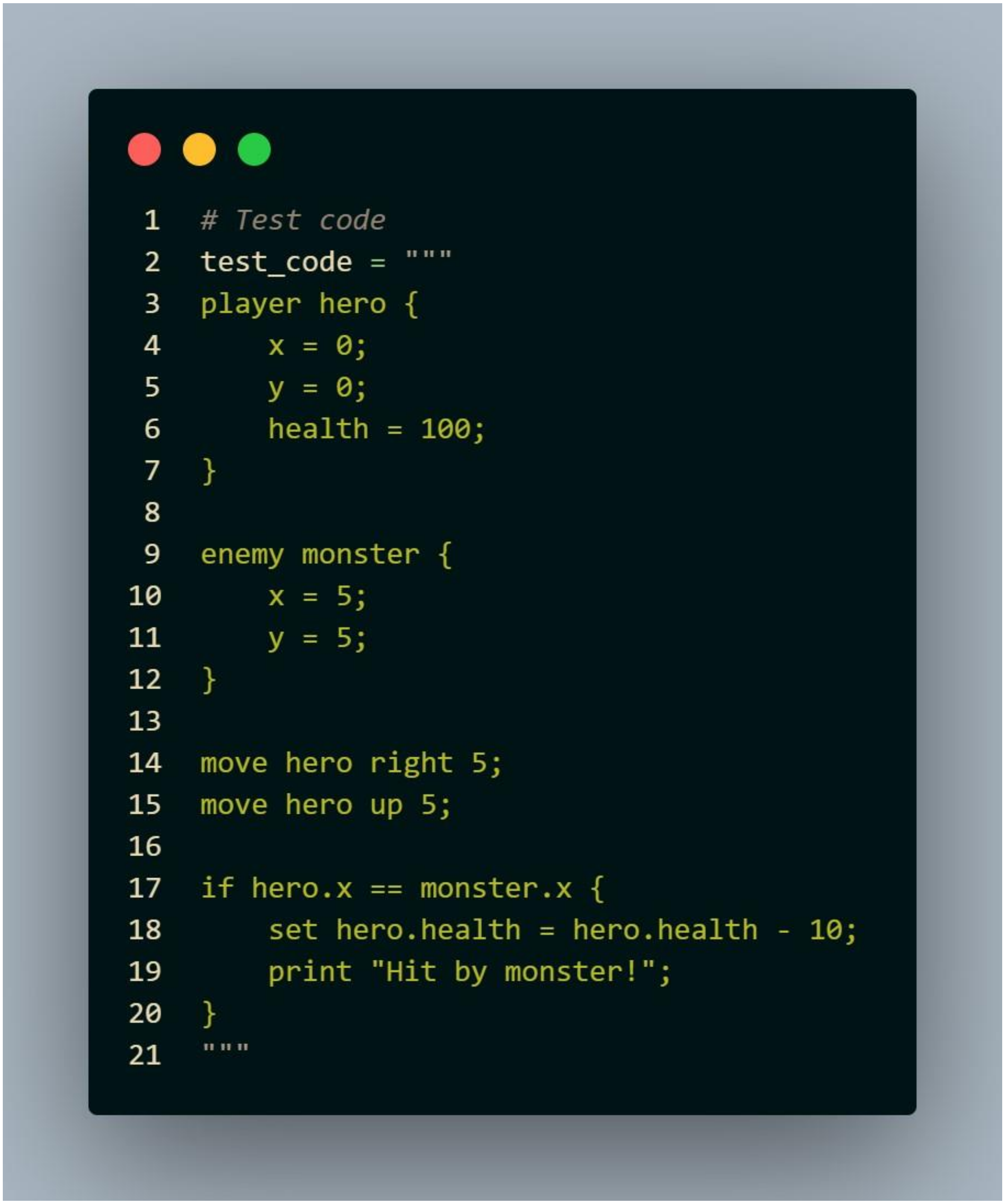
```python
@dataclass
class ASTNode:
    """Base class for all AST nodes"""
    line: int
    column: int
```

Functions



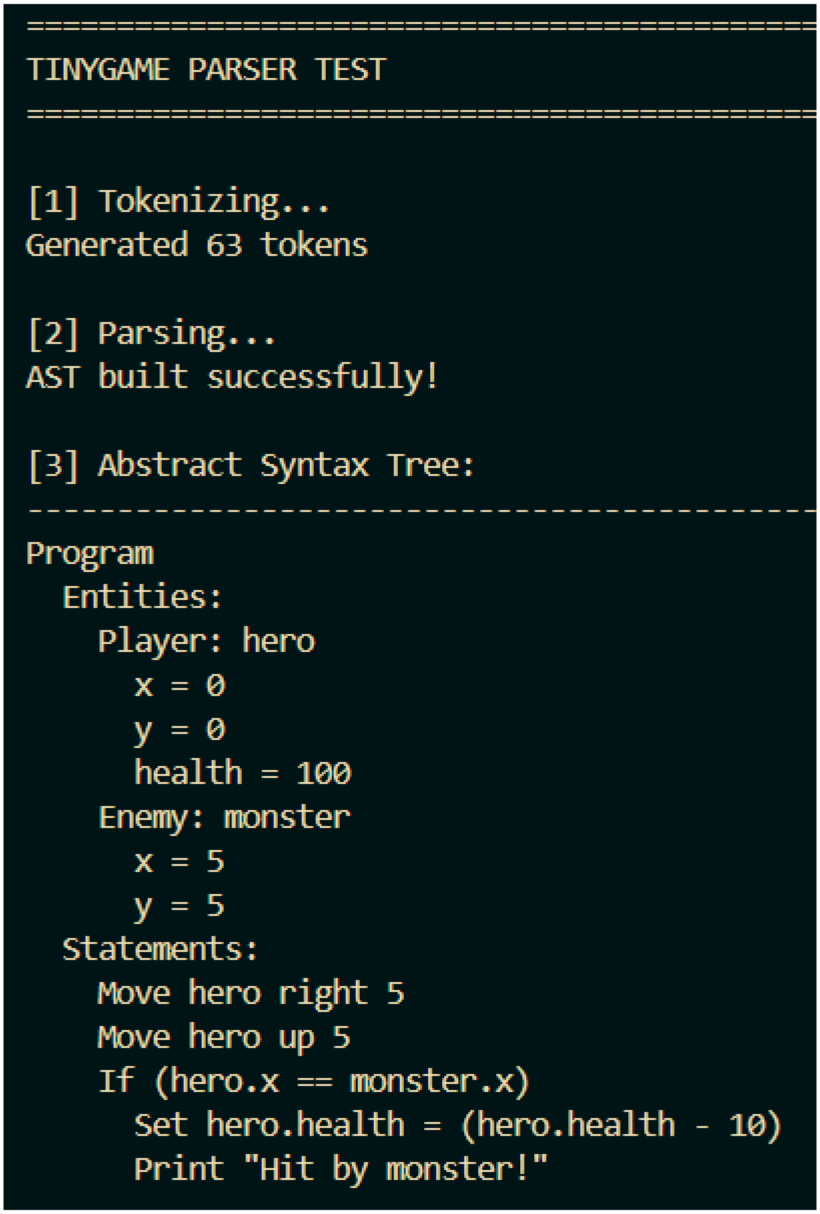
```python
def advance(self):
    """Move to next token"""
    self.pos += 1
    if self.pos < len(self.tokens):
        self.current_token = self.tokens[self.pos]
    else:
        self.current_token = None

def peek(self, offset: int = 1) -> Optional[Token]:
    """Look ahead at next token(s)"""
    peek_pos = self.pos + offset
    if peek_pos < len(self.tokens):
        return self.tokens[peek_pos]
    return None

def expect(self, token_type: TokenType) -> Token:
    """Consume token of expected type or raise error"""
    if not self.current_token or self.current_token.type != token_type:
        self.error(f"Expected {token_type.name}, got {self.current_token.type.name if self.current_token else 'EOF'}")
    token = self.current_token
    self.advance()
    return token

def match(self, *token_types: TokenType) -> bool:
    """Check if current token matches any of given types"""
    if not self.current_token:
        return False
    return self.current_token.type in token_types
```

Input



```
# Test code
test_code = """
player hero {
    x = 0;
    y = 0;
    health = 100;
}

enemy monster {
    x = 5;
    y = 5;
}

move hero right 5;
move hero up 5;

if hero.x == monster.x {
    set hero.health = hero.health - 10;
    print "Hit by monster!";
}
"""
```

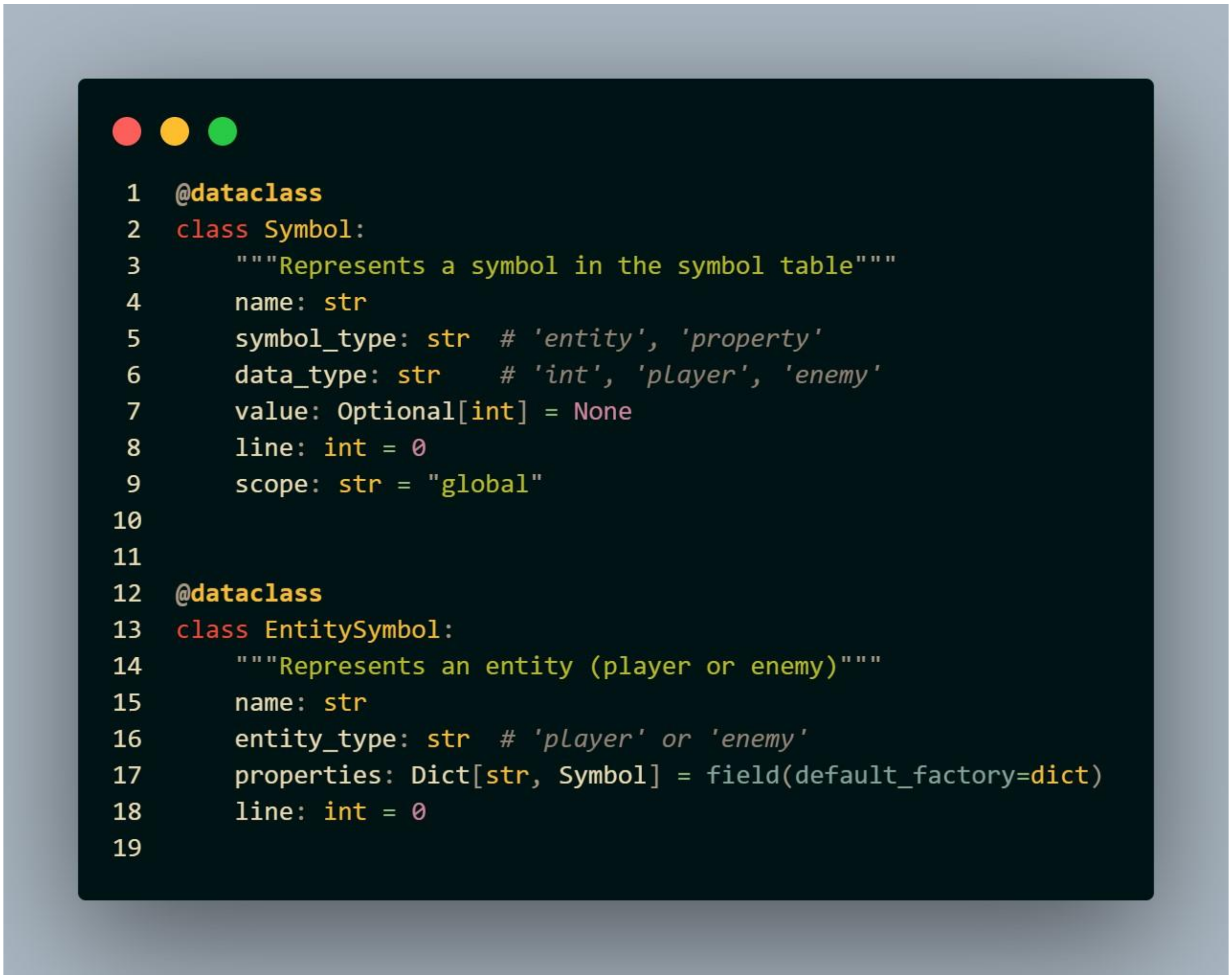
Output



```
============================================
TINYGAME PARSER TEST
============================================

[1] Tokenizing...
Generated 63 tokens

[2] Parsing...
AST built successfully!

[3] Abstract Syntax Tree:
--------------------------------------------
Program
  Entities:
    Player: hero
      x = 0
      y = 0
      health = 100
    Enemy: monster
      x = 5
      y = 5
  Statements:
    Move hero right 5
    Move hero up 5
    If (hero.x == monster.x)
      Set hero.health = (hero.health - 10)
      Print "Hit by monster!"
```

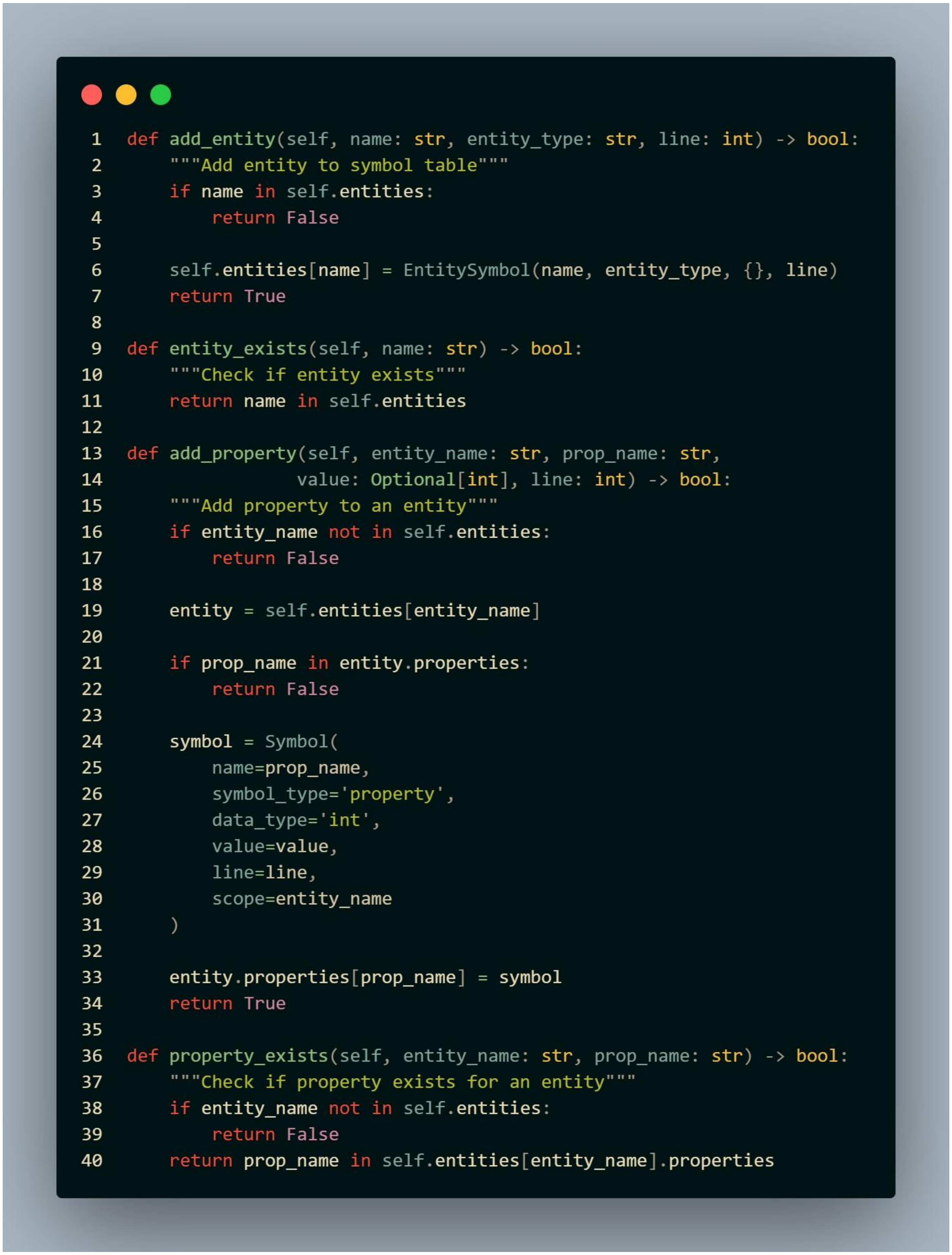
Stage 3 - Semantic Analysis

Symbol DataClass



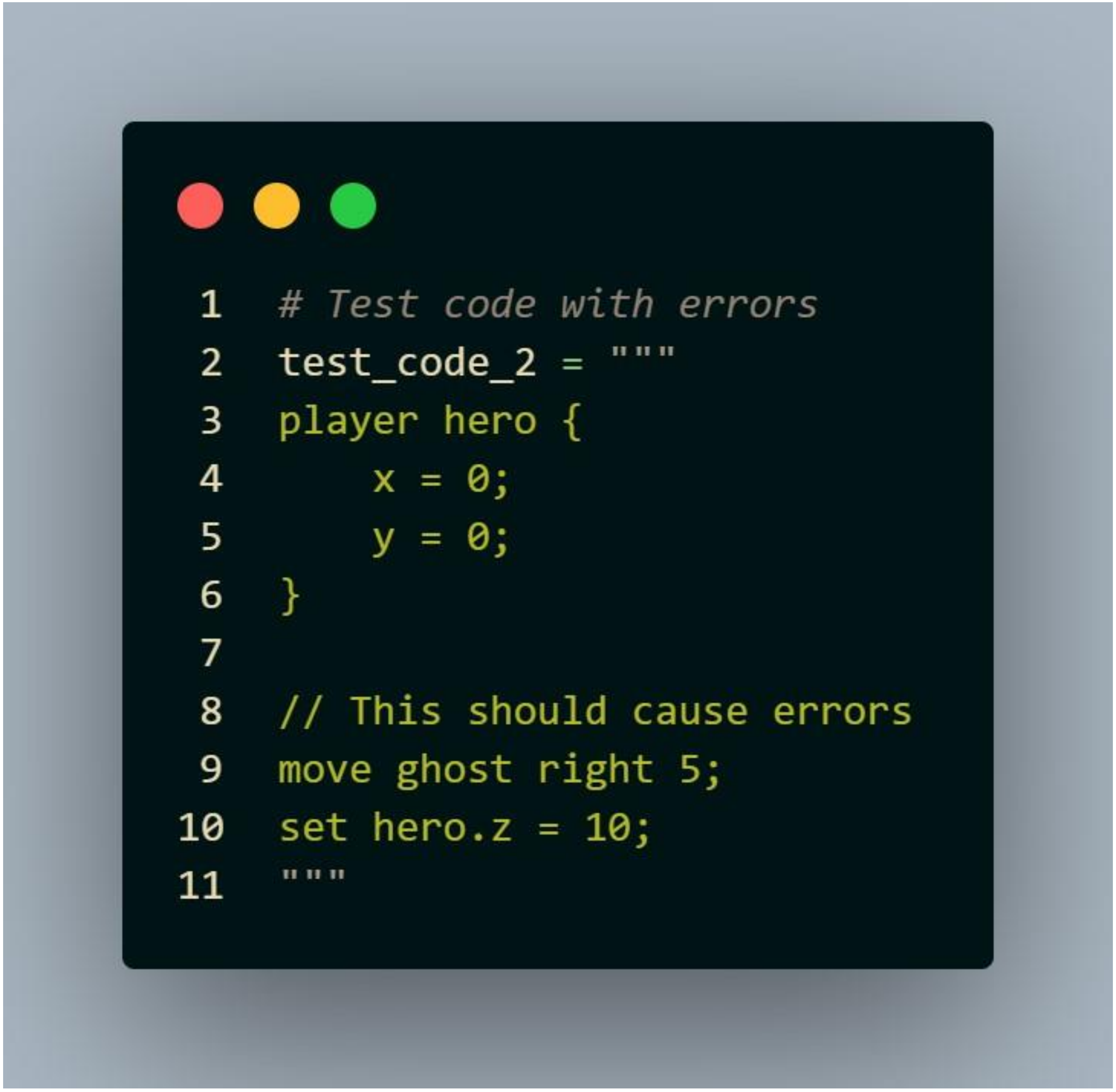
```python
@dataclass
class Symbol:
    """Represents a symbol in the symbol table"""
    name: str
    symbol_type: str  # 'entity', 'property'
    data_type: str    # 'int', 'player', 'enemy'
    value: Optional[int] = None
    line: int = 0
    scope: str = "global"


@dataclass
class EntitySymbol:
    """Represents an entity (player or enemy)"""
    name: str
    entity_type: str  # 'player' or 'enemy'
    properties: Dict[str, Symbol] = field(default_factory=dict)
    line: int = 0

```

Functions



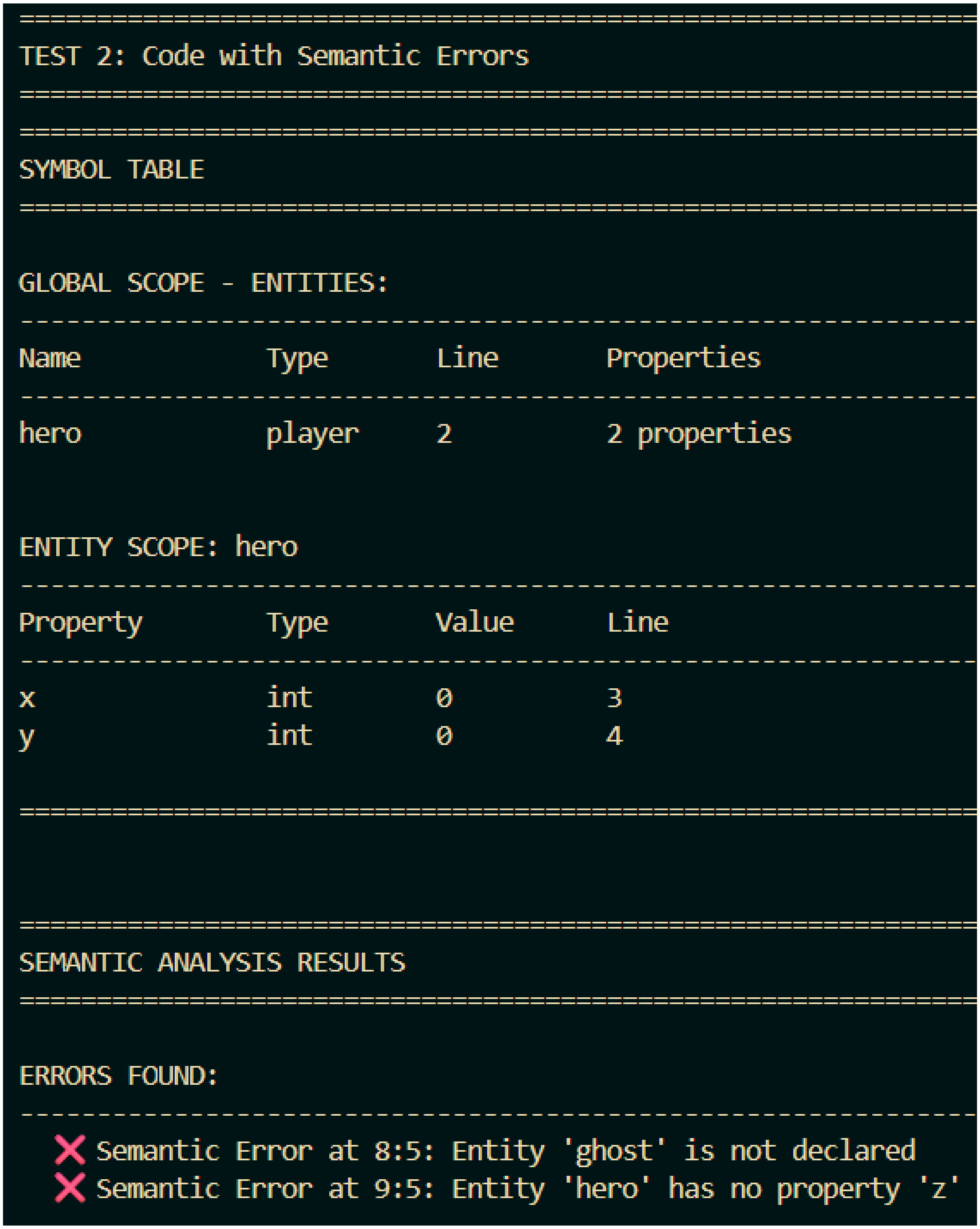
```python
def add_entity(self, name: str, entity_type: str, line: int) -> bool:
    """Add entity to symbol table"""
    if name in self.entities:
        return False

    self.entities[name] = EntitySymbol(name, entity_type, {}, line)
    return True

def entity_exists(self, name: str) -> bool:
    """Check if entity exists"""
    return name in self.entities

def add_property(self, entity_name: str, prop_name: str,
                value: Optional[int], line: int) -> bool:
    """Add property to an entity"""
    if entity_name not in self.entities:
        return False

    entity = self.entities[entity_name]

    if prop_name in entity.properties:
        return False

    symbol = Symbol(
        name=prop_name,
        symbol_type='property',
        data_type='int',
        value=value,
        line=line,
        scope=entity_name
    )

    entity.properties[prop_name] = symbol
    return True

def property_exists(self, entity_name: str, prop_name: str) -> bool:
    """Check if property exists for an entity"""
    if entity_name not in self.entities:
        return False
    return prop_name in self.entities[entity_name].properties
```

Input



```
# Test code with errors
test_code_2 = """
player hero {
    x = 0;
    y = 0;
}

// This should cause errors
move ghost right 5;
set hero.z = 10;
"""
```

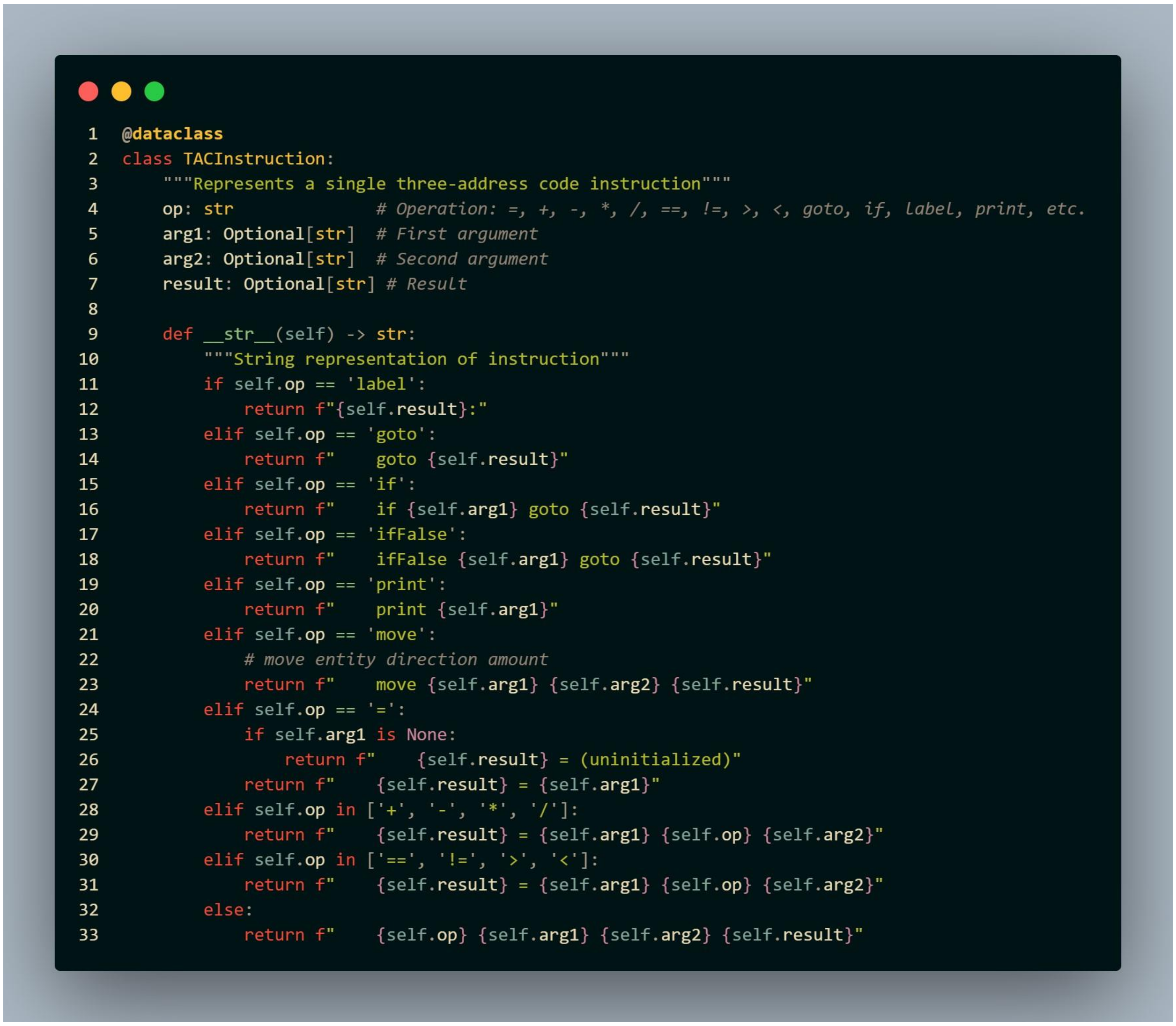
Output



```
==============================================================
TEST 2: Code with Semantic Errors
==============================================================
==============================================================
SYMBOL TABLE
==============================================================

GLOBAL SCOPE - ENTITIES:
--------------------------------------------------------------
Name            Type       Line       Properties
--------------------------------------------------------------
hero            player     2          2 properties


ENTITY SCOPE: hero
--------------------------------------------------------------
Property        Type       Value      Line
--------------------------------------------------------------
x               int        0          3
y               int        0          4

==============================================================



==============================================================
SEMANTIC ANALYSIS RESULTS
==============================================================

ERRORS FOUND:
--------------------------------------------------------------
  ❌ Semantic Error at 8:5: Entity 'ghost' is not declared
  ❌ Semantic Error at 9:5: Entity 'hero' has no property 'z'
```

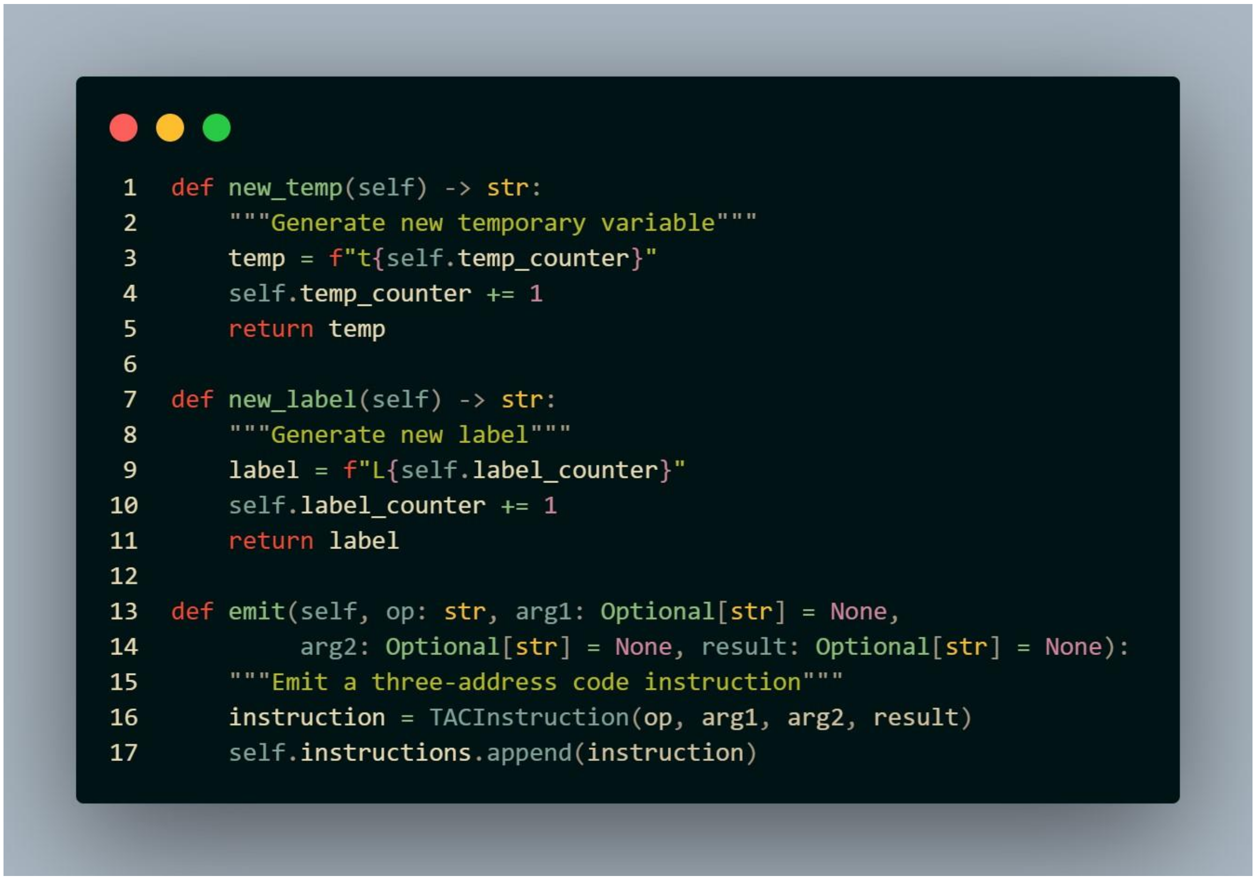
Stage 4 - Intermediate Code Generation

TAC Instruction DataClass



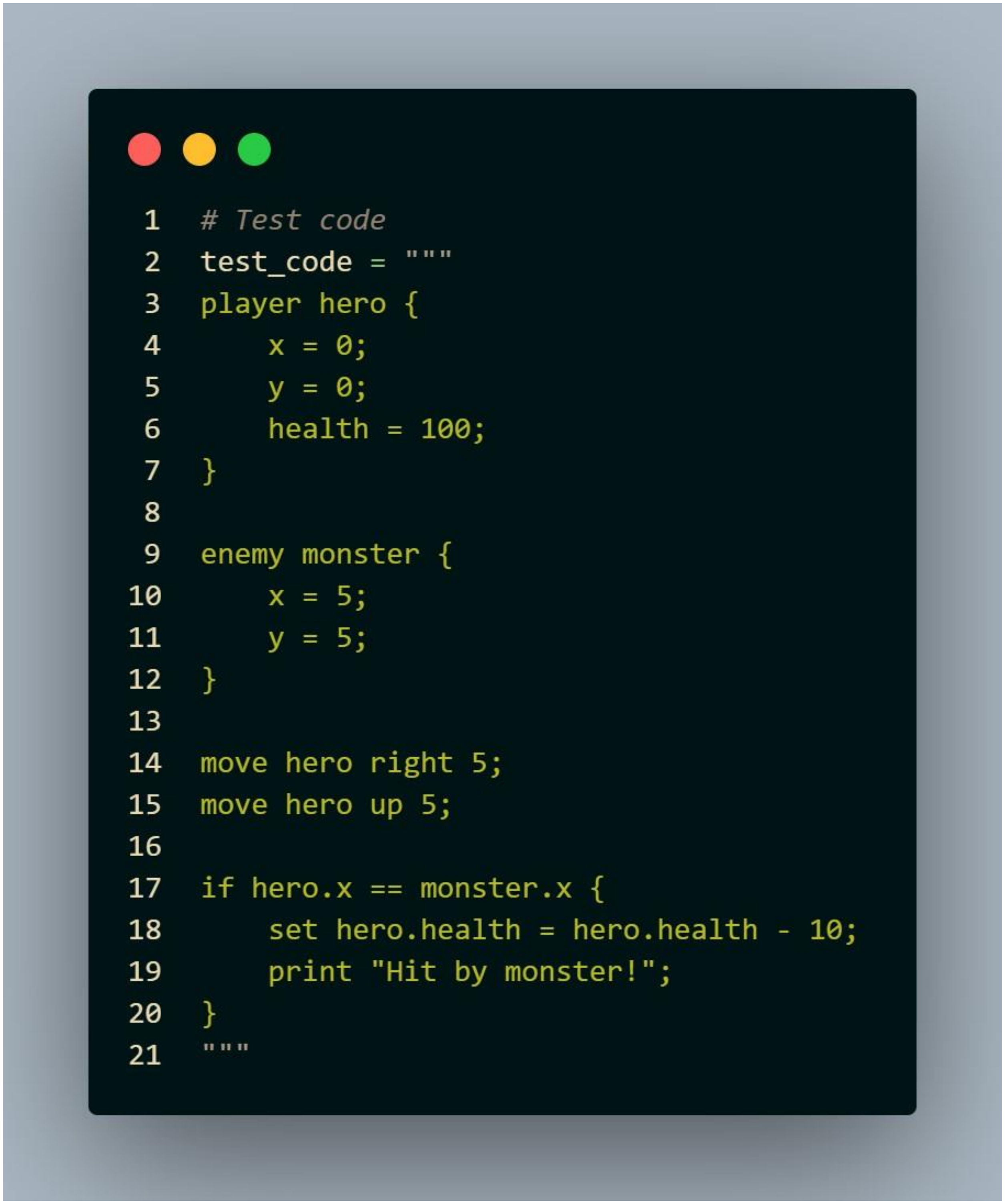
```python
@dataclass
class TACInstruction:
    """Represents a single three-address code instruction"""
    op: str              # Operation: =, +, -, *, /, ==, !=, >, <, goto, if, label, print, etc.
    arg1: Optional[str]  # First argument
    arg2: Optional[str]  # Second argument
    result: Optional[str] # Result

    def __str__(self) -> str:
        """String representation of instruction"""
        if self.op == 'label':
            return f"{self.result}:"
        elif self.op == 'goto':
            return f"    goto {self.result}"
        elif self.op == 'if':
            return f"    if {self.arg1} goto {self.result}"
        elif self.op == 'ifFalse':
            return f"    ifFalse {self.arg1} goto {self.result}"
        elif self.op == 'print':
            return f"    print {self.arg1}"
        elif self.op == 'move':
            # move entity direction amount
            return f"    move {self.arg1} {self.arg2} {self.result}"
        elif self.op == '=':
            if self.arg1 is None:
                return f"    {self.result} = (uninitialized)"
            return f"    {self.result} = {self.arg1}"
        elif self.op in ['+', '-', '*', '/']:
            return f"    {self.result} = {self.arg1} {self.op} {self.arg2}"
        elif self.op in ['==', '!=', '>', '<']:
            return f"    {self.result} = {self.arg1} {self.op} {self.arg2}"
        else:
            return f"    {self.op} {self.arg1} {self.arg2} {self.result}"
```

Functions



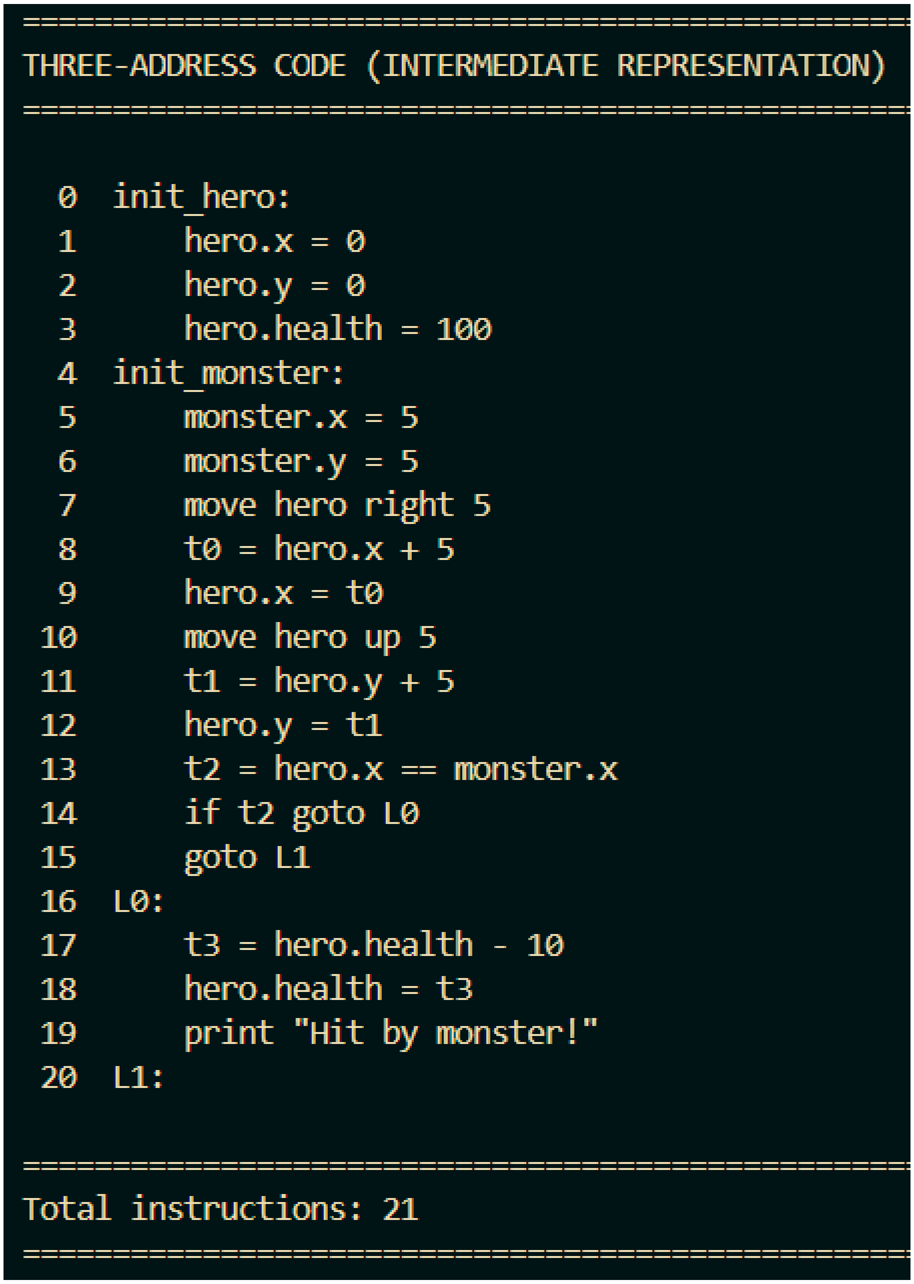
```python
def new_temp(self) -> str:
    """Generate new temporary variable"""
    temp = f"t{self.temp_counter}"
    self.temp_counter += 1
    return temp

def new_label(self) -> str:
    """Generate new label"""
    label = f"L{self.label_counter}"
    self.label_counter += 1
    return label

def emit(self, op: str, arg1: Optional[str] = None,
         arg2: Optional[str] = None, result: Optional[str] = None):
    """Emit a three-address code instruction"""
    instruction = TACInstruction(op, arg1, arg2, result)
    self.instructions.append(instruction)
```

Input

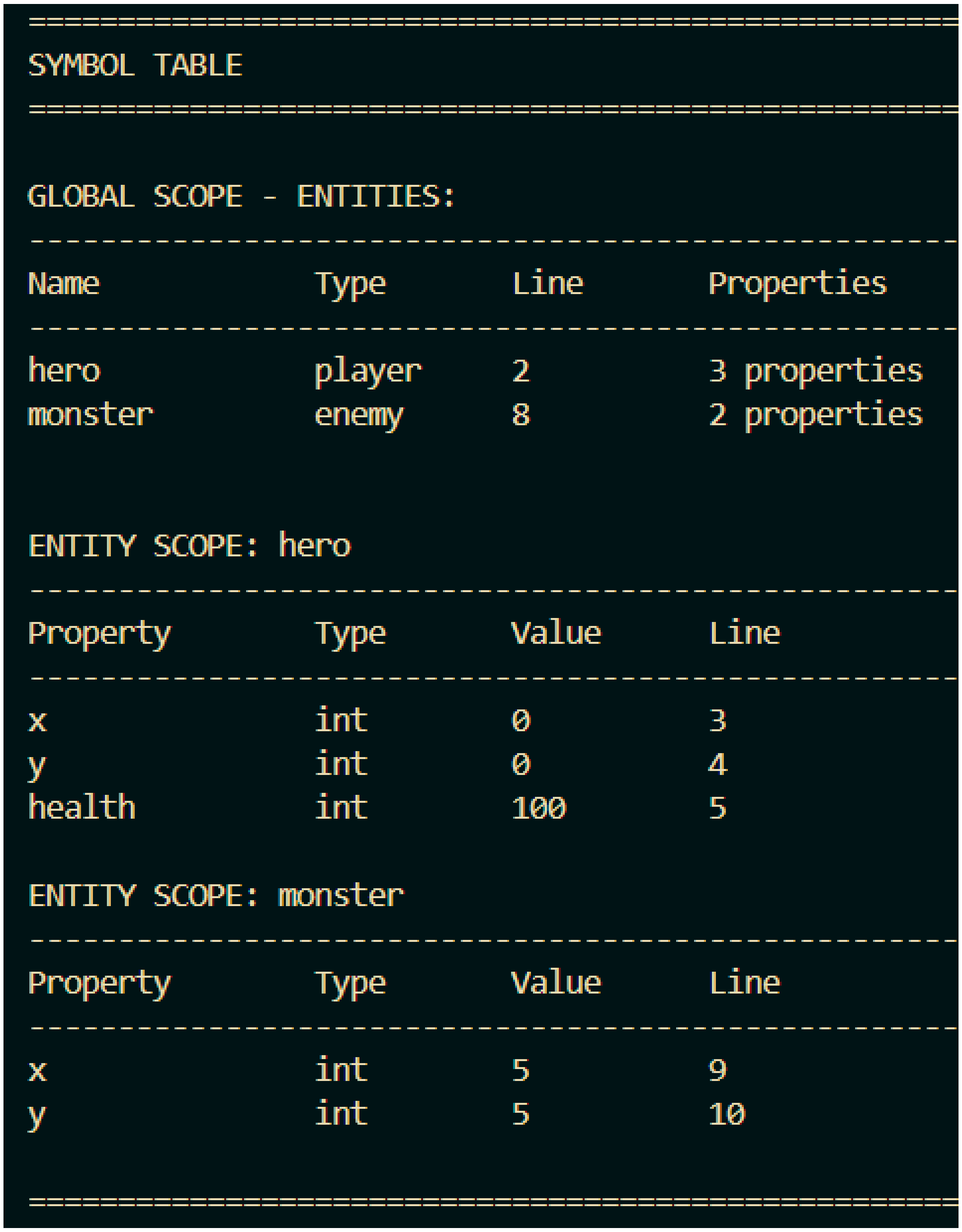


```
# Test code
test_code = """
player hero {
    x = 0;
    y = 0;
    health = 100;
}

enemy monster {
    x = 5;
    y = 5;
}

move hero right 5;
move hero up 5;

if hero.x == monster.x {
    set hero.health = hero.health - 10;
    print "Hit by monster!";
}
"""
```

Output



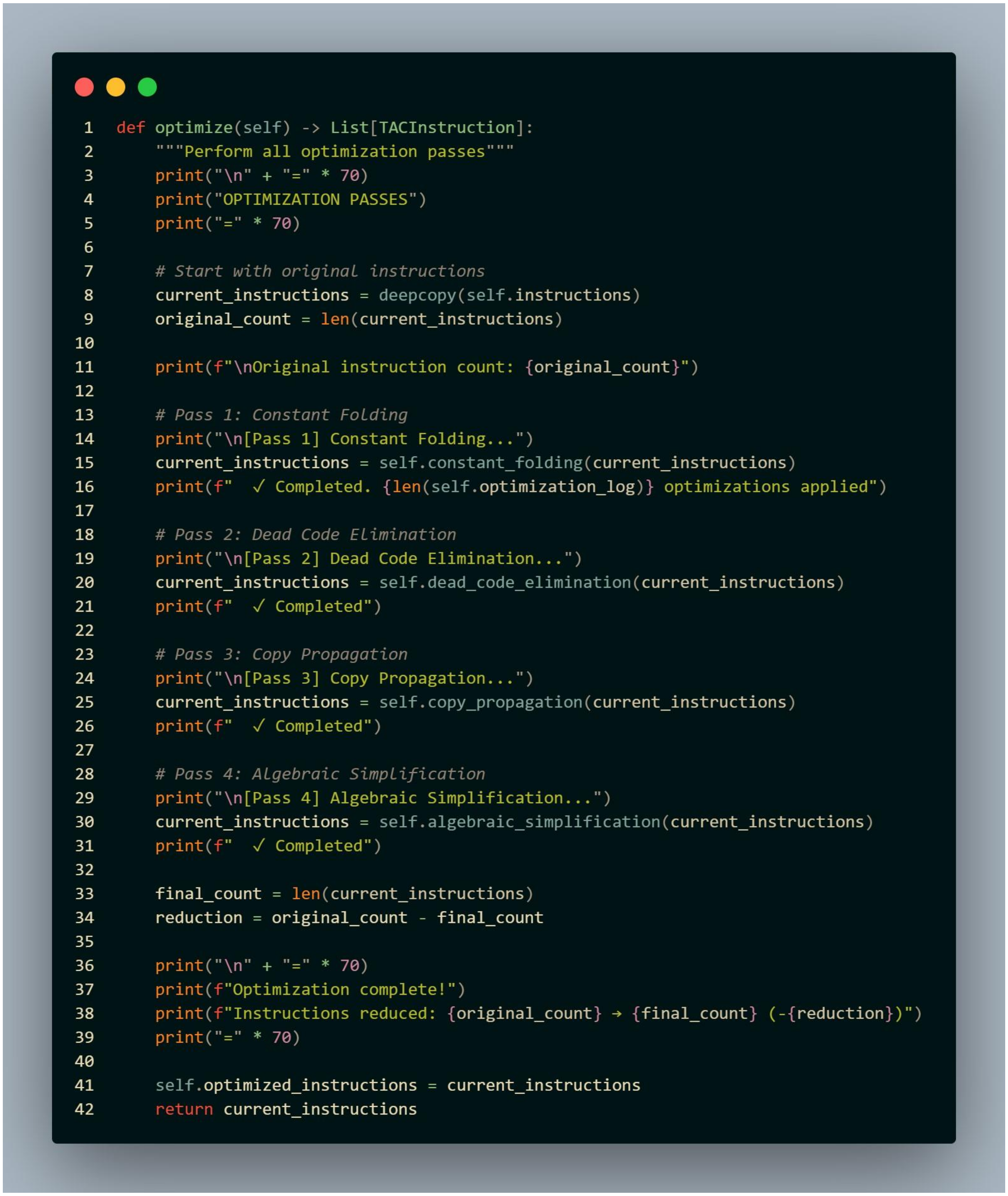
```
==================================================
THREE-ADDRESS CODE (INTERMEDIATE REPRESENTATION)
==================================================

  0  init_hero:
  1      hero.x = 0
  2      hero.y = 0
  3      hero.health = 100
  4  init_monster:
  5      monster.x = 5
  6      monster.y = 5
  7      move hero right 5
  8      t0 = hero.x + 5
  9      hero.x = t0
 10      move hero up 5
 11      t1 = hero.y + 5
 12      hero.y = t1
 13      t2 = hero.x == monster.x
 14      if t2 goto L0
 15      goto L1
 16  L0:
 17      t3 = hero.health - 10
 18      hero.health = t3
 19      print "Hit by monster!"
 20  L1:

==================================================
Total instructions: 21
==================================================
```



```
====================================================
SYMBOL TABLE
====================================================

GLOBAL SCOPE - ENTITIES:
----------------------------------------------------
Name            Type       Line       Properties
----------------------------------------------------
hero            player     2          3 properties
monster         enemy      8          2 properties


ENTITY SCOPE: hero
----------------------------------------------------
Property        Type       Value      Line
----------------------------------------------------
x               int        0          3
y               int        0          4
health          int        100        5

ENTITY SCOPE: monster
----------------------------------------------------
Property        Type       Value      Line
----------------------------------------------------
x               int        5          9
y               int        5          10

====================================================
```

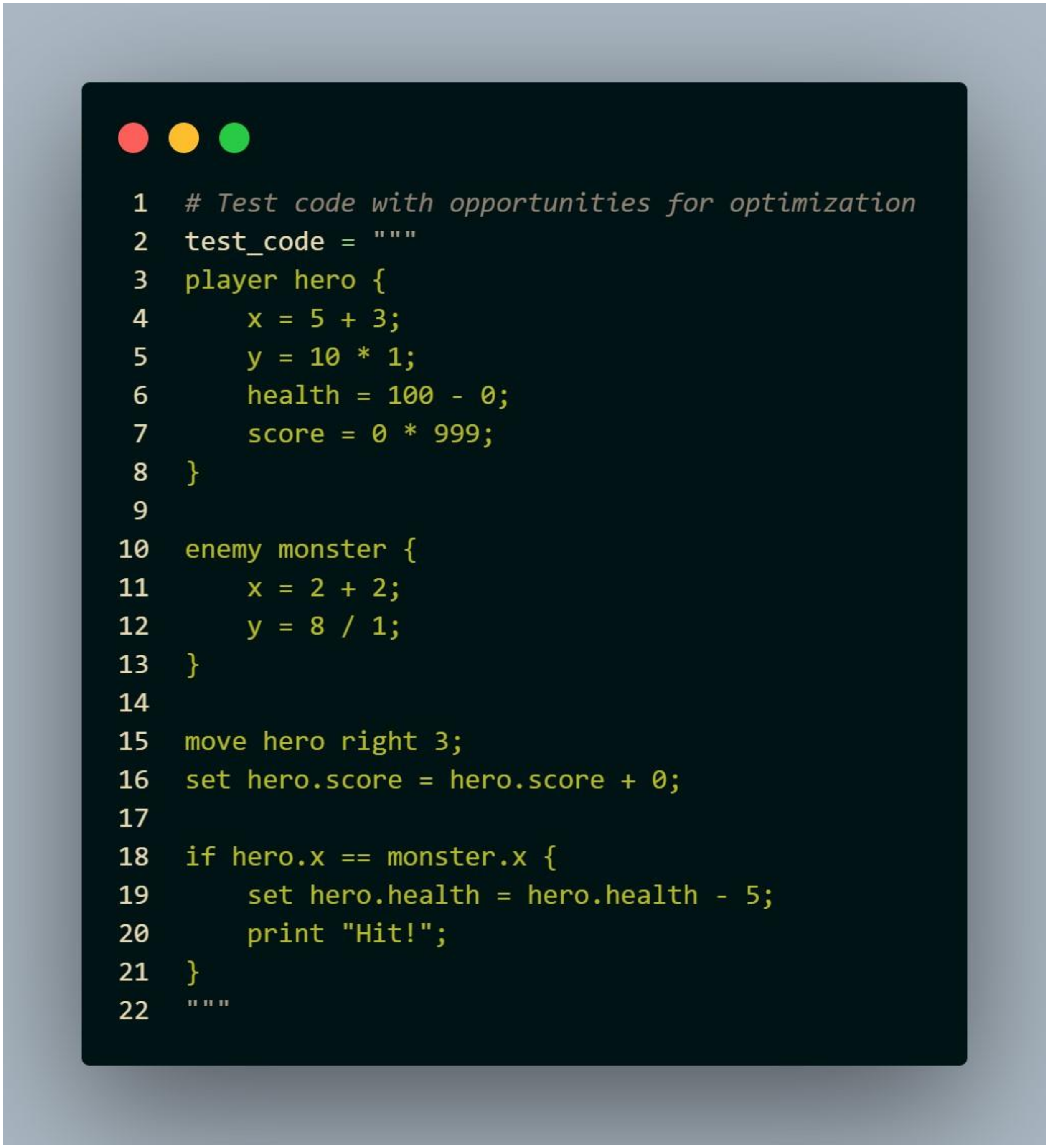
Stage 5 - Code Optimization

Functions



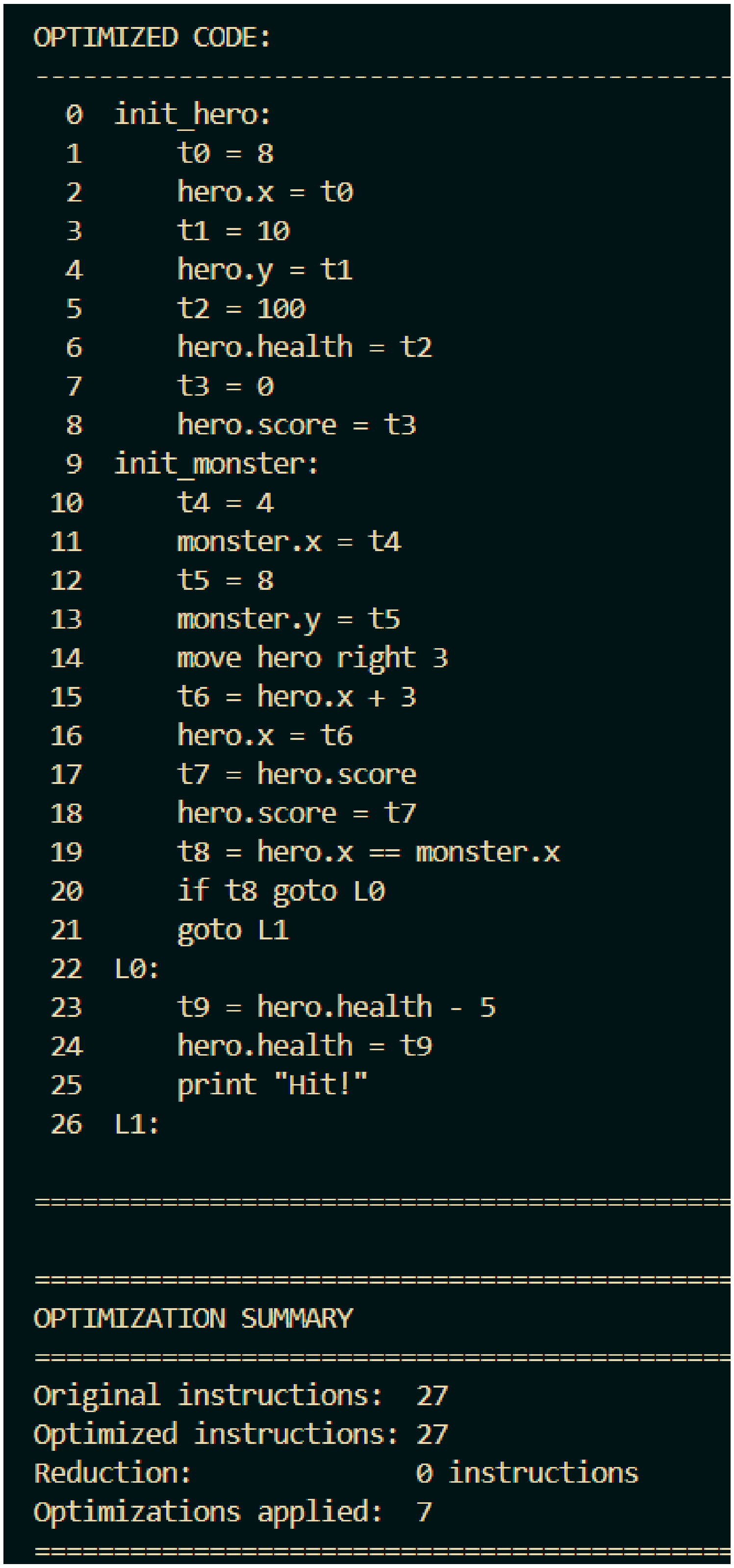
```python
def optimize(self) -> List[TACInstruction]:
    """Perform all optimization passes"""
    print("\n" + "=" * 70)
    print("OPTIMIZATION PASSES")
    print("=" * 70)

    # Start with original instructions
    current_instructions = deepcopy(self.instructions)
    original_count = len(current_instructions)

    print(f"\nOriginal instruction count: {original_count}")

    # Pass 1: Constant Folding
    print("\n[Pass 1] Constant Folding...")
    current_instructions = self.constant_folding(current_instructions)
    print(f"  ✓ Completed. {len(self.optimization_log)} optimizations applied")

    # Pass 2: Dead Code Elimination
    print("\n[Pass 2] Dead Code Elimination...")
    current_instructions = self.dead_code_elimination(current_instructions)
    print(f"  ✓ Completed")

    # Pass 3: Copy Propagation
    print("\n[Pass 3] Copy Propagation...")
    current_instructions = self.copy_propagation(current_instructions)
    print(f"  ✓ Completed")

    # Pass 4: Algebraic Simplification
    print("\n[Pass 4] Algebraic Simplification...")
    current_instructions = self.algebraic_simplification(current_instructions)
    print(f"  ✓ Completed")

    final_count = len(current_instructions)
    reduction = original_count - final_count

    print("\n" + "=" * 70)
    print(f"Optimization complete!")
    print(f"Instructions reduced: {original_count} → {final_count} (-{reduction})")
    print("=" * 70)

    self.optimized_instructions = current_instructions
    return current_instructions
```

Input



```
# Test code with opportunities for optimization
test_code = """
player hero {
    x = 5 + 3;
    y = 10 * 1;
    health = 100 - 0;
    score = 0 * 999;
}

enemy monster {
    x = 2 + 2;
    y = 8 / 1;
}

move hero right 3;
set hero.score = hero.score + 0;

if hero.x == monster.x {
    set hero.health = hero.health - 5;
    print "Hit!";
}
"""
```

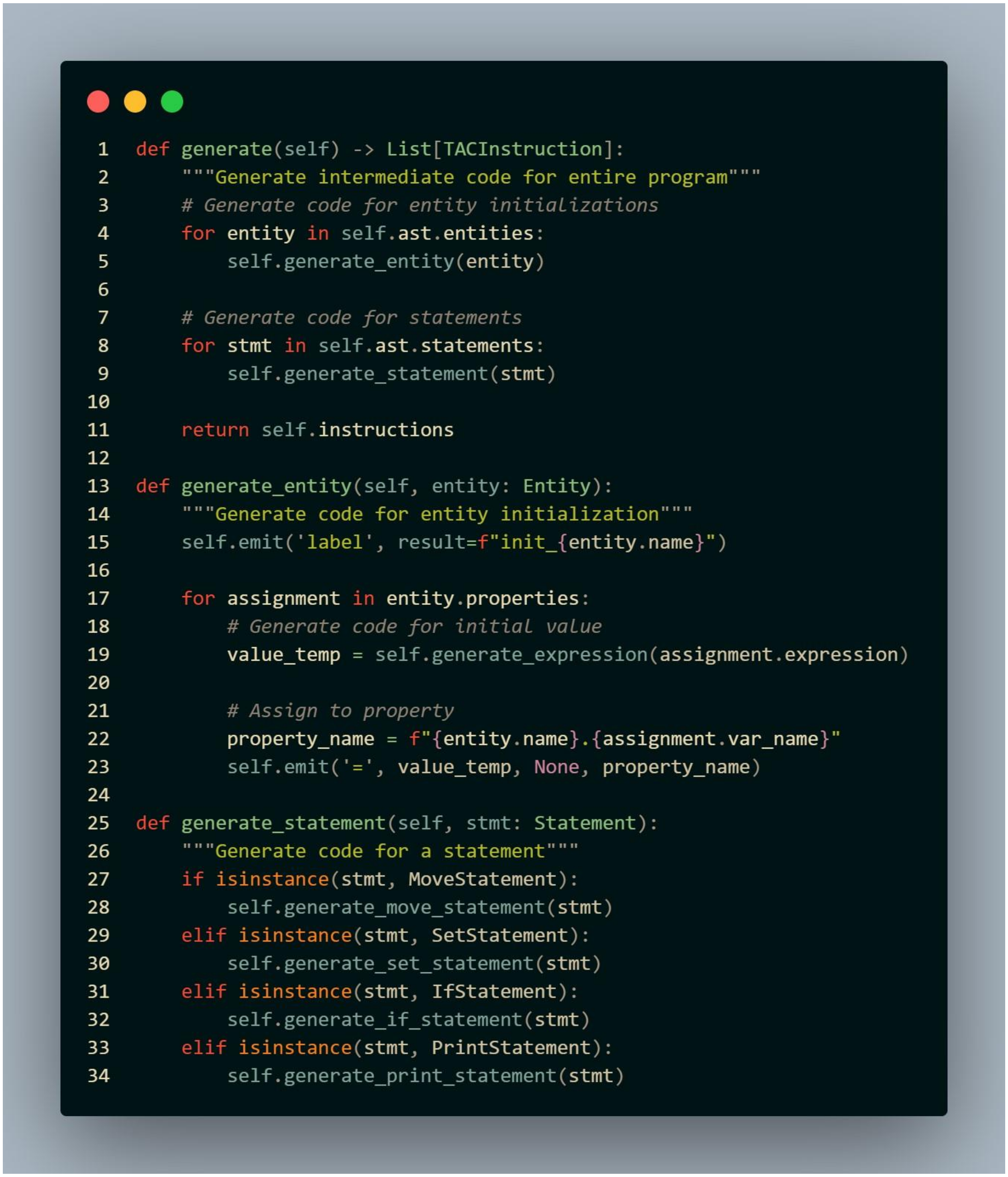
Output



```
OPTIMIZED CODE:
--------------------------------------------
  0  init_hero:
  1      t0 = 8
  2      hero.x = t0
  3      t1 = 10
  4      hero.y = t1
  5      t2 = 100
  6      hero.health = t2
  7      t3 = 0
  8      hero.score = t3
  9  init_monster:
 10      t4 = 4
 11      monster.x = t4
 12      t5 = 8
 13      monster.y = t5
 14      move hero right 3
 15      t6 = hero.x + 3
 16      hero.x = t6
 17      t7 = hero.score
 18      hero.score = t7
 19      t8 = hero.x == monster.x
 20      if t8 goto L0
 21      goto L1
 22  L0:
 23      t9 = hero.health - 5
 24      hero.health = t9
 25      print "Hit!"
 26  L1:

============================================


============================================
OPTIMIZATION SUMMARY
============================================
Original instructions:  27
Optimized instructions: 27
Reduction:              0 instructions
Optimizations applied:  7
============================================
```

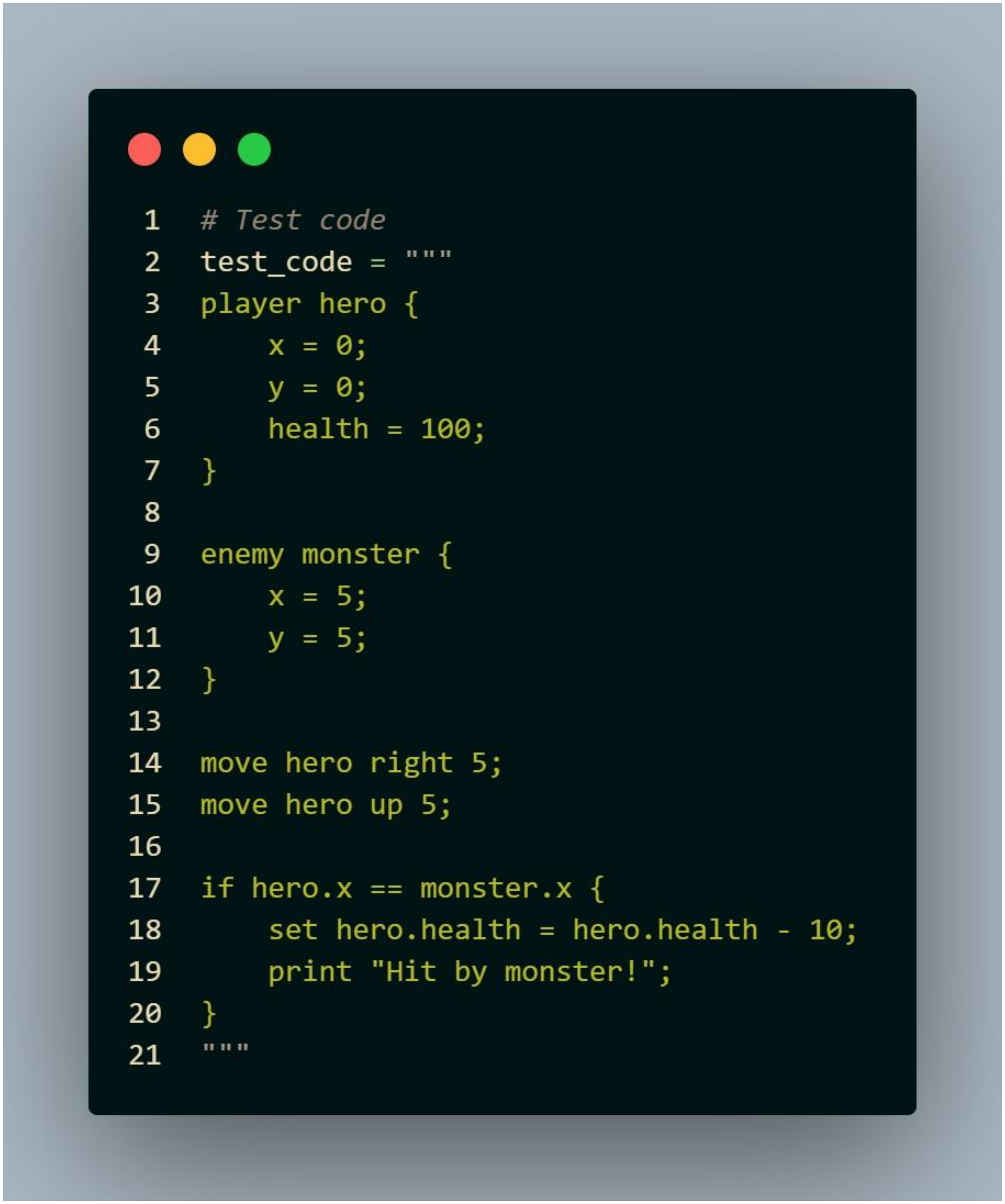
Stage 6 - Code Generation

Functions



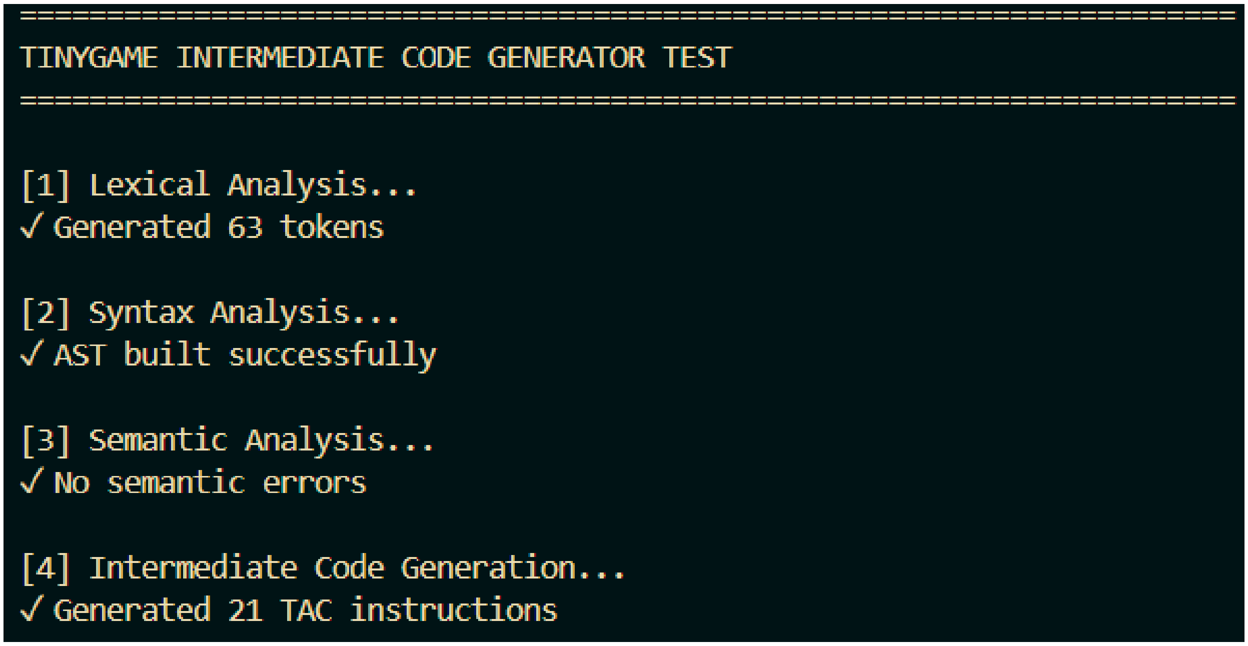
```python
def generate(self) -> List[TACInstruction]:
    """Generate intermediate code for entire program"""
    # Generate code for entity initializations
    for entity in self.ast.entities:
        self.generate_entity(entity)

    # Generate code for statements
    for stmt in self.ast.statements:
        self.generate_statement(stmt)

    return self.instructions

def generate_entity(self, entity: Entity):
    """Generate code for entity initialization"""
    self.emit('label', result=f"init_{entity.name}")

    for assignment in entity.properties:
        # Generate code for initial value
        value_temp = self.generate_expression(assignment.expression)

        # Assign to property
        property_name = f"{entity.name}.{assignment.var_name}"
        self.emit('=', value_temp, None, property_name)

def generate_statement(self, stmt: Statement):
    """Generate code for a statement"""
    if isinstance(stmt, MoveStatement):
        self.generate_move_statement(stmt)
    elif isinstance(stmt, SetStatement):
        self.generate_set_statement(stmt)
    elif isinstance(stmt, IfStatement):
        self.generate_if_statement(stmt)
    elif isinstance(stmt, PrintStatement):
        self.generate_print_statement(stmt)
```

Input

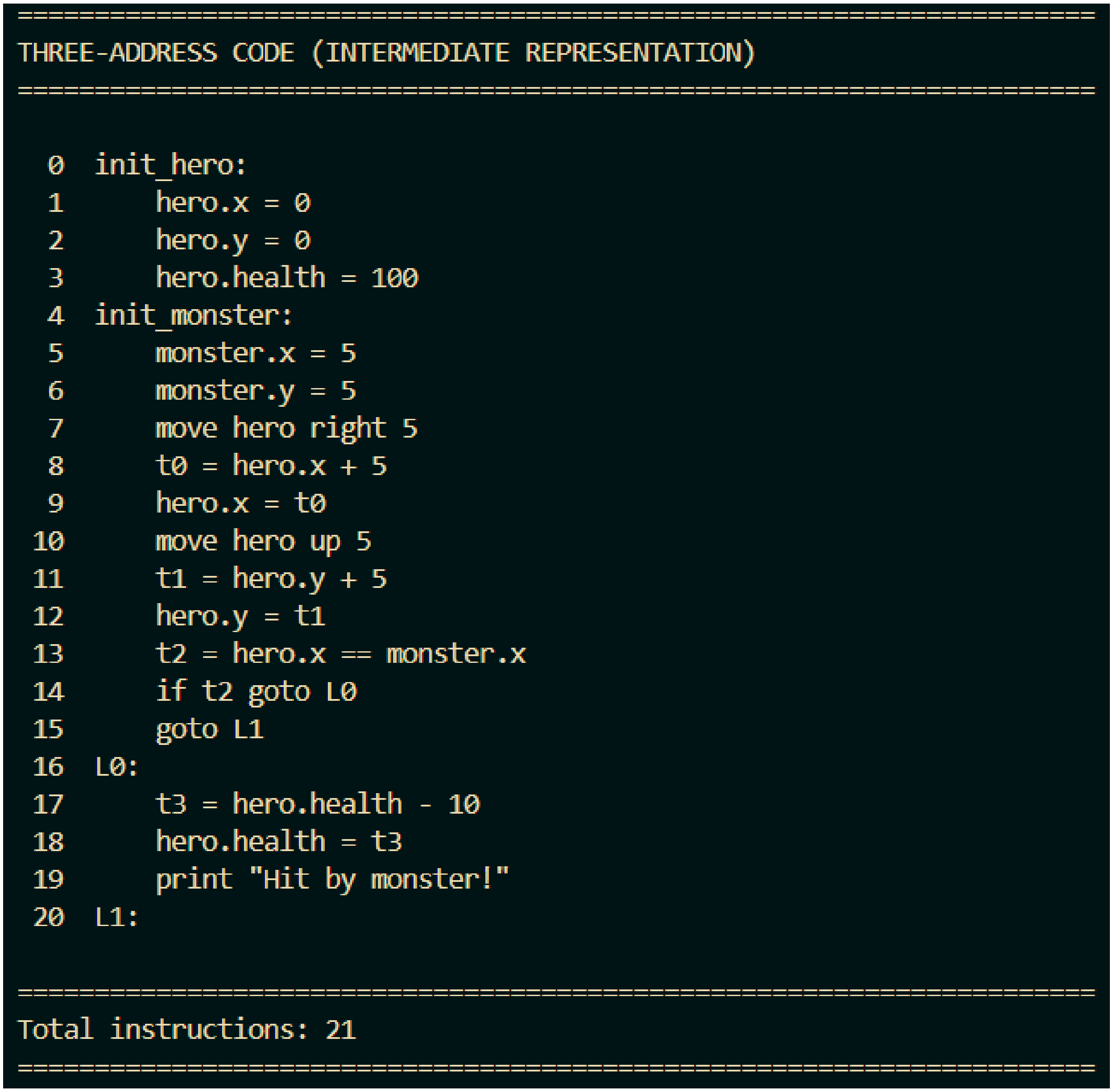


```
# Test code
test_code = """
player hero {
    x = 0;
    y = 0;
    health = 100;
}

enemy monster {
    x = 5;
    y = 5;
}

move hero right 5;
move hero up 5;

if hero.x == monster.x {
    set hero.health = hero.health - 10;
    print "Hit by monster!";
}
"""
```

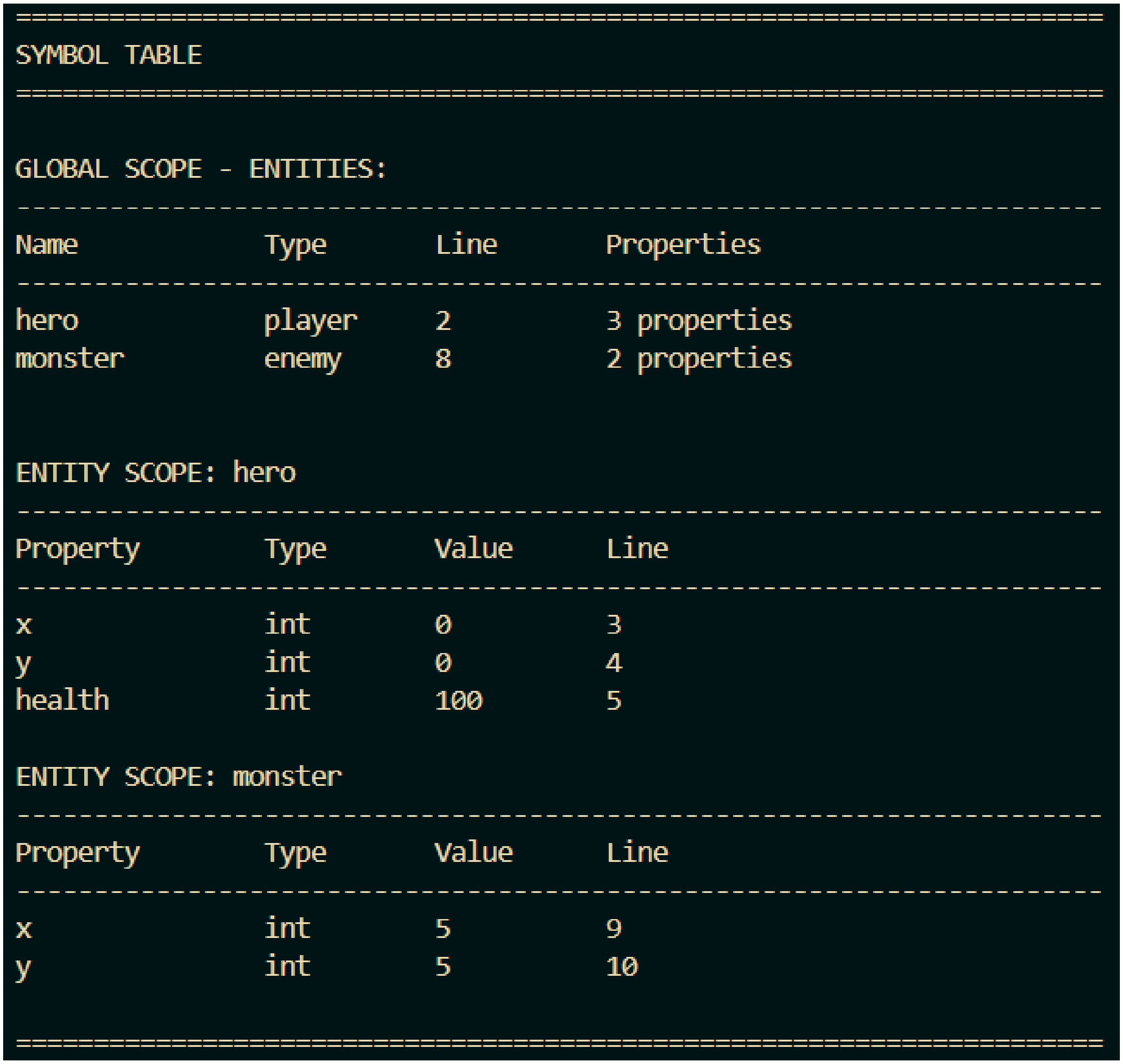
Output



```
======================================================================
TINYGAME INTERMEDIATE CODE GENERATOR TEST
======================================================================

[1] Lexical Analysis...
✓ Generated 63 tokens

[2] Syntax Analysis...
✓ AST built successfully

[3] Semantic Analysis...
✓ No semantic errors

[4] Intermediate Code Generation...
✓ Generated 21 TAC instructions
```



```
======================================================================
THREE-ADDRESS CODE (INTERMEDIATE REPRESENTATION)
======================================================================

  0  init_hero:
  1      hero.x = 0
  2      hero.y = 0
  3      hero.health = 100
  4  init_monster:
  5      monster.x = 5
  6      monster.y = 5
  7      move hero right 5
  8      t0 = hero.x + 5
  9      hero.x = t0
 10      move hero up 5
 11      t1 = hero.y + 5
 12      hero.y = t1
 13      t2 = hero.x == monster.x
 14      if t2 goto L0
 15      goto L1
 16  L0:
 17      t3 = hero.health - 10
 18      hero.health = t3
 19      print "Hit by monster!"
 20  L1:

======================================================================
Total instructions: 21
======================================================================
```



```
======================================================================
SYMBOL TABLE
======================================================================

GLOBAL SCOPE - ENTITIES:
----------------------------------------------------------------------
Name            Type       Line       Properties
----------------------------------------------------------------------
hero            player     2          3 properties
monster         enemy      8          2 properties


ENTITY SCOPE: hero
----------------------------------------------------------------------
Property        Type       Value      Line
----------------------------------------------------------------------
x               int        0          3
y               int        0          4
health          int        100        5

ENTITY SCOPE: monster
----------------------------------------------------------------------
Property        Type       Value      Line
----------------------------------------------------------------------
x               int        5          9
y               int        5          10

======================================================================
```