CS 162C++
Dungeon Crawl Suggestions

Here are some suggestions for the Dungeon Crawl Game. This document is similar in organization to the one for Lab 1 (Tic-Tac-Toe).

Starting the program

When you are given a problem like this to solve, many people immediately jump into their IDE and start writing code. A better approach is to sit down and think about how the program should function and then build it incrementally doing one step at a time.

As with Tic-Tac-Toe, this is a game and has the same basic structure. The only major change is that there is only one player, so there is no need to keep track of which player is currently moving or to swap player. This first version of the program has comments for most of the functionality. The only function declared in this version is repeat();

```
#include <iostream>

#include <cstdlib> // for random

#include <ctime>   // for time

using namespace std;

// global constants
const int MAX_ROW;
const int MAX_COL;
const char SPACE = '.';
const char TRAP = '#';
const char PLAY = '@';
const char CASH = '$';

// function declarations
bool repeat(); // do you want to play again?

int main()
{
   // initialize random
   srand(static_cast<int>(time(NULL)));

   // play again loop
   do {

      // define variables
      bool win, lose;

      // initialize dungeon
      // display instructions

      // game loop
      do {

         // display dungeon
         // get move
         // move monsters if you have any

         win = checkMove(dungeon, cash);
         if (not win)
            lose = checkMove(dungeon, trap);
```

```
            // check for monster kill if monsters exist
            if (not win and not lose)
               lose = checkMove(dungeon, monster);

            if (not win and not lose)
               // update dungeon

         } while (not win and not lose);

      } while (repeat());

      return 0;
   }
```

Now, think of what functions you want to use and declare them. For the first pass, any that you are not sure of, you can leave as void return type with no parameters. Add those details later as you find how the function will be used.

```
#include <iostream>
#include <cstdlib> // for random
#include <ctime>   // for time

using namespace std;

// global constants
const int MAX_ROW;
const int MAX_COL;
const char SPACE = '.';
const char TRAP = '#';
const char PLAY = '@';
const char CASH = '$';
const char UP = 'U';
const char DOWN = 'D';
const char LEFT = 'L';
const char RIGHT = 'R';

// function declarations

void displayInstructions(); // displays the instructions
void initDungeon(); // sets up a dungeon with traps and treasure
void displayDungeon(); // display the current dungeon state
char getMove(); // gets a valid move (L,R,U,D)
bool checkMove(); // sees if move is onto checkCode object
void updateDungeon(); // update the dungeon for next move
bool repeat(); // do you want to play again?

int main()
{

   // initialize random
   srand(static_cast<int>(time(NULL)));

   // play again loop
   do {

      // define variables
      bool win, lose;
      char dungeon [MAX_ROW][MAX_COL];
      int numTraps = 6; // using fixed number to start with

      // display instructions
      displayInstructions();

      // initialize dungeon
      initDungeon();


      // game loop
      do {

         // display dungeon
         displayDungeon();
```

```
            // get move
            char move = getMove();


            // move monsters if you have any
            // no monsters in this first version

            // check for win or lose
            // win if steps on treasure
            win = checkMove();

            // if did not win, see if died from a trap or monster
            lose = (!win and checkMove(/* trap */)) and checkMove(/* Monster */);



            // now that we know it was safe move, update the world
            // for the next turn
            if (not win and not lose)
                updateDungeon();

        } while (not win and not lose);

    } while (repeat());

    return 0;
}

// function definitions

// displays the instructions
void displayInstructions()
{
    cout << "Displaying Instructions" << endl;
}

// sets up dungeon with traps and treasure
void initDungeon()
{
    cout << "Initializing the dungeon" << endl;
}

// display current dungeon state
void displayDungeon()
{
    cout << "Displaying the dungeon" << endl;
}

// get a valid move (L,R,U,D)
char getMove()
{
    cout << "Getting a move, returning UP for debug" << endl;
    return UP;
}

// sees if move is onto checkCode object
bool checkMove()
{
    cout << "Checking move for stepping on " << "checkCode" << endl;
    cout << "  Return true for debug purposes" << endl;

    return true;
}

// update the dungeon for next move
void updateDungeon()
{
    cout << "Updating Dungeon for next move"  << endl;
}

// do you want to play again?
bool repeat()
{
    cout << "Checking for play again, returning false for debug" << endl;
```

```
    return false;
}
```

Now you can continue to develop the program one function at a time. Each time make sure that it compiles and runs without problems. I did it in the following order:

1. display instructions
2. initialize dungeon
3. display dungeon
4. get move
5. check move for each of the options
6. repeat

By using this order, each step built upon the previous and it kept the game compiling and showing the latest change

Displaying the dungeon

Displaying the board should be simpler than in TTT, since you just output the dungeon array.

Initializing the dungeon

You create the dungeon in main, so you can pass it around as need be.

The initialization code sets all locations to a blank space, then randomly sets as many traps as you want, then randomly locates one treasure and finally randomly places the player.

If you have monsters, you set them up also. You can make sure that each new item is on its own space or you can allow later placed items to overwrite earlier placed ones. Your choice of how you want to proceed.

Passing the dungeon

You pass the dungeon around as a 2D array.

Moves

The possible moves are Up, Down, Left, and Right.

Consider the following dungeon

. . . @ . . .

. . . . . # .

. . # . . . .

. . . $ . . .

If you move **Down**, then it should be

. . . . . .

. . . @ . # .

. . # . . . .

. . . $ . . .

To determine how to make a move, consider what the change in row and column was from the move **Down**. Do the same for each of the other three moves. Make sure that you do not allow illegal moves and that you do not allow the player to move outside the dungeon.