CSC 222: Computer Programming II

Spring 2004

Structuring data & class design

- Hunt the Wumpus game
- Cave class
- CaveMaze class

Hunt the Wumpus: OO design

first step: identify the object and their properties/behaviors

Cave: cave number, name, contents, visited?, adjacent caves

get cave number	get cave name
get cave contents	change cave contents
get adjacent cave	determine if visited before
mark as visited	

CaveMaze: list of caves, # of wumpi, # of grades, current location, alive?

move down a tunnel to an adjacent cave
toss a grenade down a tunnel to an adjacent cave
show info about current location
determine how many grenades remaining
determine how many wumpi remaining
determine if alive

Class prototypes

as before: can design the classes based on behavior before having to decide on data structures

```
enum CaveContents {EMPTY, WUMPUS, PIT, BATS};

class Cave
{
  public:
   Cave(int num, string name, int left, int straight, int right);
    void SetContents(CaveContents c);
    void MarkAsVisited();
    CaveContents GetContents() const;
    bool HasVisited() const;
    int GetCaveNum() const;
    string GetCaveName() const;
    int GetAdjacent(int tunnel) const;
  private:
    // PRIVATE DATA FIELDS AND/OR MEMBER FUNCTIONS
};
```

```
class CaveMaze
{
  public:
    CaveMaze(string filename);
    void Move(int tunnel);
    void Toss(int tunnel);
    void ShowLocation() const;
    bool StillAlive() const;
    bool StillWumpi() const;
  private:
    // PRIVATE DATA FIELDS AND/OR MEMBER FUNCTIONS
};
```


Hunt the Wumpus program

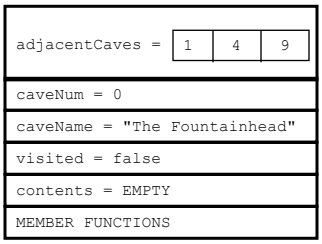
note: user input is crude

- what if user enters action other than 't' or 'm'?
- what if target < 1 or > 3?

```
#include <iostream>
#include <cctype>
#include <string>
#include "CaveMaze.h"
using namespace std;

const string MAZE_FILE = "caves.dat";

void DisplayIntro();

int main()
{
    CaveMaze maze(MAZE_FILE);

    DisplayIntro();
    while (maze.StillAlive() && maze.StillWumpi()) {
        maze.ShowLocation();

        char action;
        int target;
        cin >> action >> target;

        if (action == 't') {
            maze.Toss(target);
        }
        else {
            maze.Move(target);
        }
    }

    cout << endl << "GAME OVER" << endl;

    return 0;
}
```

Cave class

```
enum CaveContents {EMPTY, WUMPUS, PIT, BATS};

class Cave
{
  public:
    Cave(int num=-1, string name="", int left=-1, int straight=-1, int right=-1);
    void SetContents(CaveContents c);
    void MarkAsVisited();
    CaveContents GetContents() const;
    bool HasVisited() const;
    int GetCaveNum() const;
    string GetCaveName() const;
    int GetAdjacent(int tunnel) const;
  private:
    vector<int> adjacentCaves;
    int caveNum;
    string caveName;
    bool visited;
    CaveContents contents;
};
```



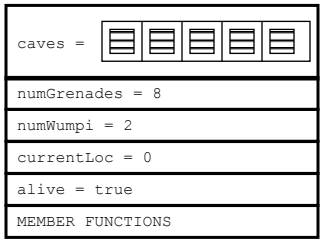
a Cave object

e.g., `Cave start(0, "The Fountainhead", 1, 4, 9);`

```
start.HasVisited()  → false      start.GetContents()  → EMPTY
start.GetCaveNum()  → 0          start.GetCaveName()  → "unknown"
start.MarkAsVisited()            start.GetCaveName()  → "The Fountainhead"
start.GetAdjacent(1) → 1         start.GetAdjacent(2) → 4
```


CaveMaze class

```
class CaveMaze
{
  public:
    CaveMaze(string filename);
    void Move(int tunnel);
    void Toss(int tunnel);
    void ShowLocation() const;
    bool StillAlive() const;
    bool StillWumpi() const;
  private:
    vector<Cave> caves;
    int numGrenades;
    int numWumpi;
    int currentLoc;
    bool alive;
    int FindEmpty() const;
    void MoveAdjacentWumpi();
    bool AdjacentTo(CaveContents room) const;
};
```



a CaveMaze object

to navigate the maze, use currentLoc and vector of caves

`caves[currentLoc]`	→ Cave object where currently located
`caves[currentLoc].GetContents()`	→ contents of current location
`caves[currentLoc].GetAdjacent(2)`	→ cave number connected to middle tunnel

MazeClass constructor

```
CaveMaze::CaveMaze(string filename)
{
    ifstream myin(filename.c_str());
    assert(myin);

    int numCaves;
    myin >> numCaves;
    caves.resize(numCaves);

    string name;
    int num, left, straight, right;
    while (myin >> num >> left >> straight >> right && getline(myin, name)) {
        Cave c(num, name, left, straight, right);
        caves[num] = c;
    }

    currentLoc = 0;
    caves[currentLoc].MarkAsVisited();

    Die d(caves.size()/4);
    numWumpi = d.Roll();
    numGrenades = 4*numWumpi;

    caves[currentLoc].SetContents(WUMPUS);
    for (int i = 0; i < numWumpi; i++) {
          caves[FindEmpty()].SetContents(WUMPUS);
    }
    caves[FindEmpty()].SetContents(BATS);
    caves[FindEmpty()].SetContents(PIT);
    caves[currentLoc].SetContents(EMPTY);
}
```

open cave data file

read number of caves & resize vector to fit

repeatedly read line of cave info, create Cave object, and put in vector

set initial location in maze

pick number of wumpi (& grenades)

place wumpi, bats & pit at random locations in the maze (uses FindEmpty to find location)

Cave data file (caves.dat)

```
20
 0  1  4  9The Fountainhead
 1  0  2  5The Rumpus Room
 2  1  3  6Buford's Folly
 3  2  4  7The Hall of Kings
 4  0  3 14The Silver Mirror
 5  1  9 11The Gallimaufry
 6  2  7 12The Den of Iniquity
 7  3  6  8The Findledelve
 8  7  3 13The Page of the Deniers
 9  0  5 10The Final Tally
10  9 11 14Ess four
11  5 10 12The Trillion
12  6 11 13The Scrofula
13  8 12 18Ephemeron
14  4 10 15Shelob's Lair
15 15 16 19The Lost Caverns of the Wyrm
16 15 17 19The Lost Caverns of the Wyrm
17 16 17 18The Lost Caverns of the Wyrm
18 13 17 19The Lost Caverns of the Wyrm
19 15 16 17The Lost Caverns of the Wyrm
```

first, number of caves

then 1 line per cave:

- cave number
- adjacent cave numbers
- cave name

note: Lost Caverns are simulated using unidirectional links

Move member function

```
void CaveMaze::Move(int tunnel)
// Results: moves to the cave connected by the specified tunnel
//          if new cave contains WUMPUS or PIT, then killed
//          if new cave contains BATS, then moved at random
{
    currentLoc = caves[currentLoc].GetAdjacent(tunnel);
    caves[currentLoc].MarkAsVisited();

    CaveContents room = caves[currentLoc].GetContents();
    if (room == WUMPUS) {
          cout << "You have entered the Wumpus' lair... CHOMP CHOMP CHOMP" << endl;
          alive = false;
    }
    else if (room == PIT) {
          cout << "Look out for the bottomless pit... AAAAAiiiiiiyyyeeeeeeee" << endl;
          alive = false;
    }
    else if (room == BATS) {
          cout << "A swarm of giant bats picks you up and flies away." << endl;
          currentLoc = FindEmpty();
          caves[currentLoc].MarkAsVisited();
    }
}
```

update currentLoc and mark new location as visited

check the contents of the new location to see if killed or moved – if moved, use FindEmpty to pick destination

HW3

for HW3, you are to complete the Hunt the Wumpus code

- www.creighton.edu/~davereed/csc222/Code/wumpus.cpp
- www.creighton.edu/~davereed/csc222/Code/Cave.h
- www.creighton.edu/~davereed/csc222/Code/Cave.cpp
- www.creighton.edu/~davereed/csc222/Code/CaveMaze.h
- www.creighton.edu/~davereed/csc222/Code/CaveMaze.cpp
- www.creighton.edu/~davereed/csc222/Code/caves.dat
- www.creighton.edu/~davereed/csc222/Code/Die.h
- www.creighton.edu/~davereed/csc222/Code/Die.cpp

you must complete the member functions for the `Cave` class in order to make the program work.

also, make user input more robust (ignore unrecognized commands)