INFO6015 – Animation 1, Mid-term Exam – Fall 2013

Instructor: Michael Feeney

Thursday, October 24th, 2013

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.
    - o http://www.fanshawec.ca/admissions/registrars-office/policies/cheating-policy
    - o http://www.fanshawec.ca/sites/default/files/assets/Ombuds/cheating_flowchart.pdf

- The questions are ***NOT*** of equal weight. There are four (4) pages with five (5) questions

- The answers may be one or a combination of the following:
    - o Short answer (in your own words)
    - o Snippets of code
    - o 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.

- Place any written 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 **9:00 PM** on **Thursday, October 24th** to submit all your files to the appropriate drop box on Fanshawe Online.

  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 9:00 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)

- You can reach me through e-mail (mfeeney@fanshawec.ca) or 519-494-7569 (but if you call me in the middle of the night, I'll be seriously annoyed...).

Questions:

1. (marks listed below) Olympic Bunny Rabbit Relay Race! Ready? Steady?? Go!!!

   With your mad programming skills, create a simulation of Bunny relay race.

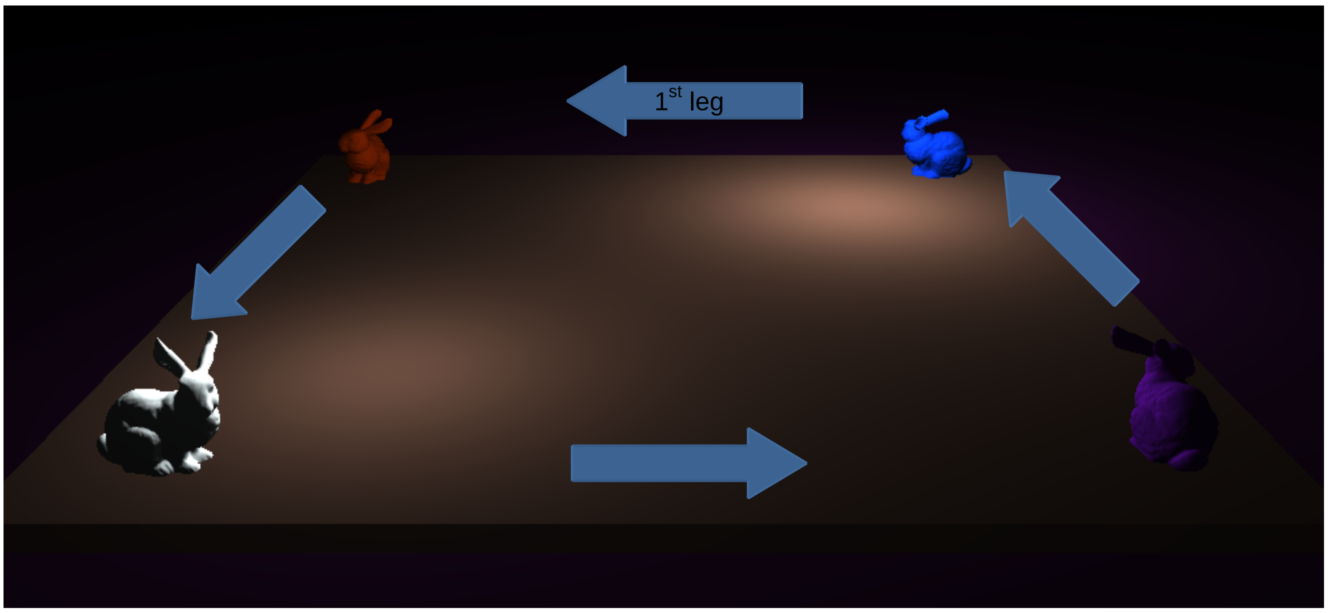
   Just in case you don't know, a 'relay race' is where a bunch of runners (often four) take turns, running only a part of a longer race. For example a 4x100 means that there are four running, each running 100m (a "leg") of a 400m race.

   The trick is that they each have a 'baton' (aka "a short wooden stick") that they have to pass from one to the other, at each leg (stage). The first runner runs 100m, then passes the baton to the next runner, who is waiting within a "zone" around the 100m mark. The 1st runner must successfully 'pass' the baton to the 2nd runner within this zone or they are all disqualified. Once the 2nd runner has the baton, they can run at full speed to the next leg of the race.

   In our Bunny Rabbit Relay Race, there are some differences:

   - They run the race in a square, not a straight line or oval track
   - They do the relay four times (i.e. when the last runner is done, they "tag" the first runner again, and do this entire thing four (4) times)
   - They don't actually have batons… because they're rabbits… and don't have hands, right…

   For example, in the race below, the blue bunny (top-right) is runner (hopper?) number 1. It would run (hop?) to the left until it touched the orange bunny (top-left). At that point, the orange bunny would run (hop) down to the white bunny, who would run to the purple bunny. Then they would repeat this (where things get a little complicated…. see below)



Some things to note:

- When the bunnies are done each leg, they would start at a different place for the next stage of the race. For example, bunny #1 (the blue one), starts the race at the top-right corner, but ends up at the top-left, so it would start there for the next "stage".
- The last bunny has to run twice as long, and has to turn for the next "stage". The purple bunny would start at the bottom-right, running to the top-right. But the blue bunny has moved to the top-left, so the purple bunny would have to turn and run from top-right to top-left to reach it. This is OK as the last runner of a relay race is usually the fastest.

Marks:

The bunnies must move from one place to another, in order, in exactly five (5) seconds. In other words, the blue bunny should take five (5) seconds to "run" from the top-right to the top-left of the screen.

a) 20 marks: If you do one complete "circuit" of the race, like a normal "human" relay race. In other words, the bunnies would only run once, then stop (instead of four times).

b) 25 marks: Same as a) above, but the bunnies "hop" instead of just move.

c) 30 marks: Same as a), but they would run the complete race (so four complete circuits, including the last bunny turning and running 2x as far. Note: the last bunny should take 10 seconds in total – **NOT** including the turn).

d) 35 mark: Same as c) above, but the bunnies "hop" all the way

NOTE: The code from here on can just use the console – it does ***not*** have to be graphical.

2. (10 marks) Create a pure virtual "Animal" interface class that has the following methods (note "float3" is just a short form for either three float values or a "float3" type class that holds things like x, y, and z – it's **not** a reference to the HLSH shader float3 variable):

```
o void Move( float3 destination )
o void Eat( string foodType )
o std::string getType(void);      // string of animal "type"
o void GetEaten( std::string whoAteMe );
```

3. (10 marks) Create a "Fox", a "Bear", and a "Rabbit" class, that implement the interface in question 2).

   These classes should have private data showing the "type" (i.e. "fox", "bear", or "rabbit") as well as a location value that gets updated when Move() is called. GetEaten() should display what animal ate them (bears can eat foxes and foxes can eat rabbits).

   White some code the shows the methods being called on an instance of each of these classes.

4. (10 marks) Next, create an Abstract Factory that allows the creation of all three types. Demonstrate the classes being created and executing.

5. (15-20 marks) Add the ability for specific instance of bears to be able to eat specific instances of foxes, and specific instances of foxes to each specific rabbits. You must use an abstract factory to create your objects (either the one you used in question 4 or an altered one, based on the requirements below).

   To do this, you may either:

   - (15 marks) Pass pointers around (so a fox eating a rabbit will have to “call” the GetEaten() method on the rabbit that it’s eating, or

   - (20 marks) Use “handles” instead (i.e. no object has an pointer to another object).

That’s it