Full Game Prototype

LINK TO GITHUB:

https://github.com/IMRAN-Anisha/CMSCI-2025

Checklist (before and after - use of AI to compare)

OOP Refinement

- **Encapsulation:**
  - *Make maze, score, position private (e.g., self._maze):* In puzzle_game.py, self.maze, self.score, self.player_pos are not private yet—they're just self.maze (line 37). Should be self._maze.
  - *Add getters/setters (e.g., get_score(), set_position()):* No getters/setters yet. For example, self.score (line 39) is accessed directly in draw_game (line 529). Should add def get_score(self): return self._score.
- **Abstraction:**
  - *Create BasePuzzle class with generate_puzzle() for all puzzles:* Not done. PuzzleGame (line 34 in puzzle_game.py) is standalone, doesn't inherit from a BasePuzzle. Should make a BasePuzzle with generate_puzzle() like generate_maze (line 108).
  - *Make PathfindingAlgorithm base class for solvers:* DFSSolver (in source/dfs_solver.py) doesn't inherit from a base class. Should create PathfindingAlgorithm with a solve() method that DFSSolver overrides (line 126 in puzzle_game.py uses solver.solve()).
  - *GameManager should work with puzzles generically:* main.py (line 34) directly uses PuzzleGame. Should use a generic BasePuzzle type to handle WordGame, SudokuGame too.
- **Inheritance:**
  - *MazePuzzle inherits from BasePuzzle:* PuzzleGame (line 34 in puzzle_game.py) doesn't inherit from anything. Should inherit from a BasePuzzle class.
  - *Add difficulty subclasses (e.g., EasyMaze, HardMaze):* Difficulty is handled in start_game (line 119) with a dict (difficulty_settings). Should make EasyMaze, MediumMaze, etc., as subclasses.
  - *AI difficulty inherits from AIAdaptiveSystem:* No AI system yet. start_game (line 119) sets static time limits (e.g., self.time_limit = 60 for Hard). Should create AIAdaptiveSystem for dynamic difficulty.
- **Polymorphism:**
  - *Override generate_puzzle() in puzzle subclasses:* No BasePuzzle yet, so generate_maze (line 108) isn't overridden. Should override in EasyMaze, etc.

- *AI difficulty changes via method overriding:* No AI yet. Difficulty is static in start_game (line 119). Should override an AI method for dynamic changes.

Other notes:

- Refactoring made it less of a headache.
- Had to fix paths so Python could find my files.
- Menus are cool with buttons that change color.
- Maze is fun, but the white play screen bugs me now.
- Instructions need a better "Back" button.

```
Traceback (most recent call last):
  File "c:\Users\anish\OneDrive\Documents\GitHub\CMSCI-2025\maze puzzle game FINAL\tests\test_word-game.py", lin
e 6, in <module>
    from games.word_game import WordGame
ModuleNotFoundError: No module named 'games'
PS C:\Users\anish\OneDrive\Documents\GitHub\CMSCI-2025> python -m unittest tests.test_word_game
E
======================================================================
ERROR: tests (unittest.loader._FailedTest.tests)
----------------------------------------------------------------------
ImportError: Failed to import test module: tests
Traceback (most recent call last):
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.11_3.11.2544.0_x64__qbz5n2kfra8p0\Lib\uni
ttest\loader.py", line 162, in loadTestsFromName
    module = __import__(module_name)
             ^^^^^^^^^^^^^^^^^^^^^^^
ModuleNotFoundError: No module named 'tests'


----------------------------------------------------------------------
Ran 1 test in 0.001s
```

Table of Errors and Fixes:

Error Description, Location, Terminal Snippet	Solution Description, Code Snippet
ModuleNotFoundError for UI (Relative Import Issue) **Location:** main.py, line 9 **Terminal:** ModuleNotFoundError: No module named 'UI'	**Fix import path to use source.UI** **Code:** from source.UI import MainMenu

ImportError for Missing constants (No __init__.py) **Location:** puzzle_game.py, line 9 **Terminal:** ImportError: cannot import name 'WIDTH' from 'source.constants'	**Add source/__init__.py and export constants** **Code:** from .constants import HEIGHT, WIDTH
ModuleNotFoundError for dfs_solver (Incorrect Path) **Location:** puzzle_game.py, line 8 **Terminal:** ModuleNotFoundError: No module named 'dfs_solver'	**Fix import path to use source.dfs_solver** **Code:** from source.dfs_solver import DFSSolver
ImportError for Circular Import (UI and Main) **Location:** UI.py importing main.py **Terminal:** ImportError: cannot import name 'run_game' from partially initialized module 'source.main'	**Remove circular import by moving shared logic to a separate file** **Action:** (Avoid from source.main import run_game in UI.py)
NameError for Undefined WHITE Constant **Location:** puzzle_game.py, line 199 **Terminal:** NameError: name 'WHITE' is not defined	**Import constants from source.constants** **Code:** from source.constants import *
White Screen in Game Selection Menu **Location:** UI.py, line 116 **Terminal:** (No error, visual issue)	**Change fill color to black** **Code:** self.screen.fill((0, 0, 0))
No Q Key to Quit in Maze Game **Location:** puzzle_game.py, line 177 **Terminal:** (No error, functionality missing)	**Add Q key to quit** **Code:** if event.key in (pygame.K_q, pygame.K_Q): self.quit_game(return_to_menu=True)
IndexError for Out-of-Bounds Maze Access **Location:** puzzle_game.py, line 195 **Terminal:** IndexError: list index out of range	**Add bounds checking for movement** **Code**: if x > 0 and x < GRID_SIZE-1 and y > 0 and y < GRID_SIZE-1
KeyError in Difficulty Settings **Location:** puzzle_game.py, line 121 **Terminal:** KeyError: 'invalid_difficulty'	**Validate difficulty input** **Code:** if difficulty not in difficulty_settings: raise ValueError("Invalid difficulty")

No Error Handling for Invalid Maze Size **Location:** puzzle_game.py, line 18 **Terminal:** IndexError: list index out of range	**Add validation for GRID_SIZE** **Code:** if GRID_SIZE < 3: raise ValueError("GRID_SIZE must be at least 3")
DFS Solver Fails for Disconnected Mazes **Location:** puzzle_game.py, line 65 **Terminal:** AttributeError: 'NoneType' object has no attribute '__getitem__'	**Check if solution_path exists** **Code:** if self.solution_path is None: raise RuntimeError("No path found")
DFS Solver Performance Issue with Large Mazes **Location:** source/dfs_solver.py, solve() **Terminal**: (No error, game lags)	**Optimize DFS with iterative approach** **Code:** (Use a loop instead of recursion in solve())
DFS Solver Doesn't Handle Invalid Start/End **Location:** source/dfs_solver.py, solve() **Terminal:** (No error, solver hangs)	**Validate start/end positions** **Code:** if maze[start[0]][start[1]] == 1: raise ValueError("Start is a wall")
No Auto-Adjustment for Difficulty **Location**: puzzle_game.py, start_game() (line 119) **Terminal:** (No error, feature missing)	Initially wanted AI auto-adjustment, but used static difficulty settings instead. AI would've been complex to tune (e.g., tracking performance metrics like time, hints) and resource-heavy for a small game. Static settings are simpler, predictable, and efficient for quick testing

UI Improvements for Next Time:

- **Add Visual Feedback for Hints:**
  - Current: Hints show a yellow square (draw(), line 208).
  - Improvement: Add a sound effect or text notification (e.g., "Hint Used!").
- **Show Difficulty in HUD:**
  - Current: HUD doesn't display difficulty (draw(), line 215).
  - Improvement: Add text like "Difficulty: Easy" to the HUD.
- **Fix Instructions Screen "Back" Button:**
  - Current: "Back" stops the game (UI.py, line 169).
  - Improvement: Return to the main menu (return "back").
- **Unify Font Sizes Across Screens:**
  - Current: puzzle_game.py uses font size 30, UI.py uses 36.

    - Improvement: Standardize to a single font size (e.g., 30).
- **Add Animated Player Movement:**
    - Current: Movement is grid-based (handle_movement(), line 183).
    - Improvement: Add smooth transitions between cells.
- **Improve Maze Visibility:**
    - Current: Maze uses basic colors (draw(), line 199).
    - Improvement: Add textures or sprites for walls, player, and goal.

Comparison Table: Initial Proposal vs. Finished Game

Feature/Requirement	Initial Proposal	Finished Game	Status
Game Type	Dynamic puzzle game with multiple types (maze, logic, number puzzles)	DFS-based maze puzzle game With a word and number game.	Done
Procedural Puzzle Generation	Generate puzzles (maze, Sudoku, etc.) using algorithms like DFS for variety	Maze generation using DFS (generate_puzzle() in puzzle_game.py, line 45)	Done
Multiple Puzzle Types	Include logical (Sudoku), spatial (maze), and numerical puzzles	Maze puzzles, WordGame and SudokuGame are implemented	Done
AI-Driven Adaptive Difficulty	AI adjusts difficulty based on player performance (time, hints, score)	Static difficulty settings (difficulty_settings in puzzle_game.py, line 119); AIAdaptiveSystem added but not fully utilized	Partially Done

Dynamic Hint System	AI uses DFS to provide hints dynamically	Manual hint system (H key toggles self.show_hint, puzzle_game.py, line 169); no AI integration	Partially Done
Score System	Performance tracking with ranking-based difficulty adjustments	Basic scoring (ScoreManager in source/score_manager.py) Further improvements in SEM 2	Done
Timer System	Timer tracks progress and influences difficulty dynamically	Timer implemented (self.timer in puzzle_game.py, line 149);	Done
OOP Principles (Encapsulation, Abstraction, Inheritance, Polymorphism)	Use OOP for modular, scalable design (Puzzle base class, subclasses, etc.)	Implemented: BasePuzzle (source/base_puzzle.py), EasyMaze, MediumMaze, HardMaze subclasses; encapsulation not fully done	Mostly Done
DFS Integration	DFS for maze generation, pathfinding, and hint system	DFS used for maze generation and pathfinding	Done
UI Design	Clean, user-friendly UI with animations and transitions	Basic UI (source/UI.py); menus, buttons, HUD; no animations or transitions (not required for simple game)	Partially Done
Save/Load System	Save/load player progress	Yet to be implemented: SEM 2	Not Done
Game Progression	Tutorial stage, core gameplay, AI adaptation, challenge mode with leaderboards	Core gameplay (maze); difficulty selection;but no leaderboards Yet to be implemented: SEM 2	Partially Done
Target Platforms	Run on desktop and mobile devices	Desktop only (uses Pygame, tested via run.py)	Partially Done

Error Handling	Robust error handling for invalid inputs, file I/O, runtime exceptions, puzzle generation	Basic error handling (e.g., bounds checks in handle_movement(), line 183); Range of Unit tests	Done
Testing Strategy	Unit tests, integration tests, system tests for all components	Manual testing during development Isolation and refactoring	Done
Ethical/Legal Considerations	Use non-copyrighted assets, ensure fair gameplay, comply with data privacy	Used basic Pygame assets; no data collection	Done

Justifications for Deviations Table

Feature	Initial Plan	What You Did	Why You Didn't Fully Implement It	Why Your Approach Was More Efficient
AI-Driven Adaptive Difficulty	AIAdaptiveSystem to adjust difficulty based on performance (time, hints, score).	Static settings (difficulty_settings, puzzle_game.py, line 119); basic AIAdaptiveSystem added.	Complexity: Tuning AI needed extensive testing to balance metrics. Resource Constraints: Added overhead, against low-resource goal.	Static settings were simpler, predictable, and lightweight, letting you focus on maze generation and movement.

Multiple Puzzle Types	Include maze, Sudoku, word puzzles for variety.	Focused on maze (PuzzleGame); WordGame, SudokuGame as placeholders (main.py, lines 37–39).	Time Constraints: Unique mechanics for each type were too ambitious. Focus on Core Mechanic: Prioritized DFS maze mastery.	Focusing on one type let you polish the maze game (e.g., ScoreManager, difficulty subclasses) without spreading efforts.
Dynamic Hint System	AI-driven hints using DFS to suggest next steps dynamically.	Manual hints (H key toggles self.show_hint, puzzle_game.py, line 208).	Complexity of AI Integration: Needed AIAdaptiveSystem and DFSSolver integration, adding complexity.	Manual system was quicker to build, reliable, and met the need for assistance without overcomplicating the code.
Save/Load System	Save/load player progress (scores, positions).	Not implemented.	Scope Limitation: File I/O, error handling, and UI for save/load were beyond scope. Focus on Gameplay: Prioritized core mechanics.	Skipping this ensured a functional game within your timeline, focusing on playable features like maze generation.
UI Animations and Transitions	Clean UI with animations (e.g., animated player movement, screen transitions).	Basic UI with menus (source/UI.py); no animations (handle_movement(), puzzle_game.py, line 183).	Technical Challenge: Animations added complexity to game loop. Time/Skill Constraints: Learning Pygame animations took time.	Functional UI was enough for gameplay, letting you focus on mechanics and stability (e.g., fixing white screen issues).

Testing Strategy	Unit, integration, system tests using unittest or pytest.	Manual testing (e.g., via run.py).	Learning Curve: Setting up tests was time-intensive. Small Scale: Manual testing caught major bugs (e.g., ModuleNotFoundError).	Manual testing allowed quick iterations, focusing on fixing issues (e.g., import errors, UI bugs) without test setup overhead.
Target Platforms	Run on desktop and mobile devices.	Desktop only (Pygame, tested via run.py).	Pygame Limitations: No native mobile support; porting was complex. Development Focus: Prioritized a working desktop version.	Targeting desktop ensured compatibility and a functional game without the complexity of mobile porting.