# Escalation Programming Test

## Instructions

Write a program that accomplishes the tasks defined below. You have three hours to complete the task. At the end of three hours, send your project and source file(s) to jflanagan@escalationstudios.com.

### Guidelines

- Code must be written in C++
- You may use the STL or other libraries provided with your compiler.
- Your final submission should include the source files in a single zip file.
- Like most take-home tests, this one is open book. Feel free to use any resources you have available except for other people.

### What we are looking for

- Correctness in solving the problem
- Elegance and quality of code
- Robustness of code
- Maintainability and extensibility of code

### Advice

Don't get bogged down on one section of the problem. Demonstrate what you know, attempt to figure out what you don't know and try to solve as much of the problem as you can. You may not be able to finish the test in the time allocated.

The problem is contrived and simple so as to fit within a small time frame. That doesn't mean we want to see a piece of code that isn't within the quality of a larger scope project. You should write as if your code were going to be the backbone for a larger, more sophisticated system.

# Monster Wars

You are working on a 2D game called Monster Wars. Your designers are able to place entities all over the map. These entities can be various things, such as monsters, traps, and chests. Each entity has some properties. In other words, monsters, traps, and chests are all specific types of entities.

## You are given

- A "Map Data" file (with .bin file extension)
- A "Map Data File Format" document which details the format of a .bin file.
- An "Entity Spec Sheet" document which details the specifics of each type of entity

## Your program should

1. Read in the map data from the file, parsing it per file format specifications
2. Create a simple representation of the entities in memory as defined in the spec.
3. Each entity must be able to output a string representation of itself.
4. Output a list of monsters that are overlapping with other monsters.
5. Ask the user for a monster on which to do an FOV test (by unique ID) and output a list of entities that the monster can see. This step should allow testing for multiple monsters until the user decides to stop testing.

## What information about entities should be printed out?

Entities all have a unique ID and a type that should be printed out in human-readable format. You should also print the location and/or forward vector of the entity in (X, Y) format.

## When are monsters overlapping?

Two monsters are overlapping when the area of intersection of their physical geometry is greater than zero. Monsters that are touching are not overlapping. All monsters have circular collision.

## When are entities visible to monsters?

A monster can see an entity when the entity's position is within the 2d arc defined by the forward vector of the monster plus or minus half the FOV given by the monster section of the Entity Spec Sheet. For simplicity's sake only the point position needs to be taken into account, not the entity's radius. Assume monsters have infinite sight distance.

# Map Data File Format

**NOTE:** All data structures are 32 byte aligned.

## File Format

| File Offset (in bytes) | Value Type | Value Description |
|---|---|---|
| 0 | Unsigned Integer (32 bits) | Number of entities |
| 4 | Entity Information (see below) | Entity 0 |
| 4 + 32 | Entity Information | Entity 1 |
| ... | ... | ... |
| 4 + (32 * N) | Entity Information | Entity N |

## Entity Information

| Offset (in bytes) | Value Type | Value Description |
|---|---|---|
| 0 | Unsigned Short (16 bits) | Unique ID |
| 2 | Unsigned Short (16 bits) | Entity Type ID (See **Entity Spec Sheet**) |
| 4 | Single-precision float (32 bits) | Position X Coordinate |
| 8 | Single-precision float (32 bits) | Position Y Coordinate |
| 12 | Single-precision float (32 bits) | Forward Normal X Component |
| 16 | Single-precision float (32 bits) | Forward Normal Y Component |

## Entity Spec Sheet

### Items

Each item has only basic entity information plus a collision radius.

| Entity Type ID | Collision Radius |
|---|---|
| 0 - Health Pickup | 10 |
| 1 - Chest | 20 |
| 2 - Trap | 64 |

### Monsters

Each monster has the basic entity information, a collision radius and a field of view.

| Entity Type ID | Collision Radius | Field of View (in degrees) |
|---|---|---|
| 3 - Troll | 12 | 45 |
| 4 - Imp | 10 | 65 |
| 5 - Ogre | 25 | 90 |

## Explanation of Terms

**Unique ID:** Each entity in the map has a unique reference ID. This is the ID the user will enter for a monster to check to see what entities it can see.

**Entity Type ID:** An identifier that maps an ID in the map file to a specific type of entity as defined in the Entity Spec Sheet.

**Forward Normal:** A normalized (unit length) vector that points in the direction the entity is facing.

**Field of View:** The total arc that the monster can see. The direction of the arc is determined by the Forward Normal.

**Collision Radius:** Entities have circular collision that's defined to be all points from their Position to the radius defined by their Collision Radius.