Project 5A: BYOW Design

Author: Jameson Hearn
Last Updated: 11/15/2025

Data

What data will you be using? What does the data consist of and how will you parse it for your program? How will your program use the data?

World Data:
Grid via TETile[][] world storing distinct tile types for every x,y coordinate in the map
Tile Types: Floor, Wall, Nothing (outside map), Avatar, Rock, Ladder
Tile data used to render world via TERenderer

Random Seeds:
Input via N<digits>S (parsed by reading between N,S chars)
Seed initializes a Random object ensuring random but deterministic world generation

World Generation Data:
Rooms: position, width, height
Hallways: path segments (randomized)
Derived via RNG
Hallways and Rooms generated based on design limitations and pseudorandom requirements
Rock tiles defined via random subset of floor tiles
Single Rock tile contains Ladder, available after breaking rock
Player location on top of Ladder triggers new world generation

Game State Data:
Player position (x,y coord)
Saved command Str
Load file content
HUD with description detail via tile-under-mouse
Depth level (how many ladders taken)

Saving/Loading Data:
Saved format: <all commands typed minus :Q>
Eg: N1237857Swasswwaddasdwwwassdadwwwsasd
L restores game state based on commands and seed

Data Structures

What data structures will you be using? How will your data structures use the data? How will your program use the data structures?

Variable	**Data Structure**	**Purpose**	**Implementation**
world	TETile[][]	Map representation	New TETile filled with nothing then split into tiles
rooms	List<Room>	Stores all rooms	ArrayList<Room>
hallways	List<List<Point>>	Stores all hallways segments	ArrayList<List<Point>>
floorTiles	Set<Point>	Stores all valid floor positions	HashSet
rockPositions	Set<Point>	Lookup for coords containing rocks	HashSet<Point>
ladderPos	Point	Coordinate location of ladder	Rand selection from rockPositions
rng	Random	Pseudorandom num generator	Random(seed) from parsed user intput
player	Player obj	Player position and state	Class
commandHistory	StringBuilder	Records all (meaningful) player inputs	Updated in real time by keyboard
gameState	GameState Enum	Menu, seedInput, World, Saving	Appends keys as processed
levelNum	int	Depth of world (dungeon level)	Starts at 0, increments based on ladder
levelSeed	long	Seed for current level RNG	Initial levelSeed parsed from input, deeper levels derived from initial seed + levelNum
mousePos	Point	Track mouse tile for HUD	Read from StdDraw.mouseX/Y cast into int
visitedRooms	Set<Room>	Validate rooms can be reached during	HashSet<Room>

		connection validation	
nextWorldParams	Config obj	Store params to generate next level deterministically	Class containing levelNum, levelSeed

Algorithms

What algorithms will you be using? How will your algorithms use the data structures and data? How will your program use the algorithms?

World Generation

input : levelSeed output: world map

Initialize world with NOTHING

Generate rooms:

- Sample rand width/height within bounds
- Choose random up/left corner
- Reject rooms that overlap or clip outside bounds
- Add valid rooms to list

Generate hallways:

- Connect rooms (spanning tree or random linking with validation)
- Pick random points along room perimeter, define and start and target
    - Add randomization bias - 70/30 towards target/lateral to target
    - Stop at distance 0 to target, create limit to ensure no wandering hallways/infinite loops, reset if hit
- Validate creation - no dead ends

Fill Walls:

- Surround outer edges with wall tiles

Place Rocks:

- Choose random subset of floorTiles
- Replace with ROCK tiles
- Add coords to rockPositions

Hide Ladder:

- Select single Point from rockPositions at random
- Store as ladderPos
- Initially tile remains as Rock

Place player:

- Choose random floor tile not occupied with rock

- Place avatar tile, update player x,y coords

Correctness validation:

- Ensure meet map specifications for fullness, dead ends, etc

Main Menu

States:

- Menu -> Seed input -> Playing
- Menu -> load -> playing
- Playing -> saving (when : called) -> quit

Input sequence:

- N11111111S starts game
- L loads save
- :Q saves and exits

Interactivity

Movement:

WASD:

- Compute target
    - If floor, move
    - If rock, block move
    - If ladder, move and trigger level transition
    - If Wall or nothing, block move

Rock Breaking:

Trigger with ‘B’

- Determine tile in front of avatar
- If rock:
    - Replace with floor
    - Remove from rockPositions
    - (nx, ny) == ladderPos, reveal ladder

Ladder Descend:

When stepping onto ladder tile:

- Increment levelNum
- Compute new levelSeed
- Generate new world

- Place avatar

HUD

Each frame:

- Read mouse tile position
- If in bounds - display world[x][y] description
- Display current depth

Saving & Loading

Saving: :Q

- Detect : and then Q
- Write commandHistory.toString() to file
- Exit program

Loading: L

- Read saved command string
- Replay characters
- World generates deterministically
- Player position and broken rock states restored through replay

Complexity

What is the input? How does the time and space complexity change with the input? What are the bounds?

World Generation:

- Room generation:
  - Attempt R rand rooms
  - Check overlap (worst case O(R^2))
  - R is typically small, so still negligible
- Hallway generation:
  - Connecting R rooms with rand hallways:
    - O(R) if randomized connectivity pattern
    - O(RlogR) if using spanning tree
- Wall Generation:
  - Surrounding all floor tiles with walls requiring scanning map once O(WxH)
- Rock placement:
  - Selecting rock tiles requires iteration over floor tiles O(WxH)
  - Maintaining rock sets is O(1) per tile
- Ladder placement:

        - Random rock picked O(1)

Total generation:
Time: O(R^2+WxH)
Space: O(WxH) O(R) for rooms O(k) for rocks

Gameplay Runtime:

- Movement:
    - Each action O(1)
- Rock breaking:
    - Check tile in front and update O(1)
    - If contains ladder, update to ladder O(1)
- HUD:
    - Reading mouse pos and tile lookup O(1) per frame
- Descending Ladder:
    - Triggers full world gen - same world gen cost above

Saving/Loading:

- Saving:
    - Writing command history to string len K O(k)
- Loading:
    - Reading saved string O(k)
    - Replaying k commands O(k)
    - Each command processed in O(1)
    - Total O(k)

Overall:

- Time per level:
    - O(R^2+WxH) for world generation
        - + O(k) for replaying
- Space per level:
    - O(WxH) for tiles
    - O(R) for rooms
    - O(k) for rocks
    - Total O(WxH)

Overall relatively manageable since fixed W and H constraints that are reasonably small

Question

What questions have come up while brainstorming your design? Consider asking them in assignment section or office hours!

Open Questions

Rock density: How to ensure reasonable sparsity of rock placement, maybe block creation inside hallways?

Rock breaking cost: What impact does this have on player, if any?

Level progression: does incrementing levels have any impact on gameplay?

Seed Generation: baseSeed + levelNum or RNG

Combat/progression: How to provide more meaningful gameplay to incentivize player progression

Closed Questions

Diagram

Draw a diagram summarizing your design. The diagram should capture the most important points about your design.

Example

If you need a guide on what to draw for the diagram, explain what happens during world generation and/or the core game loop. Show how user inputs affect the state of the world, and how that new world state is displayed to users. Draw out the flow of methods called and the data structures that are passed around.