Project: Build Your Own World — CS 61B

Spring 2024

Ajay Sharma

Remark

If you'd like to review my code or discuss design decisions, please reach out at ajay.sharma@berkeley.edu.

Overview

In this capstone project, I created a 2D tile–based world exploration engine in Java. The project is split into two phases: (1) *World Generation* (Project 3A), where I built a seeded, pseudo–random generator producing interconnected rooms and hallways; (2) *Gameplay* (Project 3B), where I implemented a user interface—main menu, avatar movement, HUD, and persistent save/load functionality. Beyond the core requirements, I also delivered three ambition features: line–of–sight rendering, an autonomous pathfinding entity, and in–game world regeneration.

Description

- **World Generation (3A)** I represented the world as a seeded `TETile[][]` grid, filling rooms and hallways subject to connectivity and minimum–dead–end constraints. Seeding with Java's `Random` ensured reproducibility for grading and debugging. The generator is designed to be pseudo–random: each new world uses a fresh seed, allowing dynamic world regeneration without restarting the JVM.
- **Main Menu** I crafted a menu allowing users to start a new game, load from `save.txt`, or quit. Seed entry echoed each digit in real time, and pressing `S/s` launched world generation.
- **Interactivity** I rendered an avatar on the first available floor tile and enabled WASD controls for movement, blocking motion into walls. A game loop updated state deterministically based on key events.
- **HUD** I implemented a non–flickering heads–up display showing the description of the tile under the mouse cursor, updating on mouse–move events via `StdDraw`.
- **Saving & Loading** In "world mode", pressing `:Q/:q` serializes the current seed, avatar position, and any dynamic state to `save.txt` before exiting. Loading rehydrates the exact game state.
- **Graph Data Structure** I created a dedicated `Graph` class to model connectivity between floor tiles. Each open tile is a vertex, and edges connect adjacent tiles. This abstraction supports both pathfinding and line–of–sight computations.

- **Ambition Features** To demonstrate advanced functionality, I added:
  - *Line–of–Sight Rendering:* Implemented a Bresenham–based algorithm using the `Graph` structure to mask tiles outside the avatar's view radius. Toggled on/off with the `L/l` key.
  - *Pathfinding Entity:* Introduced an enemy NPC that uses breadth–first search (BFS) on the `Graph` to chase the avatar. The BFS algorithm computes the shortest path each turn, which can be visualized when toggled via `K/k`.
  - *Dynamic World Regeneration:* Enabled pressing `N/n` at any time in world mode (or upon "game over") to spawn a brand–new world with a fresh seed, without restarting the JVM.

Methodology

- **Object–Oriented Design**: Structured the codebase around classes such as `WorldGenerator`, `GameEngine`, `Graph`, and `Entity`, following encapsulation and separation of concerns.
- **Data Structures**: Used a 2D `TETile[][]` array for the world and the custom `Graph` class for connectivity. The `Graph` uses an adjacency list representation for efficient traversal.
- **Rendering & Input**: Leveraged `TERenderer` and `StdDraw` for high–frequency redraws with non–blocking key/mouse event polling.
- **Algorithms**:
  - *World Generation*: Randomly place rectangular rooms, carve width–1 hallways ensuring full connectivity, and verify >50% fill.
  - *Line–of–Sight*: Ray–cast along edges in the `Graph`, checking for occlusions by walls.
  - *Pathfinding*: Performed BFS on the `Graph` each game tick to compute the enemy's shortest path to the avatar.
  - *Save/Load*: Serialize seed, avatar coordinates, and dynamic flags via a simple text protocol to maintain determinism.
- **Testing**: Wrote comprehensive unit tests (JUnit) for world generation connectivity, pathfinding correctness, HUD updates, save/load integrity, and ambition feature toggles. Tests ensure $O(1)$ tile queries and $O(V + E)$ BFS performance on the `Graph`.

Results and Insights

My implementation passes all autograder and checkoff criteria. The generator yields pseudo–random but reproducible worlds, and the gameplay interface responds smoothly to user input. Ambition features enrich the experience: line–of–sight fosters strategic navigation, BFS–driven pathfinding NPCs introduce emergent challenge, and dynamic world regeneration streamlines exploration. This project honed my ability to architect systems with clear OOP structure, integrate graph algorithms into interactive applications, and rigorously validate functionality through unit testing.