Testing Plan

## Chess Game

CS 4230

Traedon Harris

Caden Black

Owen Rasor

Table of Contents

[Scope 3](#_Toc211196888)

[In-Scope 3](#_Toc211196889)

[Out of Scope 3](#_Toc211196890)

[Strategy 4](#_Toc211196891)

[Testing Levels 4](#_Toc211196892)

[Unit Testing 4](#_Toc211196893)

[Testing Types 4](#_Toc211196896)

[Test Design Techniques 5](#_Toc211196901)

[Test Environment 5](#_Toc211196909)

[Test Data Strategy 6](#_Toc211196910)

[Test Coverage Goals 6](#_Toc211196914)

[Resources 7](#_Toc211196915)

[Hardware Resources 7](#_Toc211196916)

[Software Resources 7](#_Toc211196917)

[Test Data 7](#_Toc211196920)

[Documentation 7](#_Toc211196921)

[Timeline 8](#_Toc211196922)

[Risk 10](#_Toc211196927)

# Scope

## In-Scope

* Basic piece movement validation (Pawn, Rook, Knight, Bishop, Queen, King)
* Turn-based gameplay
* Capture mechanics
* Board state management
* Move validation
* Basic win condition
* Game initialization and reset
* Player input handling
* Display of current board state

Out of Scope

* En passant (special pawn capture rule - not required)
* Castling (special King-Rook move - not required)
* Pawn promotion (not required for this project)
* Stalemate detection (not required)
* Check and checkmate validation (may be simplified or omitted)
* Move history and undo functionality
* Timer or clock functionality
* AI opponent (human vs human only)
* Graphical user interface (command-line only)
* Network/multiplayer functionality
* Save/load game state

# Strategy

## Testing Levels

## Unit Testing

* Individual piece movement methods (e.g., Pawn.isValidMove())
* Board cell validation (inBounds checks)
* Coordinate parsing functions
* Turn management logic
* Capture validation methods

### Integration Testing

* Piece objects interacting with Board object
* Game controller coordinating Player, Board, and Piece objects
* Input parser connecting user input to game logic
* Display system rendering current board state

### System Testing

* Complete game playthrough scenarios
* Multiple consecutive games
* Various opening sequences
* Edge case scenarios (corner moves, board edges)

Testing Types

### Functional Testing

* Each piece type moves according to chess rules
* Pieces cannot move off the board
* Pieces cannot move through other pieces (except Knight)
* Players alternate turns correctly
* Captures remove pieces from board

### Negative Testing

* Invalid coordinate inputs (out of range, malformed)
* Moving opponent's pieces
* Moving to occupied square (same color)
* Moving pieces in invalid patterns
* Playing out of turn
* Moving from empty squares

### Boundary Testing

* Moves to board edges (a1, h8, etc.)
* Maximum movement ranges (Queen across full board)
* Minimum movements (King one square)
* First and last moves of the game

### Regression Testing

* Re-run all tests after bug fixes
* Verify fixes don't break existing functionality

## Test Design Techniques

### Equivalence Partitioning

* Valid board coordinates (a1-h8) vs invalid coordinates
* Own pieces vs opponent pieces vs empty squares
* Legal move destinations vs illegal destinations

### Boundary Value Analysis

* Edge squares: a1, a8, h1, h8
* Corner cases for each piece type
* First move vs subsequent moves (especially for Pawns)

Decision Table Testing

* Move validation logic (piece type + from + to + board state = valid/invalid)
* Capture scenarios (attacker + defender + position = captured/not captured)

### State Transition Testing

* Game states: Setup -> White's Turn -> Black's Turn -> Game Over
* Piece states: On board -> Captured -> Off board

### Error Guessing

* Null pointer exceptions (empty squares)
* Off-by-one errors in coordinate conversion
* Case sensitivity in input (e.g., "A1" vs "a1")
* Whitespace in input strings

### Entry Criteria

* All source code compiles without errors
* Executables generated and tested for basic launch
* Test environment set up on all team machines
* All test cases written and peer-reviewed
* Test data prepared (sample board states, move sequences)

### Exit Criteria

* 100% of planned test cases executed
* 90% pass rate achieved (allowing for known limitations)
* All Critical and High priority defects resolved or documented
* Medium/Low priority defects documented for future work
* Test results exported to Excel file
* Test summary report completed

## Test Environment

* Programming Language: Python 3.7+
* Testing Framework: pytest
* Build Tools: pip
* Command Line: PowerShell

Test Data Strategy

### Valid Test Data

* Standard starting board configuration
* Mid-game board states (5-10 moves in)
* End-game scenarios (few pieces remaining)
* Common opening moves (e4, d4, Nf3, etc.)

### Invalid Test Data

* Malformed coordinates: "i9", "z1", "a0", "99"
* Non-alphanumeric input: "@#$%", empty strings
* Wrong format: "1a" instead of "a1"
* Out of turn moves
* Impossible piece movements

### Edge Case Data

* All pieces at board edges
* Single piece remaining scenarios
* Maximum capture scenarios (one piece takes multiple over game)

Test Coverage Goals

* Code Coverage: 75% minimum
* Requirement Coverage: 100% of in-scope requirements
* Piece Types: 100% (all 6 piece types tested)
* Move Types: 100% (normal moves, captures, invalid moves)
* Board Coverage: All 64 squares involved in at least one test

# Resources

Hardware Resources

**Test laptop (minimum specs):**

* 8GB RAM
* Dual-core processor
* 20GB available storage
* Multiple OS support for cross-platform testing

## Software Resources

**Programming Environment:**

* Python 3.9+
* pytest
* pip

### Development Tools

* IDE
* Version Control: Git
* Repository: GitHub

### Testing Tools

* Openpyxl
* Microsoft Excel
* Markdown for README

## Test Data

* Standard chess starting position (FEN notation or manual setup)
* 10-15 pre-configured board states for specific scenarios
* Collection of valid and invalid move commands
* Expected output files for comparison

## Documentation

* Source code with inline comments
* How to execute game and tests
* Running individual tests
* Test case templates
* Exporting Excel file

# Timeline

## Week 1

**Overview:** Team communication and chess game implementation

**TODO:**

* Initial team formation and role assignment
* Requirements analysis and specification review
* Source code development (main.py, board.py, pieces.py, etc.)
* Individual module implementation and unit debugging
* Code integration and initial testing of core functionality
* Version control setup and code sharing

**Key Milestones:**

* Chess game source code complete and functional
* Working executables that can play a basic chess game

## Week 2

**Overview:** Team coordination, test planning, and initial test case development

**TODO:**

* Full team meeting: Code walkthrough and demonstration
* Clarify ambiguities in requirements vs implementation
* Distribute test planning roles and responsibilities
* Begin writing test cases

**Key Milestones:**

* Test plan 80% complete, 30-40 test cases written
* Draft test plan, initial test case suite

## Week 3

**Overview:** Complete testing, build export module, finalize deliverables

**TODO:**

Integrate all sections

Finish writing remaining test cases

Implement Excel export module with all required fields

Integrate test runner with Excel export functionality

**Key Milestones:**

* Initial test execution complete
* Final test execution with Excel generated
* All deliverables finalized and submitted

## Risk-Adjusted Timeline

**Optimistic:**

* Complete test execution 2 days before due date
* Extra day for additional test coverage

**Realistic:**

* Test execution completes 1 day before due date
* Review and submit on due date

**Pessimistic:**

* Test execution extends into due date
* Rushed documentation
* Submit just before deadline
* **Activate contingency:** reduced scope, manual processes

# Risk

Risk 1

Ambiguous or Missing Chess Rules in Implementation

* **Probability:** High
* **Impact:** High
* **Description:** Source code may not implement all standard chess rules, making it unclear what to test or what "correct" behavior is
* **Mitigation:** 
  + Early code review session with whole team
  + Document all implemented vs non-implemented rules in Scope
  + Clarify with instructor if needed
* **Contingency:**
  + Test only what is actually implemented
  + Document assumptions in "Specification Clarifications" section
  + Focus on what code does rather than full chess rules

## Risk 2

Team Member Unavailability

* **Probability:** Medium
* **Impact:** High
* **Description:** Teammate becomes sick, has emergency, or has conflicting deadline
* **Mitigation:**
  + Maintain detailed documentation
  + Use version control for all work
  + Regular check-ins to catch issues early
* **Contingency:**
  + Redistribute tasks among remaining members
  + Activate 2-day buffer time
  + Prioritize critical deliverables over nice-to-haves

Risk 3

Bugs in Source Code Prevent Testing

* **Probability:** Medium
* **Impact:** High
* **Description:** Game crashes immediately, has infinite loops, or major functionality broken
* **Mitigation:**
  + Identify critical bugs immediately
  + Fix showstopper bugs before writing tests
* **Contingency:**
  + Document bugs that prevent testing in addenda
  + Test around broken functionality
  + Focus on modules that do work

Risk 4

Test Framework Implementation Takes Longer Than Expected

* **Probability:** Medium
* **Impact:** Medium
* **Description:** Setting up Excel spreadsheet generation and test automation proves complex
* **Mitigation:**
  + Start with simple manual tests first
  + Use existing libraries where possible
* **Contingency:**
  + Execute tests manually if automation fails
  + Reduce automation scope, focus on core functionality

Risk 5

Insufficient Test Coverage

* **Probability:** Low
* **Impact:** Medium
* Description: Not enough test cases to adequately cover functionality
* **Mitigation:**
  + Create requirements traceability matrix early
  + Use systematic test design techniques (equivalence partitioning, boundary analysis)
  + Peer review test cases for gaps
  + Track coverage metrics
* **Contingency:**
  + Prioritize high-risk areas (complex pieces like Queen, Knight)
  + Document known gaps in test coverage
  + Explain prioritization rationale in test plan

Risk 6

Misunderstanding Assignment Requirements

* **Probability:** Low
* **Impact:** High
* **Description:** Deliverables don't match what instructor expects
* **Mitigation:**
  + Review assignment rubric carefully
  + Ask clarifying questions early
* **Contingency:**
  + Pivot deliverables if caught early enough