Crystal Caves Programming Challenge

Year 12 Programming & Computational Thinking

The Setting

You are exploring the Crystal Caves.
The cave is represented as a grid. Each cell contains a single character such as a wall, empty space, crystal, player, trap, or enemy.

You will build a simple game that allows a player to move through the cave, collect crystals, avoid hazards, and eventually navigate more complex situations.

No dictionaries may be used. All data must be stored and processed using lists and 2D lists.

Tier 1 – Grid Basics and Movement

1. Display a cave grid on screen in a readable format. (Code in Teams)
2. Allow the player to move one square at a time using keyboard input.
3. Prevent the player from walking through walls.
4. Update the grid each turn to show the new player position.
5. End the turn cleanly when the user chooses to exit.

Focus skills:
Reading a 2D list, locating elements, checking boundaries, updating lists.

Tier 2 – Collecting Crystals

6. Add crystals to the cave layout.
7. Detect when the player moves onto a crystal.
8. Replace the crystal with the player symbol and increase the score.
9. Track how many crystals remain in the level.
10. End the level when all crystals are collected.

Focus skills:
State changes, counters, win conditions, iteration over grid data.

Tier 3 – Hazards and Simple Enemies

11. Add traps. If the player steps on a trap, the game ends immediately.
12. Add an enemy that takes a random valid step after each player move.
13. Prevent the enemy from walking into walls.
14. If the enemy reaches the player's position at any time, the game ends.
15. Make sure the grid updates enemies and player cleanly each turn.

Focus skills:
Simultaneous state management, turn sequencing, random movement, conditional logic.

Tier 4 – Multiple Levels and Game Structure

16. Create a list that stores multiple cave layouts.
17. Allow the player to choose a level by entering a number.
18. Track the number of crystals required for each level in a separate list.
19. Track best scores per level using a parallel list.
20. After completing a level, display an updated level summary showing scores.

Focus skills:
Parallel lists, selecting and loading data structures, designing a simple menu system.

Tier 5 – Advanced Mechanics (Extension)

Choose one or more of the following to extend the challenge.

A. Portals

21. Add portal entrances and exits.
22. Moving onto a portal instantly moves the player to the corresponding exit.
23. Handle cases where there are multiple portal pairs.
24. Clearly update the grid so players can see where they reappeared.

B. Additional Enemy Behaviours

25. Create an enemy that always tries to move closer to the player.
26. Create another that moves in a fixed pattern.
27. Ensure each enemy moves correctly within the rules.
28. Prevent enemies from moving on top of each other unless intended.

C. Hint System

29. When the player presses a key, the game suggests one useful move.
30. The hint system should consider crystals, traps, and enemies.
31. Explain your approach clearly in comments or a separate note.

Focus skills:
Algorithmic reasoning, multi-step decision-making, heuristics, complex grid analysis.

Tier 6 – Final Design Task

32. Design a new rule, mechanic, or challenge for the Crystal Caves.
33. Document the rule clearly.
34. Explain what changes would need to be made to the grid, player logic, and turn structure.
35. Implement the mechanic if time allows.

Focus skills:
Abstraction, system design, creative extension.

APPENDIX: Scaffold Code Snippets

Tier 1 – Movement and Grid Basics

Finding the player

```
# Returns (row, col)
def find_player(cave):
    for r in range(len(cave)):
        for c in range(len(cave[r])):
            if cave[r][c] == "P":
                return r, c
```

Movement offsets

```
# Example mapping from input to row/col change
moves = {
    "w": (-1, 0),
    "s": (1, 0),
    "a": (0, -1),
    "d": (0, 1)
}
```

Checking a move

```
def can_move_to(cave, r, c):
    if r < 0 or r >= len(cave):
        return False
    if c < 0 or c >= len(cave[r]):
        return False
    return cave[r][c] != "#"
```

Updating the grid

```
def move_player(cave, old_r, old_c, new_r, new_c):
    cave[old_r][old_c] = "."
    cave[new_r][new_c] = "P"
```

Tier 2 – Collecting Crystals

Detecting a crystal

```
if cave[new_r][new_c] == "C":
    score = score + 1
```

Counting remaining crystals

```
def count_crystals(cave):
    total = 0
    for row in cave:
        for cell in row:
            if cell == "C":
                total += 1
    return total
```

Tier 3 – Hazards and Enemies

Enemy random movement

```
import random


def move_enemy(cave):
    er, ec = find_enemy(cave)       # you write find_enemy()
    possible = []

    # Check all four directions
    for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
        nr, nc = er + dr, ec + dc
        if can_move_to(cave, nr, nc):
            possible.append((nr, nc))

    if len(possible) > 0:
        nr, nc = random.choice(possible)
        # Update grid
        cave[er][ec] = "."
        cave[nr][nc] = "E"
```

Collision with the player

```
if cave[nr][nc] == "P":
    # game over
```

Trap detection

```
if cave[new_r][new_c] == "T":
    # game over
```

Tier 4 – Multiple Levels

Level list structure

```
levels = [level1, level2, level3]   # each level is a 2D list
crystals_required = [3, 5, 7]       # parallel list
best_scores = [0, 0, 0]             # parallel list
```

Loading a level

```
def load_level(index):
    # Make a *deep copy* so the original isn't modified
    import copy
    return copy.deepcopy(levels[index])
```

Tier 5 – Advanced Mechanics

A. Portal scanning

```
def find_all(cave, symbol):
    positions = []
    for r in range(len(cave)):
        for c in range(len(cave[r])):
            if cave[r][c] == symbol:
                positions.append((r, c))
    return positions
```

Teleporting

```
if cave[new_r][new_c] == "O":
    exits = find_all(cave, "o")
    if len(exits) > 0:
        new_r, new_c = exits[0]   # simple version
```

B. “Chasing” enemy logic (Manhattan distance)

(Very light scaffolding; students complete the logic.)

```
def chase_step(er, ec, pr, pc):
    # choose direction that reduces distance
    # pseudocode outline:
    # if pr < er: try move up
    # elif pr > er: try move down
    # else if pc < ec: try left
    # else try right
    pass
```

C. Hint system skeleton

```
def give_hint(cave):
    # example: look for nearest crystal
    crystal_positions = find_all(cave, "C")

    # compute distances
    # choose best crystal
    # suggest direction

    return "Try moving north."   # placeholder
```

Tier 6 – Extending the Game

Adding a new symbol

```
# For example, a key 'K'
# Students must extend movement rules, win conditions, etc.
```

Hooking new rules into main loop

```
while True:
    display(cave)
    command = input("Move: ")

    # Player movement
    # Enemy movement
    # New mechanic processing here
```