Cairo University
Faculty of Engineering
CMP 102 & CMP N102

Fall 2016

Data Structures and Algorithms

Project Requirements

Project Phases

Project Phase	%	Due Date
Phase 1	15%	Week 9
Phase 2	85%	Week 14

Objectives

By the end of this project, the student should be able to:

- Understand unstructured, natural language problem description and derive an appropriate design.
- Intuitively modularize a design into independent components and divide these components among team members.
- Build and use data structures to implement the proposed design.
- Write a complete procedural C++ program that performs a non-trivial task.
- Use third-party code libraries as parts of the project

Introduction

Back to the Middle Ages assume you are the liege of a castle that is protected by 4 towers where every tower is required to protect a certain region (See Fig 1). Every day some enemies attack the castle and they want to destroy your towers. You need to use your programming skills and knowledge to data structures to write a ***simulation program*** of a game between your castle towers and enemies. You should simulate the war between the towers and the attacking enemies then calculate some statistics from this simulation.

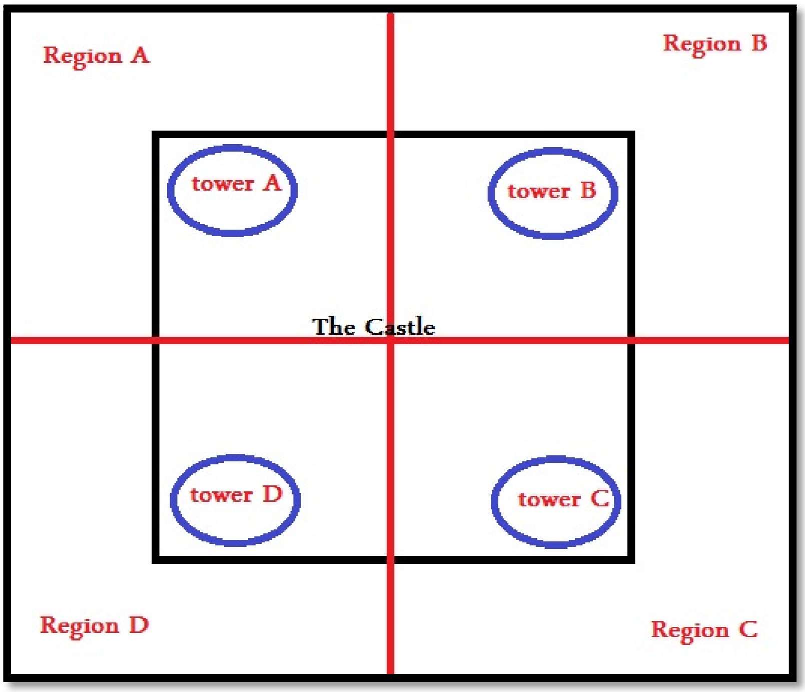


Figure 1 the castle

Problem description

Four towers are defending the castle. Each tower guards one region and can shoot only enemies in its region. Each tower has a starting health and can shoot at most N enemies at each time step.

Your system (your program) will receive a list of enemies as input. This list represents the scenario to be simulated. For each enemy the system will receive the following information:

- **Time stamp (Enemy Arrival Time):** When the enemy will appear.
- **Health:** The start health of the enemy.
- **Fire Power:** The shot hit power of the enemy.
- **Reload Period:** Time for an enemy to reload its weapon. During reload period, an enemy cannot fight but can move.
- **Type:** Three types of enemies : ***paver***, ***fighter*** and ***shielded fighter***
- **Region:** The attack region of the enemy.

Simulation Approach & Assumptions

You will use incremental time simulation. You will divide the time into discrete time steps of 1 time unit each and simulate the changes in the system in each time step.

Some Definitions and formulas

- **Enemy State:**
  At any time an enemy should be in one of three states: ***inactive*** (not arrived yet), ***active*** (described below) or ***killed*** (health = 0). Only active enemies can fight.

- **Active Enemy** is an enemy with Arrival Time ≤ current time step & Health > 0.
  At each time step, each tower should choose ***N*** active enemies to shoot (N is given in the input file).

- **Enemy distance** is the ***horizontal*** distance between the enemy and the tower.

- **Paver Enemy**
  All enemies can approach the castle **one meter** at every time step ***only if the next meter is paved***. At the start of the simulation, the last **30 meters** in the road to the castle are not paved. Only the paver enemies can enter the unpaved distance to pave it so that enemies of other types can enter this paved distance in the **next** time steps.

- **Damage to the tower by certain enemy**

$$\textbf{\textit{DE = Damage (Enemy}} \rightarrow \textbf{\textit{Tower) =}} \frac{\boldsymbol{k}}{\boldsymbol{Enemy\_distance}} * Enemy\_fire\_power$$

Note: If enemy is not allowed to fire at current step, it will not cause any damage to the tower.

- **Damage to a certain enemy by the tower**

$$\textbf{\textit{DT = Damage (Tower}} \rightarrow \textbf{\textit{Enemy) =}} \frac{\mathbf{1}}{\boldsymbol{Enemy\_distance}} * Tower\_fire\_power * \frac{\mathbf{1}}{\boldsymbol{k}}$$

  Use k=2 for shielded enemies and k=1 for other enemies

Note:

A paver does not shoot the towers and its “Fire Power” represents the number of meters it can pave at each ***allowed*** attack time step. Hence, Formula Damage(Enemy→Tower) described above is not applicable for pavers. However, Formula Damage(Tower → Enemy) is applicable.

- **Enemy Priority**
  As mentioned, a tower can shoot N enemies at each time step.
  - If there are no active shielded enemies, the tower picks the next active enemy to shoot based on its arrival time according to FCFS (First Come First Serve) criterion.
  - Active Shielded enemies have higher priority than other enemies and should be shot first regardless of their arrival time.
  - If there is more than one active shielded enemy, each of them is given a priority according to the next formula:

Priority (Shielded Enemy) =

$$\frac{\boldsymbol{Enemy\_Fire\_Power}}{\boldsymbol{Enemy\_Distance}} * C1 + \frac{\boldsymbol{C2}}{\boldsymbol{Enemy\_remaming\_time\ \_to\_shoot+1}} + Enemy\_health * C3$$

Where C1, C2 and C3 are three constants that are read from the input file.

- **Fight Delay (FD)**
  The time elapsed until an enemy is first shot by a tower
  $$FD = T_{first\_shot} - T_{arrival}$$
- **Kill Delay (KD)**
  The time elapsed between first time a tower shoots an enemy and its kill time
  $$KD = T_{enemy\_killed} - T_{first\_shot}$$
- **Fight Time (FT)**
  The total time an enemy stays alive until being killed
  $$FT = FD+KD = T_{enemy\_killed} - T_{arrival}$$

Assumptions

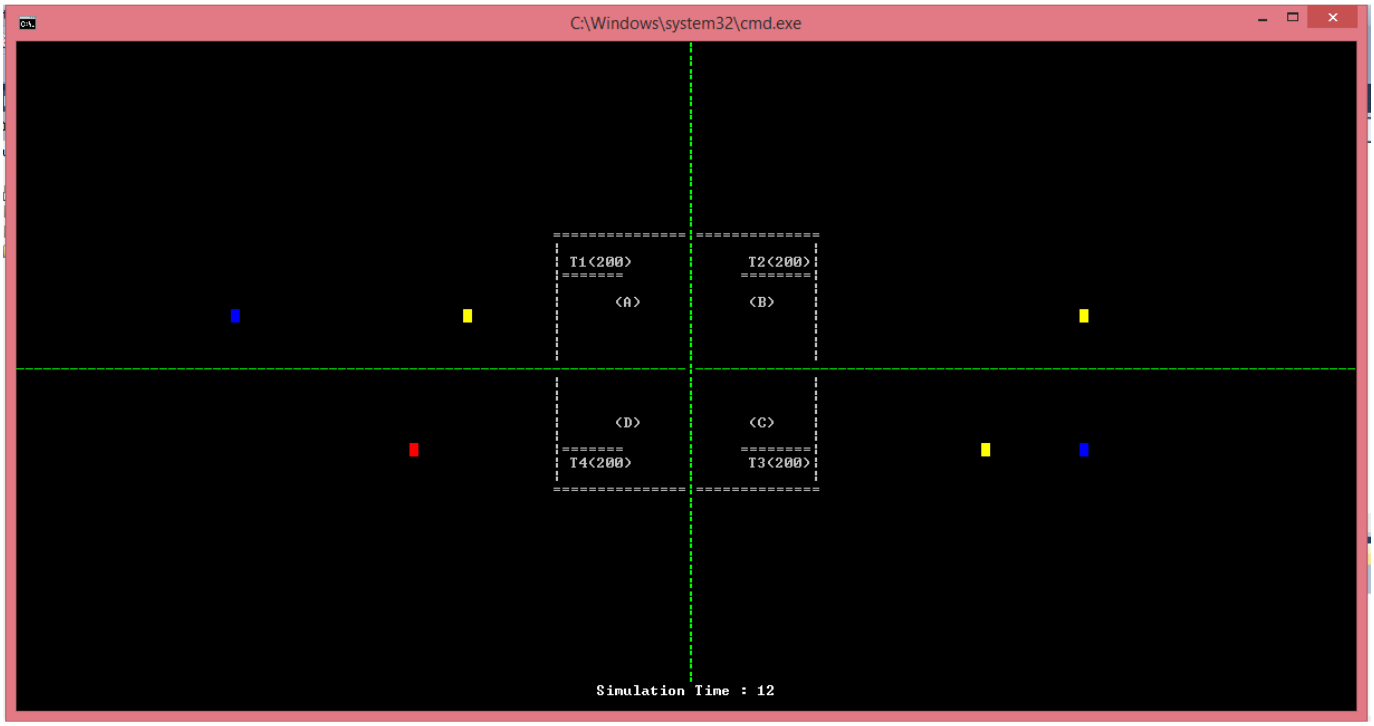
- Every tower can attack **only** enemies in its region.
- Every enemy can attack **only** the tower in its region.
- All enemies start at **60 meters** distance from the castle.
- Every tower can attack at most **N enemies** at each time step. N is read from the input file.
- The minimum possible distance for any enemy to approach is **2 meters**.
- The enemies can approach to the castle **one paved meter** at every **time step**. This includes paver enemy during its reload period.
- The game is “**win**” if all enemies are killed
- The game is “**loss**” if the all towers are destroyed.
- If a tower in a region is destroyed all enemies (current and incoming enemies) in that region should be transported to the next region. The next region means the adjacent region moving in the clockwise direction. (A → B → C → D → A).
- If an enemy is transported to the next region, it should be placed at the same distance it reached in the previous region. If such distance is not paved, it should be placed at the closest paved distance to the new region tower.

Program Interface

The program should read an integer from standard input (*cin*). If this integer is 1, the program runs in **interactive mode**. If it is 2, the program runs in **step-by-step mode**. Otherwise the program runs in **silent mode**.

Interactive mode allows you to monitor the attacking of enemies to the castle and active enemies as time goes on. At each time step, the program should provide output similar to that in the following figure (Figure 2) on the screen and pause for a user input until instructed to continue. At the bottom of the screen, the following information should be shown:

- Simulation Time Step
- For each region print:
    - total number of current enemies
    - number of enemies killed at last time step
    - total number of killed enemies from the beginning of simulation
    - unpaved distance to castle in this region



```
T1(200)            T2(200)

      (A)          (B)

      (D)          (C)

T4(200)            T3(200)

Simulation Time : 12
```

Figure 2 Program Output

Step-by-step mode is identical to interactive mode except that the program pauses for one second (instead of pausing for user input) then resumes automatically.

In **silent mode**, the program produces only an output file. It neither pauses nor provides screen output.

You are a code library (set of functions) for drawing the above interface and you **should** integrate it with your system.

File Formats

Your program should receive all information to be simulated from an input file and produces an output file that contains some information and statistics about the simulation. This section describes the format of both files and gives a sample for each.

The Input File

- First line contains three integers: **TH N TP**
  TH is the starting health of all towers; **N** is the maximum number of enemies a tower can attack at any time step and **TP** is the tower fire power.
- Second line contains **C1 C2 C3** the constants of equation "**Priority (Enemy)"**
- Then the input file contains many lines (one line for each enemy) of the format **S TY T H Pow Prd R**
  where **S** is a sequence number that identifies the enemy, **TY** is the enemy type, **T** is the enemy arrival time, **H** is the enemy health, **Pow** is the enemy fire power, **Prd** is the enemy reload period and **R** is the enemy region. The input lines are sorted by arrival time in ascending order.
- The last line in the input file should be
  -1
  which indicates the end of input file.
  The input file name must be "input.txt".

The Output File (No matter what mode of operation your program is running in, the output file should be produced)

The output file you are required to produce should contain **M** output line of the format **KTS S FD KD FT**
which means that the enemy identified by sequence number **S** is killed at time step **KTS** and its fight delay is **FD** and kill delay is **KD** and total fight time is **FT**. The output lines should be sorted by **KTS** in ascending order. If more than one enemy are killed at the same time step, **they should be ordered by FD**.

A line at the end of the file should indicate the total damage for each tower by the attacking enemies.
T1_Total_Damage T2_Total_Damage T3_Total_Damage T4_Total_Damage
Another line should indicate the remaining unpaved distance in each region.
R1_Distance R2_Distance R3_Distance R4_Distance

Another line for string of **"Game is WIN" or "Game is LOSS"** for the game.

Then the following statistics should be shown at the end of the file

1- In case of game "**win**"
   a. Total number of enemies
   b. Average "Fight Delay" and Average "Kill Delay"
2- In case of game "**loss**"
   a. Number of killed enemies
   b. Number of alive enemies (active and inactive)
   c. Average "Fight Delay" and Average "Kill Delay" for killed enemies only

Sample Input File

```
200  3     14
1    0.03  0.01
1    0     1     10    2     4     A
2    2     3     15    5     4     A
3    1     7     15    2     3     B
-1
```

The above file initializes the towers with health 200, each tower can attack at most 3 enemies at every time step with fire power is 14.

The second line means that constants C1=1, C2=0.03 and C3=0.01

Then enemies' details:

- An enemy of type=0 (paver) arrived at time step 1 in region A with Health =10 and Fire_power = 2 and Reload_period = 4.
- An enemy of type=2 (shielded fighter) arrived at time step 3 in region A with Health=15 and Fire_power = 5 and Reload_period=4
- An enemy of type=1 (fighter) arrived at time step 7 in region B with Health=15 and Fire_power = 2 and Reload_period=3.

Sample Output File

The following numbers are just for clarification and are not produced by actual calculations.

```
KTS   S     FD   KD    FT
 5    1     0     5    5
10    2     4    4     8
15    3     5    2     7
.
.
T1_Total_Damage   T2_Total_Damage   T3_Total_Damage   T4_Total_Damage
    33.5              12.5               55                200
R1_Distance            R2_Distance     R3_Distance         R4_Distance
    30                   30               25                 2
Game is WIN
Total Enemies          = 50
Average Fight Delay    = 4.5
Average Kill Delay     = 12.36
```

The second line in the above file indicates that enemy with sequence number 1 killed at time step=5 and it took FD=0 And KD= 5

Last four lines indicate you won the game, total enemies=50, average fight delay = 4.5, and average kill delay=12.36

Project Phases

Phase 1: Due date: Week 9

In this phase you should finish all simple functions that are not involved in fighting logic nor statistics calculation and collection.
The required parts to be finalized and delivered at this phase are:

1- Full declarations of Enemy, Tower, and Castle "***struct**s*".

2- The data structures that you will use to represent the lists of enemies

Important: Keep in mind that you are **NOT** selecting the DS that would work in phase1 only. You should choose the DS that would work for both phase1 and phase2.
Logically, any enemy should be in one of four states: ***inactive, active, high-priority active,*** or ***killed***.
So, when choosing the DS think about the following:

a. Do you need a separate list for each enemy state?
b. Which list type is much suitable to represent the lists taking into account operations needed for each list (e.g. insert, delete, shift)
c. Will you use one list for all regions or a separate list for each region and why?
d. How will you store the high priority enemies?
e. Do you need to store killed enemies? In a separate list or not? When should you get rid of them to save memory?

All the above data structures should be fully implemented at this phase

3- File loading function. The function that reads input file to:
a. Load Towers data
b. Load constants values
c. Create and populate inactive enemies list.

4- Simple Simulator function. This function should
a. Perform any needed initializations
b. Call file loading function
c. At ***each time step*** do the following
i. Move active enemies from inactive list to active list
ii. Pick at most 4 random active enemies to kill: Randomly pick two from high priority active enemies and randomly pick two from normal active enemies.
iii. Remove killed enemies from the list(s).
iv. **For each region**, print
1. Total number of active enemies and information of each one of them.
2. Total number of killed enemies and information of each one of them.
The killed enemies have to be printed ordered by enemy health.

Notes:
- No output files should be produced at this phase.
- No graphical output is needed at this phase.
- No fighting logic is needed at this phase.

Deliver a CD that contains Phase1 code and three sample input files.

Phase 2: Due date: Week 14

In this phase, you should extend code of phase 1 to build the full application and produce the final output file. Your application should support the different operation modes described in “Program Interface” section.

Phase 2 Deliverables:

Each team is required to deliver a **CD** that contains:

- A text file named ***ID.txt*** containing team members’ names, IDs, and emails.
- Program source code (.cpp and .h files, project file(s)) [Do not include executable files].
- Six Sample input files (test cases) and their output files.
- Write your team number on the back of the CD cover.
- A project document with 2 or more pages describing your solution modules and any clever or innovative alternatives you followed in implementing the solution

Project Evaluation

Evaluation Criteria

- **[5%] Successful Compilation:** Your program should compile successfully with zero errors and zero warnings. If you find a warning that you cannot get rid of, post a question on the project discussion forum.

- **[50%] Data Structure & Algorithm**: After all, this is what the course is about. You should be able to provide a concise description and a justification for: (1) the data structure(s) and algorithm(s) you used to solve the problem, (2) the complexity of the chosen algorithm, and (3) the logic of the program flow.

- **[20%] Modularity and Maintainability**: A ***modular*** code does not mix several program features within the same unit (module). For example, the code that does the core of the simulation process should be separate from the code that reads the input file which, in turn is separate from the code that implements the data structure. A ***maintainable*** code is the one whose modules are easily modified or extended without a severe effect on other modules.

- **[5%] Interface modes**: Your program should support the three interface modes described in the document. The existence of one mode does not compensate the absence of another.

- **[15%] Test Cases**: You should prepare comprehensive test cases (at least 6) that range from weak to strong castle and enemies (e.g. weak-castle-moderate-enemy case). Your program should be able to simulate different scenarios not just trivial ones.

- **[5%] Coding style**: How elegant and **consistent** is your coding style (indentation, naming convention ...etc)? How useful and sufficient are your comments?

  Note: Each team member will be evaluated individually.

Bonus Criteria (maximum 10%)

- **[2%] More modularity:** Divide program modules among multiple files. Each module should be implemented in at least one *.cpp* file and one *.h* file.
- **[3%] Enemies speed:** Handling enemies with different speed.
- **[5%] More enemy types:** Think about two more enemy types other than those given in the document.

Team Size: A team should consist of 3-4 students.

Appendix A - Game Interface code library

To help produce a nice output for your game, we are providing you with some helpful functions that you should use to "draw" the user interface of the game and other functions that you will use to draw towers/enemies at every time step. This appendix describes the given functions. Functions code can be found in *utility.h* and *utility.cpp* files.

Functions that you should use with no need to modify:

1. *void PrintMsg(char*msg);*
   This function prints any given message (text) on the screen.
   Inputs: msg ➔ a char pointer (or array of characters).
   Use this function to print the statistics at every time step
   E.g. **PrintMsg("Total current enemies A=5 B=4 C=7 D=2\n");**

2. *void DrawEnemies(enemy *enemies[ ],int size)*
   This function draws ALL active enemies in the game.
   Inputs: enemies [ ] ➔ an array of enemy pointers. ALL active enemies from ALL regions should be pointed by this array.
   size ➔ size of the enemies array (Total number of active enemies)

Note: No matter what list type you are using to hold enemies, you must pass the enemies to the above function as an array of enemy pointers. At every time step, you should update those pointers to point to the current active enemies then pass the pointers list.

3. *void DrawRegions(const castle & C);*
   This function partitions the game area into four regions (A, B, C, and D)
   Inputs: Castle ➔ Castle data used to identify screen center

4. *void DrawCastle(const castle & C, int SimulationTime);*
   This function draws the castle and its four towers
   Inputs: Castle ➔ a castle struct that contains info about castle and towers
   SimulationTime ➔ the current simulation time step.

Important Note: At each time step you need to: (1) clear the screen, (2) draw the castle, (3)draw the regions, (4) draw the enemies, and (5) print any required messages/statistics.

Functions you don't need to use and should not modify:

These functions are used by the above function. You should not modify these functions

1. *void gotoxy(int x, int y);* A function to set the position of cursor on the screen
2. *void SetWindow();* A function to set the command window length and height for the game specification
3. *void color(int thecolor);* A function to color the cmd text
4. *void DrawEnemy(const enemy& E, int Ypos=0);* A function to draw a single enemy using its distance from the castle. It is used by *DrawEnemies* function described above.

Also you may need to add more attributes to the structs declared in *utility.h* (enemy, tower, and castle)

Finally, a demo code (*demo.cpp*) is given just to test the above functions and show you how they can be used.