```cpp
#include <iostream>
#include <string>
#include <ctime>
#include <vector>
#include <cmath>

using namespace std;

const int LEVEL_WIDTH = 60;
const int LEVEL_HEIGHT = 15;
const int NUM_ITEMS = 5;

const int ITEM_FOOD = 1;
const int ITEM_GOLD = 2;
const int ITEM_TRAP = 3;

const int TILE_WALL = 1;

const int START_HEALTH = 100;
const int START_FOOD = 20;
const int START_GOLD = 0;

int LevelData[LEVEL_WIDTH][LEVEL_HEIGHT];

int charPos[] = {0,0};
int playerAttack = 10;
int charStatus[3];

int itemsLocations[NUM_ITEMS][3];

bool IsPlaying;

class Ogre
{
public:
        Ogre();
        ~Ogre();
        int x;
        int y;
        void update();
        void debugPrint();
        void takeDamage(int);
        static int START_HEALTH;
        static int ATTACK_STR;
private:
        int health;
        int hunger;
        void reset();
};

int Ogre::START_HEALTH = 30;
int Ogre::ATTACK_STR = 2;

Ogre::Ogre()
{
        reset();
        // debugPrint();
}
```

```
Ogre::~Ogre()
{
        cout << "Ogre killed!\n";
}

void Ogre::update()
{
        int skipTurn = rand() % 4;
        if (skipTurn == 0) return;

        int direction = rand() % 4;

        int newX = x;
        int newY = y;

        hunger++;

        // random motion
        switch (direction)
        {
                case 0:
                        newX++;
                        break;
                case 1:
                        newY++;
                        break;
                case 2:
                        newX--;
                        break;
                case 3:
                        newY--;
                        break;
        }

        // get distance to player, used for hunting and running away
        int diffX = charPos[0] - x;
        int diffY = charPos[1] - y;

        // if damaged, move away player
        if (health < 20)
        {
                if (diffY == 0 || rand() % 2 == 0)
                {
                        if (diffX < 0) newX = x + 1;
                        else newX = x - 1;
                }
                else
                {
                        if (diffY < 0) newY = y + 1;
                        else newY = y - 1;
                }

        }
        else if (hunger > 10) // if healthy, check hunger and, if hungry, move
towards the player
        {
                if (diffY == 0 || rand() % 2 == 0)
                {
                        if (diffX < 0) newX = x-1;
```

```
                        else newX = x+1;
                }
                else
                {
                        if (diffY < 0) newY = y - 1;
                        else newY = y + 1;
                }
        }

        if (newX == charPos[0] && newY == charPos[1])
        {
                cout << "The Ogre strikes!\n";
                charStatus[0] -= Ogre::ATTACK_STR;
                return;
        }

        if (LevelData[newX][newY] == TILE_WALL)
        {
                return;
        }

        // the ogre moved to a new, unoccupied space, so update its position
        x = newX;
        y = newY;

}

void Ogre::debugPrint()
{
        cout << "Ogre at " << x << "," << y << " health: " << health << " hunger: "
<< hunger;
        if (health < 20) cout << " running away";
        if (hunger > 10) cout << " hunting";
        cout << "\n";
}

void Ogre::takeDamage(int damage)
{
        health -= damage;
        if (health <= 0)
        {
                int goldFound = rand() % 10 + 2;
                cout << "You kill the ogre and find " << goldFound << " pieces of
gold!\n";
                charStatus[2] += goldFound;
                reset();
        }
}

void Ogre::reset()
{
        x = (rand() % (LEVEL_WIDTH - 2)) + 1;
        y = (rand() % (LEVEL_HEIGHT - 2)) + 1;

        hunger = rand() % 5;
        health = Ogre::START_HEALTH;
}
```

```
void PrintPlayerStatus()
{
	cout << "\n1: move left, 2: move up, 3: move down, 4: move right, 5: wait,
6: quit\n";
	cout << "Current status: ";
	cout << "Health: " << charStatus[0];
	cout << " Food: " << charStatus[1];
	cout << " Gold: " << charStatus[2];
	cout << "\n";
}

void PrintLevel(vector<Ogre*> &ogres)
{
	for (int i = 0; i < LEVEL_HEIGHT; i++)
	{
		for (int j = 0; j < LEVEL_WIDTH; j++)
		{
			if (j == charPos[0] && i == charPos[1])
			{
				cout << "@";
			}
			else
			{
				bool ogreHere = false;

				for (int k=0; k<ogres.size(); k++)
				{
					if (ogres[k]->x == j && ogres[k]->y == i)
					{
						ogreHere = true;
					}
				}

				if (ogreHere)
				{
					cout << "0";
				}
				else
				{
					switch (LevelData[j][i])
					{
					case 1:
						cout << "*";
						break;
					case 3:
						cout << "%";
						break;
					default:
						cout << ".";
					}
				}
			}

		}
		cout << endl;
	}

}
```

```
void InitLevel()
{
	srand(time(NULL));

	charStatus[0] = START_HEALTH;
	charStatus[1] = START_FOOD;
	charStatus[2] = START_GOLD;

	IsPlaying = true;

	for (int i = 0; i < LEVEL_HEIGHT; i++)
	{
		for (int j = 0; j < LEVEL_WIDTH; j++)
		{
			LevelData[j][i] = 0;
			if (j == 0 || j == (LEVEL_WIDTH - 1) || i == 0 || i ==
(LEVEL_HEIGHT - 1))
			{
				LevelData[j][i] = 1;
			}
		}
		cout << endl;
	}

}


void ProcessInput(int i, vector<Ogre*> &ogres)
{
	int newX = charPos[0];
	int newY = charPos[1];

	switch (i)
	{
		case 1:
			newX -= 1;
			break;
		case 4:
			newX += 1;
			break;
		case 3:
			newY += 1;
			break;
		case 2:
			newY -= 1;
			break;
		case 5:
			break;
		case 6:
			IsPlaying = false;
			break;
	}

	// see if we're in a wall
	if (LevelData[newX][newY] == TILE_WALL) return;

	// see if we're hitting an ogre
	for (int i=0; i<ogres.size(); i++)
```

```
        {
                if (newX == ogres[i]->x && newY == ogres[i]->y)
                {
                        ogres[i]->takeDamage(playerAttack);
                        cout << "You hit the ogre for " << playerAttack << "
damage!\n";
                        return;
                }
        }

        // we moved to an unoccupied space, so update our position
        charPos[0] = newX;
        charPos[1] = newY;
}

int main()
{
        charPos[0] = floor(LEVEL_WIDTH/2);
        charPos[1] = floor(LEVEL_HEIGHT/2);

        vector<Ogre*> enemies;

        int numOgres = 5;

        for (int i=0; i < numOgres; i++)
        {
                enemies.push_back(new Ogre);
        }

        int input;
        InitLevel();

        while (IsPlaying)
        {
                PrintPlayerStatus();
                PrintLevel(enemies);
                cin >> input;
                ProcessInput(input, enemies);

                for (int i=0; i < numOgres; i++)
                {
                        enemies[i]->update();
                }
        }

        for (int i=0; i < numOgres; i++)
        {
                delete enemies[i];
        }

        return 0;
}
```