Tetris Game Development Project

Final Presentation

Error 404: Name Not Found

Team Members: Anna Dinius, Cody King, Owen Newberry

Project Duration: September 8 – November 23, 2025

Sprints: 2 (Sprint 1: 5 weeks, Sprint 2: 5 weeks)

Project Overview

🎯 Mission Statement

Develop a fully functional Tetris game demonstrating modern software development practices, team collaboration, and agile project management.

📍 Repository

GitHub: https://github.com/jeffreyperdue/ase-420-team-project

Key Numbers - Total Project

📊 Overall Statistics

- **Total Lines of Code:** 6,877
  - Source Code: ~1,200 lines
  - Test Code: ~5,677 lines
- **Total Features:** 14
  - Sprint 1: 8 features
  - Sprint 2: 6 features
- **Total Requirements:** 43
  - Sprint 1: 8 requirements
  - Sprint 2: 35 requirements

- **Total Tests:** 411 test cases
- **Burndown Rate:**
  - Sprint 1: 88.9% (8/9 requirements)
  - Sprint 2: ~100% (35/35 requirements)

Individual Contributions - Anna Dinius

Sprint 1 Achievements

- **Lines of Code:** 1,434
- **Features:** 2
  - Board & Line Clearing System
  - Grid Playing Field
- **Requirements:** 2
  - Grid playing field display
  - Automatic line clearing

Sprint 2 Achievements

- **Features:** 3
  - Scoring System
  - Start Screen
  - Enhanced Game Over Screen
- **Requirements:** 14
  - Scoring logic with multipliers
  - Session high score tracking
  - UI display and persistence
  - Start screen with controls
  - Game over screen with stats

Total Contribution

- **Features:** 5 total
- **Requirements:** 16 total
- **Focus Areas:** Game logic, scoring, UI systems

Individual Contributions - Cody King

Sprint 1 Achievements

- **Lines of Code:** 832
- **Features:** 3
  - Piece Representation
  - Movement & Rotation
  - Collision Detection
- **Requirements:** 3
  - Distinct piece shapes/colors
  - Move and rotate pieces
  - Collision boundaries

Sprint 2 Achievements

- **Features:** 2
  - Next Piece Preview
  - Pause/Resume Functionality
- **Requirements:** 12
  - Preview display area
  - Piece generation integration
  - Pause state management
  - Input handling

Total Contribution

- **Features:** 5 total
- **Requirements:** 15 total
- **Focus Areas:** Game mechanics, user experience

Individual Contributions - Owen Newberry

Sprint 1 Achievements

- **Lines of Code:** 176
- **Features:** 3
  - Board Rendering
  - Keyboard Input Mapping
  - Game Over Overlay
- **Requirements:** 3
  - Keyboard input support
  - Game over window
  - Board and piece rendering

Sprint 2 Achievements

- **Features:** 2
  - Difficulty Levels
  - Ghost Piece Preview
- **Requirements:** 9
  - Level progression system
  - Speed adjustment
  - Landing position calculation

Total Contribution

- **Features:** 5 total
- **Requirements:** 12 total
- **Focus Areas:** Rendering, controls, game progression

The Problem

🎮 Why This Application Matters

Challenge: Build a production-quality Tetris game that demonstrates:

- **Software Engineering Excellence:** Clean architecture, design patterns, SOLID principles
- **Team Collaboration:** Distributed development with clear communication
- **Quality Assurance:** Comprehensive testing at unit, integration, and acceptance levels
- **Agile Methodology:** Sprint planning, burndown tracking, iterative development

How We Solved It

🏗️ Architecture Approach

Layered Architecture with Clear Separation of Concerns:

1. **Game Logic Layer**
   - `Board`: Grid management, line clearing
   - `Piece`: Piece representation and state
   - `Game`: Game state orchestration
   - `Row`: Bitmask-based row representation

2. **View & Input Layer**
   - `PygameRenderer`: Visual rendering
   - `InputHandler`: Keyboard/mouse input mapping
   - `ButtonManager`: UI interaction

3. **Utility Layer**
    - `LinkedList`: Custom data structure
    - `SessionManager`: Session persistence
    - `Score`: Scoring calculations

4. **Integration Layer**
    - `app.py`: Main game loop
    - Intent-based communication between layers

How We Solved It (Continued)

🔄 Development Process

Agile Sprint Methodology:

- **Sprint 1:** MVP foundation (refactoring single source file into OOP-led design)
- **Sprint 2:** Feature enhancement

Quality Assurance:

- **Unit Tests:** 411 test cases covering core logic
- **Integration Tests:** Feature interaction validation
- **Acceptance Tests:** End-to-end user scenarios
- **Regression Tests:** Cross-sprint feature validation

Team Coordination:

- Clear feature ownership
- Regular progress updates
- Regular integration

Technical Implementation - Design Patterns

🎨 Patterns Applied

1. Factory Pattern

```python
# Board uses row_factory for dependency injection
def __init__(self, row_factory, height=HEIGHT, width=WIDTH):
    self._row_factory = row_factory
    # Creates rows without knowing concrete implementation
```

2. Singleton Pattern

```python
# SessionManager ensures single instance
class SessionManager:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

3. Strategy Pattern

```
# Intent-based input handling
intents = input_handler.get_intents(events)
game.apply(intents)  # Game handles different strategies
```

4. Dependency Injection

```
# Game receives dependencies rather than creating them
game = Game(board, spawn_piece, session)
```

Technical Implementation - SOLID Principles

📐 Single Response Principle

Each class has one clear purpose:

- `Board`: Grid management only
- `Piece`: Piece state only
- `Game`: Game orchestration only
- `PygameRenderer`: Rendering only
- `InputHandler`: Input mapping only

Example:

```python
class Board:
    """Encapsulates the playing field grid and related operations"""
    # Only board-related operations
```

Technical Implementation - SOLID Principles

📐 Open/Closed Principle

Open for extension, closed for modification:

- Factory functions allow extension without changing core classes
- Intent system allows new commands without modifying Game class
- Popup system extensible for new screen types

Example:

```python
# Can extend with new row types without modifying Board
board = Board(lambda: CustomRow(WIDTH))
```

📐 Dependency Inversion Principle

High-level modules depend on abstractions:

- `Game` depends on `spawn_piece` function, not concrete Piece creation
- `Board` depends on `row_factory`, not concrete Row class
- `Game` depends on `SessionManager` interface, not implementation

Example:

```python
# Game doesn't know how pieces are created
def spawn_piece():
    return Piece(START_X, START_Y)
game = Game(board, spawn_piece, session)
```

Technical Implementation - SOLID Principles

📐 Interface Segregation Principle

Clean, focused interfaces:

- `Board` exposes only necessary methods (get_cell, set_cell, clear_full_lines)
- `Piece` exposes only position, rotation, type, color
- `InputHandler` returns simple intent strings

Example:

```python
# Clean, minimal interface
class Board:
    def get_cell(self, row, col) -> bool
    def set_cell(self, row, col, color) -> None
    def clear_full_lines(self) -> int
```

📐 Liskov Substitution Principle

Subtypes must be substitutable:

- `Row` implementations can be swapped via factory
- `ButtonManager` can manage different button types
- Intent system allows different input sources

Code Quality Metrics

✅ Quality Indicators

- **Test Coverage:** 411 comprehensive test cases
- **Code Organization:** Clear module separation
- **Documentation:** Comprehensive docstrings
- **Error Handling:** Robust validation and exceptions
- **Maintainability:** DRY principles, reusable components

📈 Test Distribution

- **Unit Tests:** Core logic validation
- **Integration Tests:** Feature interaction
- **Acceptance Tests:** User scenarios
- **Regression Tests:** Cross-sprint validation
- **Performance Tests:** Gameplay smoothness

Architecture Highlights

🏛️ Key Architectural Decisions

1. Intent-Based Communication

- Commands as strings ("LEFT", "ROTATE", "PAUSE")
- Decouples input from game logic
- Easy to test and extend

2. Bitmask Row Representation

- Efficient memory usage
- Fast line-fullness checks
- Color mapping separate from occupancy

3. Custom LinkedList

- Optimized for board operations
- Efficient top insertion (new rows)
- Efficient mid-list deletion (line clearing)

4. State Machine Pattern

- START_SCREEN → PLAYING → GAME_OVER
- Clear state transitions
- State-specific behavior

Sprint Achievements Summary

🎯 Sprint 1 (MVP)

- ✅ Working Tetris game
- ✅ Piece movement and rotation
- ✅ Line clearing
- ✅ Basic rendering
- ✅ Keyboard controls
- **Burndown:** 88.9%

🎯 Sprint 2 (Enhancement)

- ✅ Scoring system with multipliers
- ✅ Next piece preview
- ✅ Pause/resume functionality
- ✅ Difficulty levels
- ✅ Ghost piece preview
- ✅ Enhanced UI screens
- **Burndown:** ~100%

Lessons Learned

✅ What Went Well

1. **Clear Architecture:** Separation of concerns enabled parallel development
2. **Agile Process:** Sprint planning and tracking kept team aligned
3. **Code Quality:** SOLID principles made code maintainable
4. **Team Communication:** Regular updates prevented integration issues

🔄 Areas for Improvement

1. **Documentation:** More architectural documentation would help onboarding

Project Impact

📊 Quantitative Results

- **6,877 lines of production code**
- **411 test cases** ensuring quality
- **14 features** delivered across 2 sprints
- **43 requirements** completed
- **100% Sprint 2 completion**

🎓 Educational Value

- Demonstrated modern software engineering practices
- Showcased team collaboration and agile methodology
- Established quality benchmarks for future projects
- Created reusable architecture patterns

Conclusion

🎉 Project Success

Delivered: A fully functional, well-tested Tetris game demonstrating:

- Clean architecture and design patterns
- SOLID principles application
- Comprehensive testing strategy
- Effective team collaboration
- Agile development methodology

Key Achievement: 6,877 lines of code, 411 tests, 14 features, 43 requirements completed across 2 sprints with high quality standards.

Thank You

Questions?

Team Error 404: Name Not Found

- Anna Dinius - Scoring & UI Systems
- Cody King - Game Mechanics & UX
- Owen Newberry - Rendering & Controls

Repository: https://github.com/jeffreyperdue/ase-420-team-project