

Monaco Editor Comprehensive TODO

Project: Heaven GUI for Solo-Git

Purpose: Complete integration roadmap for Monaco Editor with Solo-Git unique capabilities

Last Updated: 2025-10-20

Status: Phase 1 - Planning

Table of Contents

1. [Current State Analysis](#)
 2. [Solo-Git Integration Opportunities](#)
 3. [“No UI” Philosophy Alignment](#)
 4. [Keyboard-First Interaction](#)
 5. [Git Integration](#)
 6. [AI-Assisted Editing](#)
 7. [Test-Driven Workflow](#)
 8. [Performance Optimization](#)
 9. [Accessibility](#)
 10. [Implementation Roadmap](#)
-

1. Current State Analysis

What's Already Good

Theme Integration

- Custom Heaven theme properly defined (`monaco-theme.ts`)
- Dark color scheme with accent colors (`#82AAFF`, `#C792EA`, `#C3E88D`, etc.)
- Syntax highlighting for all major languages
- Proper token colorization

Core Features

- Basic code editing with Monaco
- Keyboard shortcuts (`Cmd+S`, `Shift+Alt+F`, `Cmd+/*`)
- Context menu integration
- Auto-save functionality (2s delay)
- Read-only mode support
- Git file status display in header
- Format document support
- Minimap and word wrap toggles

Editor Options

- JetBrains Mono font with ligatures
- Proper indentation (2 spaces)
- Bracket matching

- Cursor animations
- TypeScript/JavaScript defaults configured

UI Components

- EditorHeader with status, language, git status
- EditorLoadingState
- EditorContextMenu
- Clean separation of concerns

What Needs Improvement ⚠️

“No UI” Philosophy Violations

- Header always visible (should be contextual/hover-to-reveal)
- Minimap at 100% opacity (should be 30% faded)
- Git status always shown (should be contextual)
- Breadcrumbs missing (should show on hover)
- Status indicators too persistent

Solo-Git Integration Gaps

- No workpad status indicators
- No AI-assisted code markers
- No inline test results
- No git diff visualization for workpads
- No cost tracking for AI operations
- No auto-merge status display
- No CI/CD pipeline status

Performance Concerns

- No lazy loading for large files
- No virtual scrolling
- No debouncing for expensive operations
- No memoization optimizations
- No Web Worker usage

Accessibility Gaps

- Limited screen reader support
- No high contrast mode
- Missing ARIA labels
- Focus management needs improvement

Git Workflow

- No inline git blame
- No conflict resolution UI
- No commit staging from editor
- No workpad diff visualization

AI Workflow

- No inline AI suggestions
 - No code generation markers
 - No confidence scores
 - No model tier indicators
 - No cost tracking per edit
-

2. Solo-Git Integration Opportunities

Based on Solo-Git's unique capabilities, Monaco Editor can be deeply integrated with:

2.1 Workpad Status Integration

Current State: No workpad awareness in editor

Goal: Real-time workpad status visualization

Implementation Ideas:

- Workpad indicator in gutter (dotted line for workpad, solid for trunk)
- Inline badges showing workpad type (ephemeral/trunk)
- Workpad diff visualization (side-by-side comparison)
- Auto-merge countdown timer when tests are running
- "Ready to Promote" banner when tests pass

Technical Approach:

```
// Add workpad decoration provider
const workpadDecorations = editor.createDecorationsCollection([
  {
    range: new monaco.Range(1, 1, 1, 1),
    options: {
      isWholeLine: true,
      className: 'workpad-line',
      glyphMarginClassName: 'workpad-glyph',
      glyphMarginHoverMessage: { value: 'Workpad: feature/ai-commit' }
    }
  }
])
```

2.2 AI-Assisted Code Markers

Current State: No AI assistance visualization

Goal: Visual indicators for AI-generated/suggested code

Implementation Ideas:

- Sparkle icon (✨) in gutter for AI-generated lines
- Confidence score on hover (80% confident)
- Model tier badge (GPT-4 / DeepSeek / Llama)
- Cost indicator per edit (\$0.002)
- "Accept/Reject" inline actions for suggestions
- Diff view showing AI changes vs original

Technical Approach:

```
// AI suggestion decoration
const aiDecorations = editor.createDecorationsCollection([
  {
    range: new monaco.Range(10, 1, 15, 1),
    options: {
      isWholeLine: true,
      className: 'ai-generated-code',
      glyphMarginClassName: 'ai-sparkle-icon',
      hoverMessage: {
        value: '✨ AI-generated (GPT-4)\nConfidence: 92%\nCost: $0.003'
      },
      afterContentClassName: 'ai-accept-reject-actions'
    }
  }
])
```

2.3 Inline Test Results

Current State: Tests shown in separate panel

Goal: Test results directly in editor

Implementation Ideas:

- Green checkmark in gutter for passing test lines
- Red X for failing test lines
- Yellow dot for pending tests
- Click gutter icon to run specific test
- Hover to see test output/error
- Coverage percentage per function
- “Test this function” code lens

Technical Approach:

```
// Test result decorations
const testDecorations = editor.createDecorationsCollection([
  {
    range: new monaco.Range(20, 1, 20, 1),
    options: {
      glyphMarginClassName: 'test-passed',
      glyphMarginHoverMessage: {
        value: '✓ Test passed (42ms)\nunit/auth.test.ts - line 120'
      }
    }
  },
  {
    range: new monaco.Range(30, 1, 30, 1),
    options: {
      glyphMarginClassName: 'test-failed',
      glyphMarginHoverMessage: {
        value: '✗ Test failed\nExpected: true\nReceived: false'
      }
    }
  }
])

// Code lens for running tests
monaco.languages.registerCodeLensProvider('typescript', {
  provideCodeLenses: (model, token) => {
    return {
      lenses: [
        {
          range: new monaco.Range(15, 1, 15, 1),
          command: {
            id: 'run-test',
            title: '▶ Run Test'
          }
        }
      ]
    }
  }
})
```

2.4 Git Diff for Workpads

Current State: No inline diff

Goal: Visual diff between workpad and trunk

Implementation Ideas:

- Gutter indicators for added/modified/deleted lines
- Inline diff view (side-by-side or unified)
- “View in Trunk” action to see original
- Conflict resolution UI
- “Accept/Reject” inline actions for changes
- Workpad promotion preview

Technical Approach:

```
// Diff decorations
const diffDecorations = editor.createDecorationsCollection([
  {
    range: new monaco.Range(10, 1, 10, 1),
    options: {
      isWholeLine: true,
      linesDecorationsClassName: 'line-added',
      className: 'line-added-bg',
      glyphMarginClassName: 'added-glyph'
    }
  },
  {
    range: new monaco.Range(20, 1, 20, 1),
    options: {
      isWholeLine: true,
      linesDecorationsClassName: 'line-modified',
      className: 'line-modified-bg',
      glyphMarginClassName: 'modified-glyph'
    }
  }
])

// Use Monaco's built-in diff editor
const diffEditor = monaco.editor.createDiffEditor(container, {
  enableSplitViewResizing: true,
  renderSideBySide: true,
  theme: 'heaven'
})

diffEditor.setModel({
  original: monaco.editor.createModel(trunkContent, 'typescript'),
  modified: monaco.editor.createModel(workpadContent, 'typescript')
})
```

2.5 Cost Tracking Integration

Current State: Cost shown in status bar only

Goal: Per-edit cost tracking in editor

Implementation Ideas:

- Inline cost badges for AI operations
- Cumulative cost counter
- Budget warning overlays
- “Cost Summary” code lens
- Model switching based on budget
- Cost breakdown on hover

Technical Approach:

```
// Cost tracking overlay
const costOverlay = editor.createOverlayWidget({
  getId: () => 'cost-tracker',
  getDomNode: () => {
    const node = document.createElement('div')
    node.innerHTML = '💰 $0.15 / $10.00 daily'
    node.className = 'cost-tracker-widget'
    return node
  },
  getPosition: () => ({
    preference: monaco.editor.OverlayWidgetPositionPreference.TOP_RIGHT_CORNER
  })
})

editor.addOverlayWidget(costOverlay)
```

2.6 CI/CD Pipeline Status

Current State: No CI status in editor

Goal: Real-time build/deploy status

Implementation Ideas:

- Pipeline status badge in editor
- Build progress indicator
- Failed build error markers
- “View Build Logs” action
- Smoke test results inline
- Deploy status indicator

Technical Approach:

```
// Pipeline status widget
const pipelineWidget = editor.createContentWidget({
  getId: () => 'pipeline-status',
  getDomNode: () => {
    const node = document.createElement('div')
    node.innerHTML = `
      <div class="pipeline-widget">
        🚀 Build #384
        <span class="status success">✓ SUCCESS</span>
      </div>
    `
    return node
  },
  getPosition: () => ({
    position: { lineNumber: 1, column: 1 },
    preference: [monaco.editor.ContentWidgetPositionPreference.ABOVE]
  })
})

editor.addContentWidget(pipelineWidget)
```

3. “No UI” Philosophy Alignment

Following Heaven’s “as little design as possible” principle:

3.1 Contextual Header

Current: Always visible

Target: Show only on hover or interaction

Implementation:

```
// Auto-hide header after 3 seconds of inactivity
const { isVisible, triggerActivity } = useContextualVisibility(3000)

<EditorHeader
  className={cn(
    'transition-opacity duration-300',
    isVisible ? 'opacity-100' : 'opacity-0'
  )}
  onMouseEnter={triggerActivity}
/>
```

3.2 Faded Minimap

Current: 100% opacity

Target: 30% opacity, full on hover

Implementation:

```
// Custom CSS for faded minimap
.monaco-editor .minimap {
  opacity: 0.3;
  transition: opacity 150ms ease-in-out;
}

.monaco-editor:hover .minimap,
.monaco-editor .minimap:hover {
  opacity: 1;
}
```

3.3 Contextual Breadcrumb

Current: Not implemented

Target: Show on hover only

Implementation:

```
// Breadcrumb overlay widget
const breadcrumbWidget = editor.createOverlayWidget({
  getId: () => 'breadcrumb',
  getDomNode: () => {
    const node = document.createElement('div')
    node.innerHTML = 'src / components / web / CodeEditor.tsx'
    node.className = 'breadcrumb-widget opacity-0 hover:opacity-100'
    return node
  },
  getPosition: () => ({
    preference: monaco.editor.OverlayWidgetPositionPreference.TOP_CENTER
  })
})
```


3.4 Hide Tabs When Single File

Current: Not implemented

Target: Only show tabs when multiple files open

Implementation:

```
// Conditional tab bar
{openFiles.length > 1 && (
  <EditorTabs
    files={openFiles}
    activeFile={activeFile}
    onSelect={onFileSelect}
  />
)}
```

3.5 Contextual Git Status

Current: Always visible in header

Target: Show only on hover or when there are changes

Implementation:

```
// Contextual git status
{(hoveredHeader || gitStatus?.hasChanges) && (
  <GitStatusBadge
    status={gitStatus}
    className="transition-opacity duration-150"
  />
)}
```

4. Keyboard-First Interaction

All Monaco Editor operations should be keyboard-accessible:

4.1 Current Keyboard Shortcuts

✅ Already Implemented:

- Cmd/Ctrl+S - Save
- Shift+Alt+F - Format document
- Cmd/Ctrl+/ - Toggle comment
- Cmd/Ctrl+F - Find
- Cmd/Ctrl+H - Replace

4.2 Missing Solo-Git Shortcuts

⚠️ Need to Add:

- Cmd+Shift+A - Open AI Commit Assistant
- Cmd+Shift+T - Run tests for current file
- Cmd+Shift+W - Create workpad
- Cmd+Shift+P - Promote workpad to trunk
- Cmd+Shift+D - Show diff with trunk
- Cmd+Shift+G - Show git blame

- Cmd+Shift+B - Open pipeline view
- Cmd+K Z - Toggle zen mode
- Cmd+K Cmd+T - Run test at cursor
- Cmd+K Cmd+C - Show AI cost breakdown

Implementation:

```
// Register Solo-Git keyboard shortcuts
editor.addCommand(
  monaco.KeyMod.CtrlCmd | monaco.KeyMod.Shift | monaco.KeyCode.KeyA,
  () => openAICommitAssistant()
)

editor.addCommand(
  monaco.KeyMod.CtrlCmd | monaco.KeyMod.Shift | monaco.KeyCode.KeyT,
  () => runTestsForCurrentFile()
)

editor.addCommand(
  monaco.KeyMod.CtrlCmd | monaco.KeyMod.Shift | monaco.KeyCode.KeyW,
  () => createWorkpad()
)

// ... add all shortcuts
```

4.3 Command Palette Integration

Current: Separate CommandPalette component

Target: Integrated with Monaco's command palette

Implementation:

```
// Register custom actions in Monaco
editor.addAction({
  id: 'solo-git.create-workpad',
  label: 'Workpad: Create',
  keybindings: [
    monaco.KeyMod.CtrlCmd | monaco.KeyMod.Shift | monaco.KeyCode.KeyW
  ],
  contextMenuGroupId: 'solo-git',
  contextMenuOrder: 1,
  run: () => createWorkpad()
})

editor.addAction({
  id: 'solo-git.run-tests',
  label: 'Tests: Run Current File',
  keybindings: [
    monaco.KeyMod.CtrlCmd | monaco.KeyMod.Shift | monaco.KeyCode.KeyT
  ],
  contextMenuGroupId: 'solo-git',
  contextMenuOrder: 2,
  run: () => runTests()
})
```

4.4 Keyboard Navigation Enhancements

Need to Add:

- `Cmd+1-9` - Switch between open files
 - `Cmd+W` - Close current file
 - `Cmd+Shift+[/ Cmd+Shift+] - Navigate between workpad/trunk`
 - `Cmd+Alt+Up/Down` - Navigate between test results
 - `Esc` - Dismiss all overlays/widgets
-

5. Git Integration

Deep git integration for Solo-Git workflow:

5.1 Inline Git Blame

Current: Not implemented

Goal: Show commit info for each line

Implementation:

```
// Git blame decorations
const blameDecorations = editor.createDecorationsCollection(
  lines.map(line => ({
    range: new monaco.Range(line.number, 1, line.number, 1),
    options: {
      isWholeLine: false,
      afterContentClassName: 'git-blame-info',
      after: {
        content: `${line.author} - ${line.message} - ${line.timeAgo}`,
        inlineClassName: 'git-blame-text opacity-0 hover:opacity-100'
      }
    }
  }))
)
```

5.2 Workpad Diff Visualization

Current: Not implemented

Goal: Side-by-side workpad vs trunk comparison

Implementation:

```
// Toggle diff view
const toggleDiffView = () => {
  if (isDiffMode) {
    // Switch back to single editor
    editor.setModel(currentModel)
  } else {
    // Create diff editor
    const diffEditor = monaco.editor.createDiffEditor(container, {
      enableSplitViewResizing: true,
      renderSideBySide: true
    })

    diffEditor.setModel({
      original: trunkModel,
      modified: workpadModel
    })
  }
}
```

5.3 Conflict Resolution UI

Current: Not implemented

Goal: Visual conflict resolution

Implementation:

```
// Conflict decorations
const conflictDecorations = editor.createDecorationsCollection([
  {
    range: new monaco.Range(10, 1, 20, 1),
    options: {
      isWholeLine: true,
      className: 'merge-conflict-region',
      glyphMarginClassName: 'merge-conflict-glyph',
      linesDecorationsClassName: 'merge-conflict-decoration'
    }
  }
])

// Conflict resolution actions
const createConflictActions = (range: monaco.Range) => [
  {
    label: 'Accept Current',
    run: () => acceptCurrent(range)
  },
  {
    label: 'Accept Incoming',
    run: () => acceptIncoming(range)
  },
  {
    label: 'Accept Both',
    run: () => acceptBoth(range)
  },
  {
    label: 'AI Resolve',
    run: () => aiResolveConflict(range)
  }
]
```

5.4 Commit Staging from Editor

Current: Not implemented

Goal: Stage/unstage chunks directly in editor

Implementation:

```
// Chunk staging actions
editor.addAction({
  id: 'git.stage-chunk',
  label: 'Git: Stage This Change',
  contextMenuGroupId: 'git',
  precondition: 'editorHasSelection',
  run: (ed) => {
    const selection = ed.getSelection()
    if (selection) {
      stageChunk(selection)
    }
  }
})

// Visual feedback for staged chunks
const stagedDecorations = editor.createDecorationsCollection([
  {
    range: stagedRange,
    options: {
      className: 'staged-chunk',
      glyphMarginClassName: 'staged-glyph'
    }
  }
])
```

5.5 AI-Powered Conflict Resolution

Current: Not implemented

Goal: Let AI resolve merge conflicts

Implementation:

```
// AI conflict resolution
const aiResolveConflict = async (conflictRange: monaco.Range) => {
  const currentCode = editor.getModel()?.getValueInRange(conflictRange)

  // Call Tauri backend
  const resolution = await invoke('ai_resolve_conflict', {
    current: currentCode,
    incoming: incomingCode,
    context: fileContext
  })

  // Apply resolution
  editor.executeEdits('ai-resolve', [{
    range: conflictRange,
    text: resolution.code
  }])

  // Show confidence score
  showNotification(`AI resolved conflict (${resolution.confidence}% confident)`)
}
```

6. AI-Assisted Editing

Leverage Solo-Git's multi-model AI capabilities:

6.1 Inline AI Suggestions

Current: Not implemented

Goal: Real-time AI code suggestions

Implementation:

```
// Register completion provider
monaco.languages.registerCompletionItemProvider('typescript', {
  provideCompletionItems: async (model, position, context, token) => {
    const textUntilPosition = model.getValueInRange({
      startLineNumber: 1,
      startColumn: 1,
      endLineNumber: position.lineNumber,
      endColumn: position.column
    })

    // Call AI for suggestions
    const suggestions = await invoke('get_ai_suggestions', {
      code: textUntilPosition,
      language: 'typescript'
    })

    return {
      suggestions: suggestions.map(s => ({
        label: s.label,
        kind: monaco.languages.CompletionItemKind.Snippet,
        insertText: s.code,
        documentation: `✨ AI-generated (${s.model})\nConfidence: ${s.confidence}%
\nCost: ${s.cost}`,
        detail: s.description
      }))
    }
  }
})
```

6.2 Code Generation Markers

Current: Not implemented

Goal: Visual markers for AI-generated code

Implementation:

```

// Track AI-generated ranges
const aiGeneratedRanges = new Map<string, {
  range: monaco.Range
  model: string
  confidence: number
  cost: number
  timestamp: Date
}>()

// Add decorations for AI code
const aiCodeDecorations = editor.createDecorationsCollection(
  Array.from(aiGeneratedRanges.values()).map(ai => ({
    range: ai.range,
    options: {
      isWholeLine: true,
      className: 'ai-generated-line',
      glyphMarginClassName: 'ai-sparkle-icon',
      hoverMessage: {
        value: `✨ AI-generated by ${ai.model}\n` +
          `Confidence: ${ai.confidence}%\n` +
          `Cost: ${ai.cost.toFixed(4)}\n` +
          `${formatTimeAgo(ai.timestamp)}`
      }
    }
  }))
)

```

6.3 Confidence Scores

Current: Not implemented

Goal: Show AI confidence for suggestions

Implementation:

```
// Confidence badge overlay
const showConfidenceBadge = (range: monaco.Range, confidence: number) => {
  const widget = editor.createContentWidget({
    getId: () => `confidence-${range.startLineNumber}`,
    getDomNode: () => {
      const node = document.createElement('div')
      node.className = cn(
        'confidence-badge',
        confidence >= 80 && 'confidence-high',
        confidence >= 60 && confidence < 80 && 'confidence-medium',
        confidence < 60 && 'confidence-low'
      )
      node.textContent = `${confidence}%`
      return node
    },
    getPosition: () => ({
      position: {
        lineNumber: range.startLineNumber,
        column: range.endColumn
      },
      preference: [monaco.editor.ContentWidgetPositionPreference.EXACT]
    })
  })

  editor.addContentWidget(widget)
}
```

6.4 Model Tier Indicators

Current: Not implemented

Goal: Show which AI model was used

Implementation:

```
// Model badge
const modelBadge = {
  'gpt-4': { icon: '🧠', color: '#82A AFF', label: 'GPT-4' },
  'deepseek': { icon: '🌊', color: '#C792EA', label: 'DeepSeek' },
  'llama': { icon: '🐫', color: '#C3E88D', label: 'Llama' }
}

// Add model indicator to gutter
const addModelIndicator = (line: number, model: string) => {
  const badge = modelBadge[model]
  editor.changeViewZones((changeAccessor) => {
    changeAccessor.addZone({
      afterLineNumber: line,
      heightInPx: 20,
      domNode: createModelBadge(badge)
    })
  })
}
```

6.5 Cost Tracking Per Edit

Current: Global cost tracking only

Goal: Per-edit cost breakdown

Implementation:


```
// Track edit costs
const editCosts = new Map<string, number>()

// Show cost on hover
editor.onMouseMove((e) => {
  if (e.target.type === monaco.editor.MouseTargetType.CONTENT_TEXT) {
    const line = e.target.position?.lineNumber
    if (line && editCosts.has(`line-${line}`)) {
      showHoverWidget(line, {
        content: `💰 AI Edit Cost: ${editCosts.get(`line-${line}`)?.toFixed(4)}`
      })
    }
  }
})
```

7. Test-Driven Workflow

Integrate testing directly into the editor:

7.1 Inline Test Status

Current: Not implemented

Goal: Test results in gutter

Implementation:

```
// Test result decorations
interface TestResult {
  lineNumber: number
  status: 'passed' | 'failed' | 'pending' | 'running'
  name: string
  duration?: number
  error?: string
}

const addTestDecorations = (results: TestResult[]) => {
  const decorations = results.map(test => ({
    range: new monaco.Range(test.lineNumber, 1, test.lineNumber, 1),
    options: {
      glyphMarginClassName: `test-${test.status}`,
      glyphMarginHoverMessage: {
        value: test.status === 'failed'
          ? `✖ Test failed: ${test.name}\n${test.error}`
          : `✔ Test passed: ${test.name} (${test.duration}ms)`
      }
    }
  }))
  editor.createDecorationsCollection(decorations)
}
```

7.2 Click to Run Test

Current: Not implemented

Goal: Run test by clicking gutter

Implementation:

```
// Gutter click handler
editor.onMouseDown((e) => {
  if (e.target.type === monaco.editor.MouseTargetType.GUTTER_GLYPH_MARGIN) {
    const lineNumber = e.target.position?.lineNumber
    if (lineNumber) {
      const test = findTestAtLine(lineNumber)
      if (test) {
        runSingleTest(test)
      }
    }
  }
})

// Run single test
const runSingleTest = async (test: Test) => {
  // Update decoration to show running
  updateTestDecoration(test.lineNumber, 'running')

  // Call Tauri to run test
  const result = await invoke('run_test', { testId: test.id })

  // Update decoration with result
  updateTestDecoration(test.lineNumber, result.status)

  // Show notification
  showNotification(
    result.status === 'passed'
      ? `✓ Test passed (${result.duration}ms)`
      : `✗ Test failed: ${result.error}`
  )
}
```

7.3 Test Coverage Visualization

Current: Not implemented

Goal: Show coverage per function

Implementation:

```

// Coverage decorations
const addCoverageDecorations = (coverage: Coverage) => {
  const decorations = []

  // Covered lines (green)
  coverage.covered.forEach(line => {
    decorations.push({
      range: new monaco.Range(line, 1, line, 1),
      options: {
        isWholeLine: true,
        className: 'coverage-covered',
        linesDecorationsClassName: 'coverage-covered-gutter'
      }
    })
  })

  // Uncovered lines (red)
  coverage.uncovered.forEach(line => {
    decorations.push({
      range: new monaco.Range(line, 1, line, 1),
      options: {
        isWholeLine: true,
        className: 'coverage-uncovered',
        linesDecorationsClassName: 'coverage-uncovered-gutter'
      }
    })
  })

  editor.createDecorationsCollection(decorations)
}

// Coverage percentage code lens
monaco.languages.registerCodeLensProvider('typescript', {
  provideCodeLenses: (model) => {
    return {
      lenses: functions.map(fn => ({
        range: fn.range,
        command: {
          id: 'show-coverage',
          title: `📊 ${fn.coverage}% coverage`
        }
      }))
    }
  }
})

```

7.4 AI-Powered Test Generation

Current: Not implemented

Goal: Generate tests for selected code

Implementation:

```
// Add "Generate Test" action
editor.addAction({
  id: 'ai.generate-test',
  label: 'AI: Generate Test',
  contextMenuGroupId: 'ai',
  precondition: 'editorHasSelection',
  run: async (ed) => {
    const selection = ed.getSelection()
    if (!selection) return

    const code = ed.getModel()?.getValueInRange(selection)
    if (!code) return

    // Call AI to generate test
    const test = await invoke('generate_test', { code })

    // Open test file and insert
    openFile(test.filepath)
    insertText(test.code)

    showNotification(`✨ Test generated (${test.confidence}% confident)`)
  }
})
```

7.5 Failed Test Navigation

Current: Not implemented

Goal: Navigate between failed tests

Implementation:

```
// Add navigation actions
editor.addAction({
  id: 'test.next-failed',
  label: 'Tests: Go to Next Failed Test',
  keybindings: [monaco.KeyMod.Alt | monaco.KeyCode.F8],
  run: (ed) => {
    const failedTests = getFailedTests()
    const currentLine = ed.getPosition()?.lineNumber || 0
    const nextFailed = failedTests.find(t => t.lineNumber > currentLine)

    if (nextFailed) {
      ed.setPosition({ lineNumber: nextFailed.lineNumber, column: 1 })
      ed.revealLineInCenter(nextFailed.lineNumber)
    }
  }
})

editor.addAction({
  id: 'test.prev-failed',
  label: 'Tests: Go to Previous Failed Test',
  keybindings: [monaco.KeyMod.Shift | monaco.KeyMod.Alt | monaco.KeyCode.F8],
  run: (ed) => {
    const failedTests = getFailedTests().reverse()
    const currentLine = ed.getPosition()?.lineNumber || 0
    const prevFailed = failedTests.find(t => t.lineNumber < currentLine)

    if (prevFailed) {
      ed.setPosition({ lineNumber: prevFailed.lineNumber, column: 1 })
      ed.revealLineInCenter(prevFailed.lineNumber)
    }
  }
})
```

8. Performance Optimization

Optimize Monaco Editor for large codebases:

8.1 Lazy Loading

Current: All content loaded upfront

Goal: Load content on-demand

Implementation:

```
// Virtual file system
const loadFileContent = async (path: string) => {
  // Only load when file is opened
  const content = await invoke('read_file', { path })
  return content
}

// Unload closed files from memory
const unloadFile = (path: string) => {
  const model = monaco.editor.getModel(monaco.Uri.file(path))
  if (model) {
    model.dispose()
  }
}
```

8.2 Virtual Scrolling

Current: Not implemented

Goal: Render only visible lines

Already Built-in to Monaco: Monaco uses virtual scrolling by default for large files. Ensure these options are set:

```
const editorOptions = {
  ...existingOptions,
  stopRenderingLineAfter: 10000, // Already set
  maxTokenizationLineLength: 20000, // Already set
  largeFileOptimizations: true // Add this
}
```

8.3 Debouncing Expensive Operations

Current: Auto-save has 2s delay

Goal: Debounce all expensive operations

Implementation:

```
// Debounce AI suggestions
const debouncedGetAISuggestions = debounce(async (code: string) => {
  const suggestions = await invoke('get_ai_suggestions', { code })
  updateSuggestions(suggestions)
}, 500)

// Debounce test coverage updates
const debouncedUpdateCoverage = debounce(async () => {
  const coverage = await invoke('get_coverage')
  updateCoverageDecorations(coverage)
}, 1000)

// Debounce git status checks
const debouncedCheckGitStatus = debounce(async () => {
  const status = await invoke('git_status')
  updateGitDecorations(status)
}, 300)
```

8.4 Memoization

Current: Not implemented

Goal: Cache expensive computations

Implementation:

```
// Memoize language detection
const getLanguage = useMemo(() => {
  return (filepath: string) => {
    const ext = filepath.substring(filepath.lastIndexOf('.'))
    return languageMap[ext.toLowerCase()] || 'plaintext'
  }
}, [])

// Memoize syntax highlighting
const tokenProvider = useMemo(() => ({
  getInitialState: () => ({ /* ... */ }),
  tokenize: (line: string, state: any) => {
    // Expensive tokenization logic
    return { tokens: [], endState: state }
  }
}), [])

// Memoize decorations
const decorations = useMemo(() => {
  return computeDecorations(testResults, aiSuggestions, gitStatus)
}, [testResults, aiSuggestions, gitStatus])
```

8.5 Web Worker Usage

Current: Not implemented

Goal: Offload expensive operations to Web Workers

Implementation:

```
// Create Web Worker for expensive operations
const worker = new Worker(new URL('./editor-worker.ts', import.meta.url))

// Offload syntax analysis to worker
const analyzeSyntax = (code: string) => {
  return new Promise((resolve) => {
    worker.postMessage({ type: 'analyze', code })
    worker.onmessage = (e) => {
      if (e.data.type === 'analysis-complete') {
        resolve(e.data.result)
      }
    }
  })
}

// Offload git diff calculation to worker
const calculateDiff = (original: string, modified: string) => {
  return new Promise((resolve) => {
    worker.postMessage({ type: 'diff', original, modified })
    worker.onmessage = (e) => {
      if (e.data.type === 'diff-complete') {
        resolve(e.data.result)
      }
    }
  })
}
```

9. Accessibility

Ensure Monaco Editor is fully accessible:

9.1 Screen Reader Support

Current: Basic Monaco support

Goal: Enhanced screen reader experience

Implementation:


```

// Enable screen reader optimizations
const editorOptions = {
  ...existingOptions,
  accessibilitySupport: 'on',
  accessibilityPageSize: 10,
  ariaLabel: 'Code Editor',
  // Announce editor changes
  screenReaderAnnounceInlineSuggestion: true
}

// Add ARIA labels to decorations
const testDecoration = {
  range: testRange,
  options: {
    glyphMarginHoverMessage: {
      value: 'Test passed',
      isTrusted: true
    },
    hoverMessage: {
      value: 'Test passed: unit/auth.test.ts',
      isTrusted: true
    },
  },
  // Add ARIA label
  glyphMarginClassName: 'test-passed',
  description: 'Test passed indicator'
}

// Announce test results
const announceTestResult = (result: TestResult) => {
  const announcement = document.createElement('div')
  announcement.setAttribute('role', 'status')
  announcement.setAttribute('aria-live', 'polite')
  announcement.textContent = result.status === 'passed'
    ? `Test ${result.name} passed`
    : `Test ${result.name} failed: ${result.error}`
  document.body.appendChild(announcement)
  setTimeout(() => announcement.remove(), 1000)
}

```

9.2 High Contrast Mode

Current: Not implemented

Goal: Support high contrast themes

Implementation:

```
// Detect high contrast mode
const isHighContrast = window.matchMedia('(prefers-contrast: high)').matches

// Define high contrast theme
const highContrastTheme: editor.IStandaloneThemeData = {
  base: 'hc-black',
  inherit: true,
  rules: [
    { token: 'keyword', foreground: 'FFFFFF', fontStyle: 'bold' },
    { token: 'string', foreground: 'FFFF00' },
    { token: 'comment', foreground: '00FF00', fontStyle: 'italic' }
  ],
  colors: {
    'editor.background': '#000000',
    'editor.foreground': '#FFFFFF',
    'editor.lineHighlightBackground': '#333333',
    'editorCursor.foreground': '#FFFF00'
  }
}

// Apply high contrast theme
if (isHighContrast) {
  monaco.editor.defineTheme('heaven-high-contrast', highContrastTheme)
  monaco.editor.setTheme('heaven-high-contrast')
}
```

9.3 Keyboard Navigation

Current: Basic Monaco support

Goal: Enhanced keyboard navigation

Implementation:

```
// Add custom keyboard commands for accessibility
editor.addCommand(
  monaco.KeyMod.Alt | monaco.KeyCode.F1,
  () => announceCurrentLine()
)

editor.addCommand(
  monaco.KeyMod.Alt | monaco.KeyCode.F2,
  () => announceCurrentSelection()
)

// Announce current line
const announceCurrentLine = () => {
  const position = editor.getPosition()
  const line = editor.getModel()?.getLineContent(position?.lineNumber || 1)
  announce(`Line ${position?.lineNumber}: ${line}`)
}

// Announce selection
const announceCurrentSelection = () => {
  const selection = editor.getSelection()
  const text = editor.getModel()?.getValueInRange(selection!)
  announce(`Selected: ${text}`)
}
```

9.4 Focus Management

Current: Basic focus

Goal: Proper focus trapping and management

Implementation:

```
// Focus trap for overlays
const trapFocus = (container: HTMLElement) => {
  const focusableElements = container.querySelectorAll(
    'button, [href], input, select, textarea, [tabindex]:not([tabindex="-1"])'
  )

  const firstElement = focusableElements[0] as HTMLElement
  const lastElement = focusableElements[focusableElements.length - 1] as HTMLElement

  container.addEventListener('keydown', (e) => {
    if (e.key === 'Tab') {
      if (e.shiftKey && document.activeElement === firstElement) {
        e.preventDefault()
        lastElement.focus()
      } else if (!e.shiftKey && document.activeElement === lastElement) {
        e.preventDefault()
        firstElement.focus()
      }
    }
  })
}


// Return focus to editor after dismissing overlay
const returnFocus = () => {
  editor.focus()
}
```


9.5 ARIA Labels

Current: Minimal ARIA support

Goal: Comprehensive ARIA labels

Implementation:

```
// Add ARIA labels to all interactive elements
<EditorHeader
  aria-label="Code editor header"
  role="banner"
>
  <button
    aria-label={`Toggle minimap. Currently ${showMinimap ? 'on' : 'off'}`}
    onClick={toggleMinimap}
  >
    
  </button>

  <button
    aria-label={`Format document. Shortcut: ${formatShortcut}`}
    onClick={formatDocument}
  >
    
  </button>
</EditorHeader>

// Add ARIA live regions for announcements
<div
  role="status"
  aria-live="polite"
  aria-atomic="true"
  className="sr-only"
>
  {statusMessage}
</div>
```

10. Implementation Roadmap

Phase 1: Foundation (Week 1-2)

Priority: HIGH

- [] **Task 1.1:** Implement contextual header (hide after 3s)
 - Estimated Time: 4 hours
 - Dependencies: useContextualVisibility hook (already exists)
 - Solo-Git Integration: None
- [] **Task 1.2:** Fade minimap to 30% opacity
 - Estimated Time: 2 hours
 - Dependencies: CSS updates
 - Solo-Git Integration: None
- [] **Task 1.3:** Add breadcrumb on hover
 - Estimated Time: 3 hours
 - Dependencies: Overlay widget API
 - Solo-Git Integration: File path from Tauri
- [] **Task 1.4:** Hide tabs when single file open

- Estimated Time: 2 hours
- Dependencies: Tab management logic
- Solo-Git Integration: None
- [] **Task 1.5:** Implement keyboard shortcuts for Solo-Git
- Estimated Time: 6 hours
- Dependencies: Command registration
- Solo-Git Integration: All (workpad, tests, AI, etc.)

Total: ~17 hours

Phase 2: Git Integration (Week 3-4)

Priority: HIGH

- [] **Task 2.1:** Add workpad status indicators in gutter
- Estimated Time: 8 hours
- Dependencies: Tauri git status API
- Solo-Git Integration: Workpad type detection
- [] **Task 2.2:** Implement inline git blame
- Estimated Time: 6 hours
- Dependencies: Tauri git blame API
- Solo-Git Integration: Author, commit message, timestamp
- [] **Task 2.3:** Add workpad diff visualization
- Estimated Time: 12 hours
- Dependencies: Monaco diff editor
- Solo-Git Integration: Trunk vs workpad comparison
- [] **Task 2.4:** Implement conflict resolution UI
- Estimated Time: 10 hours
- Dependencies: Git merge conflict detection
- Solo-Git Integration: AI-powered resolution
- [] **Task 2.5:** Add commit staging from editor
- Estimated Time: 8 hours
- Dependencies: Git staging API
- Solo-Git Integration: Chunk-level staging

Total: ~44 hours

Phase 3: AI Integration (Week 5-6)

Priority: HIGH

- [] **Task 3.1:** Add AI-generated code markers
 - Estimated Time: 8 hours
 - Dependencies: AI tracking system
 - Solo-Git Integration: Model, confidence, cost
- [] **Task 3.2:** Implement inline AI suggestions
 - Estimated Time: 12 hours
 - Dependencies: Completion provider API
 - Solo-Git Integration: Multi-model routing
- [] **Task 3.3:** Add confidence score badges
 - Estimated Time: 4 hours
 - Dependencies: Content widget API
 - Solo-Git Integration: AI confidence metrics
- [] **Task 3.4:** Show model tier indicators
 - Estimated Time: 6 hours
 - Dependencies: View zones API
 - Solo-Git Integration: GPT-4/DeepSeek/Llama
- [] **Task 3.5:** Implement cost tracking per edit
 - Estimated Time: 8 hours
 - Dependencies: Edit tracking system
 - Solo-Git Integration: Cost calculation API

Total: ~38 hours

Phase 4: Test Integration (Week 7-8)

Priority: HIGH

- [] **Task 4.1:** Add inline test status in gutter
 - Estimated Time: 8 hours
 - Dependencies: Test runner integration
 - Solo-Git Integration: Test results from Tauri
- [] **Task 4.2:** Implement click-to-run tests
 - Estimated Time: 6 hours
 - Dependencies: Gutter click handler
 - Solo-Git Integration: Single test execution
- [] **Task 4.3:** Add test coverage visualization

- Estimated Time: 10 hours
- Dependencies: Coverage data API
- Solo-Git Integration: Coverage metrics
- [] **Task 4.4:** Implement failed test navigation
- Estimated Time: 4 hours
- Dependencies: Test result tracking
- Solo-Git Integration: Failed test locations
- [] **Task 4.5:** Add AI test generation
- Estimated Time: 8 hours
- Dependencies: AI code generation
- Solo-Git Integration: Test generation API

Total: ~36 hours

Phase 5: CI/CD Integration (Week 9)

Priority: MEDIUM

- [] **Task 5.1:** Add pipeline status widget
- Estimated Time: 6 hours
- Dependencies: CI/CD API integration
- Solo-Git Integration: Build status
- [] **Task 5.2:** Show build progress indicator
- Estimated Time: 4 hours
- Dependencies: Real-time updates
- Solo-Git Integration: Build streaming
- [] **Task 5.3:** Add failed build error markers
- Estimated Time: 6 hours
- Dependencies: Build error parsing
- Solo-Git Integration: Error locations
- [] **Task 5.4:** Implement smoke test results
- Estimated Time: 4 hours
- Dependencies: Smoke test API
- Solo-Git Integration: Post-merge tests

Total: ~20 hours

Phase 6: Performance (Week 10)

Priority: MEDIUM

- [] **Task 6.1:** Implement lazy loading for files
 - Estimated Time: 8 hours
 - Dependencies: Virtual file system
 - Solo-Git Integration: File loading API
- [] **Task 6.2:** Add debouncing for expensive operations
 - Estimated Time: 4 hours
 - Dependencies: Debounce utility
 - Solo-Git Integration: AI, git, test operations
- [] **Task 6.3:** Implement memoization
 - Estimated Time: 6 hours
 - Dependencies: React.useMemo
 - Solo-Git Integration: Decoration computation
- [] **Task 6.4:** Offload to Web Workers
 - Estimated Time: 12 hours
 - Dependencies: Web Worker API
 - Solo-Git Integration: Syntax analysis, diff

Total: ~30 hours

Phase 7: Accessibility (Week 11)

Priority: MEDIUM

- [] **Task 7.1:** Enhance screen reader support
 - Estimated Time: 8 hours
 - Dependencies: ARIA labels
 - Solo-Git Integration: All features
- [] **Task 7.2:** Implement high contrast mode
 - Estimated Time: 6 hours
 - Dependencies: Theme system
 - Solo-Git Integration: None
- [] **Task 7.3:** Add keyboard navigation enhancements
 - Estimated Time: 6 hours
 - Dependencies: Custom commands
 - Solo-Git Integration: All features
- [] **Task 7.4:** Implement focus management

- Estimated Time: 4 hours
- Dependencies: Focus trap utility
- Solo-Git Integration: None

Total: ~24 hours

Phase 8: Polish & Testing (Week 12)

Priority: HIGH

- [] **Task 8.1:** Write unit tests for all features
 - Estimated Time: 16 hours
 - Dependencies: Test framework
 - Solo-Git Integration: All features
- [] **Task 8.2:** Perform accessibility audit
 - Estimated Time: 4 hours
 - Dependencies: Accessibility tools
 - Solo-Git Integration: None
- [] **Task 8.3:** Performance profiling and optimization
 - Estimated Time: 8 hours
 - Dependencies: Performance tools
 - Solo-Git Integration: All features
- [] **Task 8.4:** Documentation and user guide
 - Estimated Time: 8 hours
 - Dependencies: None
 - Solo-Git Integration: All features

Total: ~36 hours

Summary

Total Estimated Time: ~245 hours (~6 weeks full-time)

Critical Path:

1. Foundation → Git Integration → AI Integration → Test Integration
2. CI/CD → Performance → Accessibility → Polish

Highest ROI Tasks:

1. Workpad status indicators (Phase 2)
2. Inline AI suggestions (Phase 3)
3. Inline test status (Phase 4)
4. AI-generated code markers (Phase 3)
5. Contextual header (Phase 1)

Dependencies:

- Tauri backend APIs for git, AI, tests, CI/CD
- Monaco Editor decorations API
- React hooks for state management
- Web Workers for performance

Success Metrics:

- [] Editor feels “invisible” (minimal chrome)
- [] All Solo-Git features accessible from editor
- [] Performance: <50ms for all operations
- [] Accessibility: WCAG 2.1 AA compliant
- [] User satisfaction: 9/10+ rating

Next Steps

1. **Review** this document with team
 2. **Prioritize** tasks based on user feedback
 3. **Prototype** highest ROI features first
 4. **Iterate** based on user testing
 5. **Document** all APIs and patterns
 6. **Ship** incrementally (weekly releases)
-

This TODO document will be updated as features are implemented and new requirements emerge.