INFO6044 – Game Engine Frameworks & Patterns - Mid-term Exam – Fall 2015

Instructor: Michael Feeney

Friday, October 30th, 2015

The exam format:

- You may use any resources you feel are necessary to complete the exam, but you are to answer the questions **on your own**. I will be looking for plagiarism (i.e. copying) very carefully. There is *no possible way* that the specific code to answer these questions, or the output to the screen, would be very similar to the look of another student's code. Remember, this is a test and there are very clear policies about cheating on tests.
    - http://www.fanshawec.ca/admissions/registrars-office/policies/cheating-policy
    - http://www.fanshawec.ca/sites/default/files/assets/Ombuds/cheating_flowchart.pdf
- It is an "open book" exam. You have access to anything you book or internet resource you'd like
- The questions are ***NOT*** of equal weight. The exam has **four (4)** questions and **four (4)** pages.
- The answers may be one or a combination of the following:
    - Short answer (in your own words)
    - Snippets of code
    - Complete running solutions
- **CLEARLY** indicate which answer goes to which question. My suggestion is that you place each answer in its own folder, named "Question_01", "Question_02" and so on (or something equally clear). Another option is to create a Visual Studio solution and add a number of projects – one per question – to it. If I can't make heads or tails of what question is what, I probably won't even mark it.
- Do ***NOT*** do some clever "*oh, you just have to comment/uncomment this block of code*" nonsense. However, if the questions ***CLEARLY AND OBVIOUSLY*** build on each other, you may combine them (like if one question places objects, then the next one moves objects around with the keys) – even so, **MAKE IT 100% CLEAR** to me what questions the solution is attempting to answer.
- Place any written ("essay" or short answer) answers into a Word, RTF, or text file. Again, *clearly* indicate which question you are answering.
- If you are combining answers (which is likely), please indicate this with a "readme" file or some note (*not* buried in the source code somewhere).
- For applications: if it doesn't build and run, *it's like you didn't answer it*. I'll correct trivial, obvious problems (like you clearly missed a semicolon, etc.), but you need to be sure that it compiles and/or runs.
- You have until **1:00 AM** on **Saturday, October 31st** to submit all your files to the Fanshawe Online dropbox.

NOTE: Although this may "look and feel" like a project, it isn't, it's an **exam**, so there is **no concept of "late marks**"; if you don't submit your files by 11:59 PM, you don't get any marks at all. *Don't Be Late submitting.*

(Also be **SURE** that you are actually submitting the correct files)

Questions:

You have been hired to help make an epic robot battle game. There are originally four types of battle robots, but the plan is to add additional robots over time.

You are to make an application (I'd suggest a console application) that implements the following features.

Note that most of the questions build on each other.

Weapons:

There are three different weapons that the battle robots can wield. They are based on a common interface:

Type	**Weight**	**Damage/second**	**Cool down (seconds)**	**Accuracy (chance of hitting)**
Machine Gun	100kg	5 health units/second	1 second	1 out of 5 shots
LASER Gun	200kg	10 health units/second	5 seconds	2 out of 5 shots
Missile Launcher	400kg	50 health units/second	30 seconds	4 out of 5 shots

Here's what the values mean:

- **Weight:** Is used to determine how quickly a robot can move, and how many legs it would need. The heavier the weapon, the more and/or larger legs it needs, and the slower it might move.
- **Accuracy:** How likely the weapon will be at hitting the target (1 in 5 means it has a 20% chance of hitting). This accuracy is only an issue if the target is moving – if the target is stopped, then the weapons have 100% accuracy (hit every time they are fired).
- **Cool down:** How quickly the weapon can fire. A cool down of 1 second means it can fire one shot per second. 30 seconds means it needs to wait 30 seconds between shots.
- **Damage/second:** If the weapon hits, this is how many health points will be taken off the target.

There is some starter code (in the code.7zip file) that implements a "Machine Gun" and a "LASER Gun". You need to implement the "Missile Launcher" yourself. While you *can* change some of the implementation details of these classes (virtual, interface, private data, adding methods, etc.), you **cannot** alter the methods (the names and/or signatures of the methods). You may also encapsulate these classes into other classes if you wish.

Robot Legs (for walking or whatever – they're robots):

Each robot has two (2) or more legs, with different capabilities and total "health". You *can't* mix and match legs, so each robot would only have 2 (or more) of *one type* of leg.

There are three main types of legs:

Type	Total Health	Power	Weight
Tiny Leg	50	Can move 100 kg	30 kg
Medium Leg	100	Can move 200 kg	75 kg
Heavy Leg	250	Can move 400 kg	150 kg

Here's what the values mean:

- **Total Health**: How much damage (from weapons) the leg can take. When the health gets to zero, the leg stops working. If all the legs stop moving, then the robot isn't moving and any robots firing on it hit every time they shoot (i.e. the accuracy of the weapons is 100%).
- **Power**: How much each leg can move. So the smallest robot, with two "Tiny Legs", can move 200 kg. If the legs are too small, or there aren't enough of them, then the robot can't move (and anything shooting it will always hit).
- **Weight**: How much each leg weighs.

Total robot weight:

The total weight of the robot is equal to:

The total number of legs * their weight
+ The weight of all the weapons
+ 10% of the total weapon weight
+ 15% of the total leg weight

These last two percentages (10% and 15%) are for the torso, batteries/engine, and all the other bits that make up the robot (that aren't legs and weapons).

Keep in mind that the legs you need are based on the ***TOTAL*** weight.

Total robot health:

As the weapons "hit" the enemy robots, they take damage in the form of losing "health". When a robot has <= 0.0f health, they are completely destroyed.

The total health of the robot is:

(50 health points per weapon + total health of all the legs) + 50%

(i.e. the health is 50% more than the total of the weapons and legs)

When a weapon "hits", damage is taken from the legs first, then whatever is left. This is so the robot can be immobilized, allowing the enemy robots to have 100% accuracy. Even if a robot can't move (their legs are destroyed), then can still fire. At least while they have health that is.

1. (30 marks) Rather than create the weapons directly, make an “Abstract Factory” that returns an instance of the specific weapon you want. You may alter the weapons classes in any way you see fit, provided that you ***do not*** change the method names, what they return, or the method signatures. You may also add any other classes you see fit.

   Write an application showing that you can create and use each type of these weapons. Note: This doesn’t have to be exhaustive, just that you can show that you are actually creating an instance, are able to call the methods, and that they weapons are, in fact, different.

2. (60 marks) Make six C++ classes that represent six types of robots, two with each type of weapon, but with different types of legs.

   Robots that have the same weapon have to differ in the type and/or number of legs. For example, you could have a robot with a single “Machine Gun” and (say) two “Tiny Legs”, and another “Machine Gun” robot with four “Tiny Legs” Or two “Medium Legs”.

   You’ll need to choose the appropriate size and number of legs for each robot (enough “power” to move that robot).

   The idea is that each weapon-leg configuration is a different type or class or robot. So you would decide that “Robot Type 1” which would have: 1 “LASER gun”, and 4 “Medium Legs”.

   Once this is decided in your game, you would only be able to create instance of these particular types (classes) of robots, rather than “assembling them” by combining a base robot + certain weapon(s) and certain legs. HOWEVER, this does not mean that you can’t not “build” them in code (nudge-nudge, wink-wink).

   Rather than creating these robot classes directly, create a software device that allows you to request a certain type of robot and it is returned (i.e. an Abstract Factory, a Builder, or a combination of both).

   Demonstrate the creation of these six types of robots. Like question 2), you don’t have to exhaustively call all methods on the robots, just demonstrate that you are actually creating working instances of each type of robots and that they are, in fact, different.

   You will have to determine how the robots “take damage” and how to retrieve what health (and any other aspects) they currently have. At a minimum, they will have to take “hits” from the weapons, lowering their “health”.

3. (40 marks) Create an application that creates ten (10) robots and battles them against each other.

4. (40 marks) Add a fourth weapon of your choice (with parameters of your choosing), and a third type of leg, and create four (4) new robot classes that incorporate a combination of the original and these new elements (i.e. “Medium Legs” + your new weapon, or “LASER Gun” with your new types of legs). Incorporate them into the battle in question 3.

(That’s it for the exam).