

### GitHub Repository:

<https://github.com/Anasahmed-10/Compiler-Construction-Project>

### Code Structure



```
1  ## Complete File Structure
2  ...
3  tinygame-compiler/
4  |— main.py           # Main CLI
5  |— lexer.py          # Part 1
6  |— parser.py         # Part 2
7  |— semantic_analyzer.py # Part 3
8  |— code_generator.py  # Part 4
9  |— optimizer.py      # Part 5
10 |— interpreter.py     # Part 6
11 |— examples/
12 |   |— test1.tg
13 |   |— test2.tg
14 |   |— test3.tg
15  ...
```

### UI

TinyGame Compiler v1.0.0  
A Mini Language Compiler for CS4031

```
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
```

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

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

## Token DataClass

```

1  @dataclass
2  class Token:
3      """Represents a single token"""
4      type: TokenType
5      value: any
6      line: int
7      column: int
8
9      def __repr__(self):
10         return f"Token({self.type.name}, {repr(self.value)}, {self.line}:{self.column})"

```

## Functions

```


1  def advance(self):
2      """Move to next character"""
3      if self.current_char == '\n':
4          self.line += 1
5          self.column = 1
6      else:
7          self.column += 1
8
9      self.pos += 1
10     if self.pos >= len(self.source):
11         self.current_char = None
12     else:
13         self.current_char = self.source[self.pos]
14
15 def peek(self, offset: int = 1) -> Optional[str]:
16     """Look ahead at next character(s) without consuming"""
17     peek_pos = self.pos + offset
18     if peek_pos >= len(self.source):
19         return None
20     return self.source[peek_pos]

```



```
1 def tokenize(self) -> List[Token]:
2     """Tokenize entire source code"""
3     tokens = []
4
5     while True:
6         token = self.get_next_token()
7         tokens.append(token)
8
9         if token.type == TokenType.EOF:
10             break
11
12     self.tokens = tokens
13     return tokens
```

## Input



```
1 # Test code
2 test_code = """
3 player hero {
4     x = 0;
5     y = 0;
6     health = 100;
7 }
8
9 enemy monster {
10     x = 5;
11     y = 5;
12 }
13
14 move hero right 5;
15 move hero up 5;
16
17 if hero.x == monster.x {
18     set hero.health = hero.health - 10;
19     print "Hit by monster!";
20 }
21 """
```

## 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

```
1  @dataclass
2  class ASTNode:
3      """Base class for all AST nodes"""
4      line: int
5      column: int
```

### Functions

```
1  def advance(self):
2      """Move to next token"""
3      self.pos += 1
4      if self.pos < len(self.tokens):
5          self.current_token = self.tokens[self.pos]
6      else:
7          self.current_token = None
8
9  def peek(self, offset: int = 1) -> Optional[Token]:
10     """Look ahead at next token(s)"""
11     peek_pos = self.pos + offset
12     if peek_pos < len(self.tokens):
13         return self.tokens[peek_pos]
14     return None
15
16 def expect(self, token_type: TokenType) -> Token:
17     """Consume token of expected type or raise error"""
18     if not self.current_token or self.current_token.type != token_type:
19         self.error(f"Expected {token_type.name}, got {self.current_token.type.name if self.current_token else 'EOF'}")
20     token = self.current_token
21     self.advance()
22     return token
23
24 def match(self, *token_types: TokenType) -> bool:
25     """Check if current token matches any of given types"""
26     if not self.current_token:
27         return False
28     return self.current_token.type in token_types
```

### Input



```
1  # Test code
2  test_code = """
3  player hero {
4      x = 0;
5      y = 0;
6      health = 100;
7  }
8
9  enemy monster {
10     x = 5;
11     y = 5;
12 }
13
14 move hero right 5;
15 move hero up 5;
16
17 if hero.x == monster.x {
18     set hero.health = hero.health - 10;
19     print "Hit by monster!";
20 }
21 """
```

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






```
1  @dataclass
2  class Symbol:
3      """Represents a symbol in the symbol table"""
4      name: str
5      symbol_type: str # 'entity', 'property'
6      data_type: str   # 'int', 'player', 'enemy'
7      value: Optional[int] = None
8      line: int = 0
9      scope: str = "global"
10
11
12  @dataclass
13  class EntitySymbol:
14      """Represents an entity (player or enemy)"""
15      name: str
16      entity_type: str # 'player' or 'enemy'
17      properties: Dict[str, Symbol] = field(default_factory=dict)
18      line: int = 0
19
```

## Functions



```
1 def add_entity(self, name: str, entity_type: str, line: int) -> bool:
2     """Add entity to symbol table"""
3     if name in self.entities:
4         return False
5
6     self.entities[name] = EntitySymbol(name, entity_type, {}, line)
7     return True
8
9 def entity_exists(self, name: str) -> bool:
10    """Check if entity exists"""
11    return name in self.entities
12
13 def add_property(self, entity_name: str, prop_name: str,
14                  value: Optional[int], line: int) -> bool:
15    """Add property to an entity"""
16    if entity_name not in self.entities:
17        return False
18
19    entity = self.entities[entity_name]
20
21    if prop_name in entity.properties:
22        return False
23
24    symbol = Symbol(
25        name=prop_name,
26        symbol_type='property',
27        data_type='int',
28        value=value,
29        line=line,
30        scope=entity_name
31    )
32
33    entity.properties[prop_name] = symbol
34    return True
35
36 def property_exists(self, entity_name: str, prop_name: str) -> bool:
37    """Check if property exists for an entity"""
38    if entity_name not in self.entities:
39        return False
40    return prop_name in self.entities[entity_name].properties
```

Input



```
1  # Test code with errors
2  test_code_2 = """
3  player hero {
4      x = 0;
5      y = 0;
6  }
7
8  // This should cause errors
9  move ghost right 5;
10 set hero.z = 10;
11 """
```

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

```

1  @dataclass
2  class TACInstruction:
3      """Represents a single three-address code instruction"""
4      op: str          # Operation: =, +, -, *, /, ==, !=, >, <, goto, if, label, print, etc.
5      arg1: Optional[str] # First argument
6      arg2: Optional[str] # Second argument
7      result: Optional[str] # Result
8
9      def __str__(self) -> str:
10         """String representation of instruction"""
11         if self.op == 'label':
12             return f"{self.result}:"
13         elif self.op == 'goto':
14             return f"    goto {self.result}"
15         elif self.op == 'if':
16             return f"    if {self.arg1} goto {self.result}"
17         elif self.op == 'ifFalse':
18             return f"    ifFalse {self.arg1} goto {self.result}"
19         elif self.op == 'print':
20             return f"    print {self.arg1}"
21         elif self.op == 'move':
22             # move entity direction amount
23             return f"    move {self.arg1} {self.arg2} {self.result}"
24         elif self.op == '=':
25             if self.arg1 is None:
26                 return f"        {self.result} = (uninitialized)"
27             return f"        {self.result} = {self.arg1}"
28         elif self.op in ['+', '-', '*', '/']:
29             return f"        {self.result} = {self.arg1} {self.op} {self.arg2}"
30         elif self.op in ['==', '!=', '>', '<']:
31             return f"        {self.result} = {self.arg1} {self.op} {self.arg2}"
32         else:
33             return f"        {self.op} {self.arg1} {self.arg2} {self.result}"

```

## Functions



```
1 def new_temp(self) -> str:
2     """Generate new temporary variable"""
3     temp = f"t{self.temp_counter}"
4     self.temp_counter += 1
5     return temp
6
7 def new_label(self) -> str:
8     """Generate new label"""
9     label = f"L{self.label_counter}"
10    self.label_counter += 1
11    return label
12
13 def emit(self, op: str, arg1: Optional[str] = None,
14          arg2: Optional[str] = None, result: Optional[str] = None):
15     """Emit a three-address code instruction"""
16     instruction = TACInstruction(op, arg1, arg2, result)
17     self.instructions.append(instruction)
```

Input



```
1  # Test code
2  test_code = """
3  player hero {
4      x = 0;
5      y = 0;
6      health = 100;
7  }
8
9  enemy monster {
10     x = 5;
11     y = 5;
12 }
13
14 move hero right 5;
15 move hero up 5;
16
17 if hero.x == monster.x {
18     set hero.health = hero.health - 10;
19     print "Hit by monster!";
20 }
21 """
```

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



```

1  def optimize(self) -> List[TACInstruction]:
2      """Perform all optimization passes"""
3      print("\n" + "=" * 70)
4      print("OPTIMIZATION PASSES")
5      print("=" * 70)
6
7      # Start with original instructions
8      current_instructions = deepcopy(self.instructions)
9      original_count = len(current_instructions)
10
11     print(f"\nOriginal instruction count: {original_count}")
12
13     # Pass 1: Constant Folding
14     print("\n[Pass 1] Constant Folding...")
15     current_instructions = self.constant_folding(current_instructions)
16     print(f"  ✓ Completed. {len(self.optimization_log)} optimizations applied")
17
18     # Pass 2: Dead Code Elimination
19     print("\n[Pass 2] Dead Code Elimination...")
20     current_instructions = self.dead_code_elimination(current_instructions)
21     print(f"  ✓ Completed")
22
23     # Pass 3: Copy Propagation
24     print("\n[Pass 3] Copy Propagation...")
25     current_instructions = self.copy_propagation(current_instructions)
26     print(f"  ✓ Completed")
27
28     # Pass 4: Algebraic Simplification
29     print("\n[Pass 4] Algebraic Simplification...")
30     current_instructions = self.algebraic_simplification(current_instructions)
31     print(f"  ✓ Completed")
32
33     final_count = len(current_instructions)
34     reduction = original_count - final_count
35
36     print("\n" + "=" * 70)
37     print(f"Optimization complete!")
38     print(f"Instructions reduced: {original_count} → {final_count} (-{reduction})")
39     print("=" * 70)
40
41     self.optimized_instructions = current_instructions
42     return current_instructions

```

Input



```
1  # Test code with opportunities for optimization
2  test_code = ""
3  player hero {
4      x = 5 + 3;
5      y = 10 * 1;
6      health = 100 - 0;
7      score = 0 * 999;
8  }
9
10 enemy monster {
11     x = 2 + 2;
12     y = 8 / 1;
13 }
14
15 move hero right 3;
16 set hero.score = hero.score + 0;
17
18 if hero.x == monster.x {
19     set hero.health = hero.health - 5;
20     print "Hit!";
21 }
22 ""
```

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



```

1  def generate(self) -> List[TACInstruction]:
2      """Generate intermediate code for entire program"""
3      # Generate code for entity initializations
4      for entity in self.ast.entities:
5          self.generate_entity(entity)
6
7      # Generate code for statements
8      for stmt in self.ast.statements:
9          self.generate_statement(stmt)
10
11     return self.instructions
12
13  def generate_entity(self, entity: Entity):
14      """Generate code for entity initialization"""
15      self.emit('label', result=f"init_{entity.name}")
16
17      for assignment in entity.properties:
18          # Generate code for initial value
19          value_temp = self.generate_expression(assignment.expression)
20
21          # Assign to property
22          property_name = f"{entity.name}.{assignment.var_name}"
23          self.emit('=', value_temp, None, property_name)
24
25  def generate_statement(self, stmt: Statement):
26      """Generate code for a statement"""
27      if isinstance(stmt, MoveStatement):
28          self.generate_move_statement(stmt)
29      elif isinstance(stmt, SetStatement):
30          self.generate_set_statement(stmt)
31      elif isinstance(stmt, IfStatement):
32          self.generate_if_statement(stmt)
33      elif isinstance(stmt, PrintStatement):
34          self.generate_print_statement(stmt)

```

Input



```
1  # Test code
2  test_code = """
3  player hero {
4      x = 0;
5      y = 0;
6      health = 100;
7  }
8
9  enemy monster {
10     x = 5;
11     y = 5;
12 }
13
14 move hero right 5;
15 move hero up 5;
16
17 if hero.x == monster.x {
18     set hero.health = hero.health - 10;
19     print "Hit by monster!";
20 }
21 """
```

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

```
-----
```