CMPUT 350 Assignment 2

Due: Wednesday Oct. 9, 22:00

Important: Read this text completely and start working on the assignment as soon as possible. Only then will you get an idea about how long it will take you to complete it.

If you worked on the assignment on your personal computer, copy your files to the undergraduate lab machine (e.g. uf13.cs.ualberta.ca) with scp and then test your code there as well before submitting it. Like the labs, we will be using this environment to test your code.

When done, submit your solution files on eClass:

```
README.txt World.cpp Marine.cpp Tank.cpp Experiments.txt Problem2.txt/pdf
```

Your submissions will be marked considering their correctness (syntax and semantics), documentation, efficiency, and consistent coding style. Add assert statements to check important pre/post-conditions.

Finally, make sure that you follow the course collaboration policy which is stated on the course webpage. File `README.txt` must be completed!

---

1. RTS-Combat Simulator [116 marks]

In this part you will complete a simulator for an RTS-like combat game. Circular units representing marines and tanks are roaming a rectangular map. They fight opponents' units whenever they come in attack range, but their motion policy is simplistic: they never collide with other units, but when hitting the map border, they either stop or bounce off similar to billiard balls.

We provide executable file `simul.lab` which runs on the lab machines and shows you how the simulator is supposed to work (in principle) when all parts have been implemented. Because there are some implementation choices, we don't expect your program to work exactly as `simul.lab`

The makefile can generate code for 2 scenarios: Option 1 creates code that can be run remotely through an `ssh -X ... session` (provided your local computer runs an XWindows server). Option 2 is generates a (faster) version that only runs when sitting in front of a lab machine. Comment out the option you don't use.

The simulator takes various parameters. To see a list, run `./simul.lab x`. To check your code for memory leaks run valgrind with `--leak-check=full`. This will report where leaks exactly happen. This way you can distinguish leaks in GL and glut from yours. Another idea is to test your code without graphical output to speed up experiments.

To compile the project, run make. This creates executable file simul, which by default opens an OpenGL window showing hundreds of units fighting until one team has no units left (or both).

We consider two unit types: Marine and Tank. They have the following unit stats:

	Marine	Tank
radius	10	20
attack_radius	40	80
max_speed	10	15
damage	1	4
hp	45	100

Those properties have to be initialized in the units' constructors. There are two steps of solving this problem:

1. Understand the provided code and complete it by implementing all functions commented with `// ... implement`

   Only change the source files you will submit: `World.cpp Marine.cpp Tank.cpp` . We won't see any changes you make to `.h` files and to `World2.cpp`

   We suggest to compare the output of your simulator with that of `simul.lab`'s using small unit counts and bounce switched off and on

   If you encounter runtime issues, it is a good idea to pass on a random number seed != 0 to make program runs repeatable

2. Use YOUR program in a set of experiments that determine which of the three attack policies (selecting a random weakest, random closest, and random most dangerous target) is the strongest for the following parameter settings:

   ```
   width=height    :   700
   marines=tanks   :   20, 50, 100, 300
   bounce          :   true
   redpol/bluepol  :   all combinations
   ```

   To gain statistical confidence, run each setting 100+ times (with differnt seeds). We'd like to see a brief writeup in `Experiment.txt` explaining your experiments and how you interpret the results

For this programming assignment you CAN use C++ containers or functions from `STL` and Boost that are installed on the lab machines. In particular, class template `std::vector<T>` will be used extensively in this assignment which we will cover later in the course. For our purposes, you only need to know that a `std::vector<T> v;` behaves like a dynamic array of type `T`. E.g., `std::vector<int> v;` defines an empty `int` vector. You can access elements by index like arrays (e.g., `v[0]`), you can query its size by calling `v.size()`, you can add elements at the end using `v.push_back(x);` and remove the last element using `v.pop_back()`. If you need more information about `std::vector`, please consult `http://www.cplusplus.com/reference/vector/vector/` or other online material.

Your submission must compile on the lab machines without errors/warnings using the provided makefile and running "`make`" . It will be marked considering its correctness (syntax + semantics), documentation, efficiency, and coding style. Add assert statements to check important pre/post-conditions

---

2. [40 marks] For this part please provide solutions (i.e., formal proofs/disproofs) in file `Problem2.txt` (or `Problem2.pdf` if you prefer to use a text processor like LaTeX or Word). Be as formal as you can, demonstrating - based on the definitions introduced in class - a rigorous chain of logical implications leading from the premise to the conclusion - or providing a counter example:

(a) Prove that optimal heuristics (i.e. $h^*(n)$) are consistent

(b) Prove: If $h_1(n), \ldots, h_k(n)$ are admissible, so is $h(n) = \max(h_1(n), \ldots, h_k(n))$

(c) Prove: If $h_1(n), \ldots, h_k(n)$ are consistent, so is $h(n) = \max(h_1(n), \ldots, h_k(n))$

(d) Prove or disprove: If $h_1(n), \ldots, h_k(n)$ are admissible, so is $h(n) = h_1(n) + ... + h_k(n)$

(e) Prove or disprove: If $h_1(n), \ldots, h_k(n)$ are admissible, so is $h(n) = (h_1(n) + ... + h_k(n))/k$