**INFO6044 – Game Engine Frameworks & Patterns**

**Midterm Exam – Tuesday, November 27th, 2017**

Instructor: Michael Feeney

## 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 **six (6)** pages.
* Your solution can be either graphical or console based (or graphical + console based if that’s helpful).
* **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 **11:59 PM** on **Tuesday, November 27th** 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 the time the drop box closes, you don’t get any marks at all.

*Please don’t be late submitting.*

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

* Your solution may **not** contain any third party “core C++” libraries (like boost) or C++11 "**auto**" feature. If it has either of these things, the question(s) will not be marked (because it won't build). You many have other “utility” libraries, like ones to load textures, models, sounds, etc.
* When ready to submit, please delete all the “extra” Visual Studio files before zipping it up (remember this is C++, so all I really need is the .h and .cpp files, right?), like the “Debug” and “Release” folders with the “obj” files, as well as the intellisense file
* **If the solution does not build (and run), I will not mark it** (so you will receive zero on questions that can't be built and/or won't run). When I say "run", I'm not speaking about some, random, unforeseen bug, but rather something that you should have obviously dealt with, like memory exceptions, etc.
* Unless otherwise indicated, all these solutions assume that you are creating/using a C++ project using Visual Studio 2008 through 2017 using the OpenGL 4.x API (with freeGLUT, GLEW, and GLM).

**“Horizon Zero Dusk, Recycling in the Future!”**



In the far future, humanity has moved off the Earth to another planet (let’s call it “Earth 2”). Since the original Earth was so polluted already, the (evil future) humans have decided to just keep dumping their garbage there.

Sure, it’s expensive to ply their garbage to another planet, but let’s assume that there’s some new “space engine” that allows ships to fly there for very little expense. ...and we’ll ignore that a civilization that could invent *that* kind of transportation still couldn’t just figure out what to do with their garbage... yeah.

Anyway, a lot of their garbage was high tech gadgets (the iPhone 57, the Android Galaxy 1627, and the xbox and PlayStation ∞, among other things, like automatic coffee machines, self driving cars, etc.), and these gadgets, through a *very* unlikely sequence of events, has created an entire race of sentient robots!

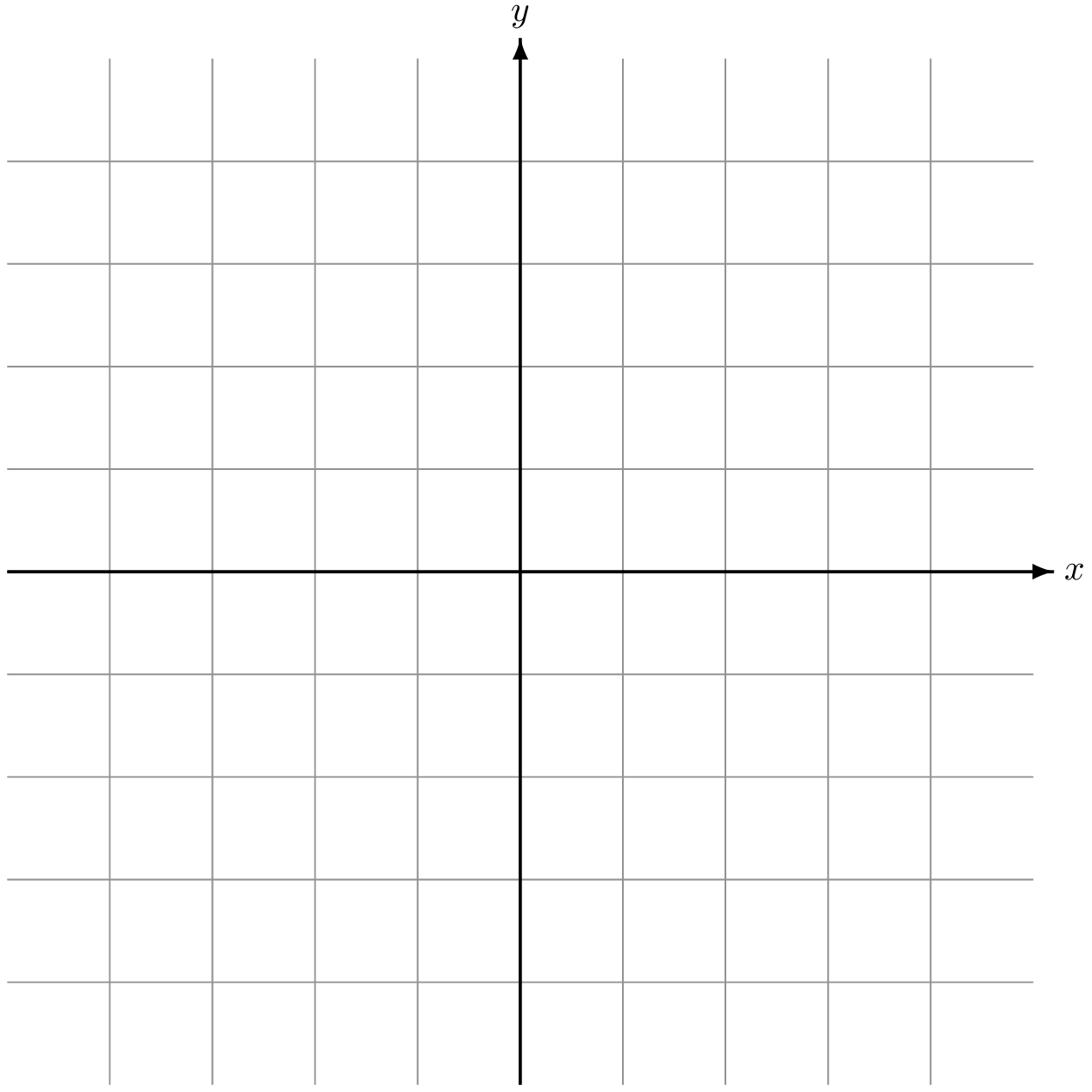
Unfortunately, they don’t know how to change the resources of Earth into stuff they need, and so rely on the constant dump of new garbage and recycling materials in order to “survive”.

You are to create a program simulating a small portion of this “sentient robots eating garbage” ecosystem.

The only mathematical calculation you will need is the distance between two points. This can be done using Pythagorean theorem, or using a “distance” or “length” method/function a library like glm.

**Simulation details, the world:**

Everything is contained in a “world” which is a circular area with a 2000 meter radius. The circle is divided into two dimensions and each object can be at a certain x, y location, however objects can’t be more than 2000 meters away from the centre of the circular world. Assume that the centre of the circle is the origin (0,0):



0,0

Every 10 seconds, a ship arrives to dump new garbage on the planet. This garbage consists of a random assortment of various items. The breakdown is listed below:

**Simulation details, garbage/Recycling item**

|  |  |  |
| --- | --- | --- |
| **Type** | **Size** | **Range of items being dumped** |
| Aluminum Can | 0.1 kg | 0 to 50 |
| Steel Can | 0.2 kg | 0 to 25 |
| Electronics | 0.1 kg | 0 to 10 |
| Plastics | 0.5 kg | 0 to 100 |

For example, a garbage dump would contain somewhere between 0 and 50 aluminum cans, 0 to 25 steel cans, etc.

**Simulation details, sentient garbage robots:**

The robots wander around the world, looking for garbage to consume.

They all consume the same garbage, but at different amounts. When created, the capacity they can hold, and the rate of consumption of materials is randomly chosen within these ranges:

|  |  |  |
| --- | --- | --- |
| **Type** | **Capacity range** | **Consumption range** |
| Aluminum Can | 0.5 to 2.0 kg | 0.01 to 0.03 kg/second |
| Steel Can | 0.5 to 4.5 kg | 0.01 to 0.05 kg/second |
| Electronics | 1.0 to 3.0 kg | 0.01 to 0.04 kg/second |
| Plastics | 2.0 to 15 kg | 0.05 to 0.12 kg/second |

So a particular robot might have a capacity of 1.8 kg of aluminum (0.5 to 2.0kg), which it will consume at a rate of 0.013 kg/second (0.01 to 0.03 kg/second).

Note: The capacity and rates don’t change during the “life” of a robot; they are chosen at “birth” and remain the same.

The simulation is to be discrete, with each “tick” of the simulation having the following rules followed:

* For each robot:
  + The amount of garbage/recycling that the robot has is consumed by the rate proportional to the time tick. For instance, if you are simulating the example robot above, the amount of aluminum is has would decrease by 0.013 kg in 1 second, and a proportionally smaller amount if the simulation was running faster, so 0.0002.167 kg per tick at 60 ticks per second.
  + If the robot has no material, it can run for another 5 seconds before “dying”.
  + If the robot has run out of only some of the material, then it will consume the remaining material at a higher rate:
    - Out of 1 material only: Consumption rate of remaining material doubles (2x)
    - Out of 2 materials: Consumption rate of remaining material triples (3x)
    - Out of 3 materials: Consumption rate of remaining material quadruples (4x)
  + The robot will only seek out more garbage when that corresponding container has reached 90% capacity. In other words, if all the containers were filled to 95%, the robot would do nothing (except attempting to reproduce).
  + The robot will seek out the closest material to consume, based on the % remaining in its “tanks”. So, if the electronics tank is the lowest % based on the size of that particular robots electronics container, it will seek electronics. If there is a tie (same % remaining), then it will find material in alphabetical order (Aluminum cans, Electronics, Plastics, then Steel Cans).
  + If two (or more) robots reach the same item in the world, then the robot *with the most* material, overall, gets the piece of garbage/recycling

The robots can reproduce if *all* the containers are >= 50% full *and* there is enough material to give its offspring enough to fill the child’s containers by 10% (according to that child robot).

For instance, if the example robot has a child, it would pick random material container sizes (and consumption rates). This will almost certainly be different from the parent. The parent must have enough spare material on hand to fill the child’s container by 10% *from the perspective of the child*, so keeping in mind that the child might have much larger container sizes: if taking this amount of material from itself will leave its tank MORE than empty, then it *cannot* reproduce this time.

Note that each time the robot considers reproducing it would generate a *new* material container size and rate *each time*. In other words, all its children will be different. This also avoids the situation where a “small” robot tries to give birth to a much “larger” child: In that case, the parent can’t reproduce, abandons the “birth” cycle, and will pick a different child (hopefully a more “reasonable sized” one) at the next birth cycle. (Note that this means that, if the simulation is running at, say, 100 Hz, the robot might attempt to give birth 100x of times, in just a few seconds, before it is successful. But, hey, remember they are robots, not animals – animals couldn’t try to give birth 100x per second!)

1. (40 marks) Implement the “behaviour” of the world, showing the dropping, and accumulation of garbage/recycling. In other words, there are no robots in this version, and more and more garbage is accumulating.   
     
   If it is a graphical simulation, you can show this by placing different coloured objects in the scene where the “garbage” would be.
2. (40 marks) After about an hour of simulated time (so after the equivalent of 360 garbage dump events), add a single robot, allowing it to go through its cycle of consuming its material and collecting more. **PLEASE** alter the code so this *initial* “hour” happens *immediately*, not with the 10 second interval delay; after the “hour”, enable the 10 second interval. Note: The robot *doesn’t* reproduce at this point – so it will simply scavenge for food when “hungry”.
3. (40 marks) Add the ability to reproduce, and run the simulation again, starting with only 1 robot (more and more robots should appear)
4. (40 marks) Add two more variations of material, which have a 10% chance of being dropped. These have various and *permanent* impacts on the robots.

|  |  |
| --- | --- |
| **Type** | **Change** |
| Super Aluminum | Reduce the consumption rate by 10% of current rate |
| Nasty Steel | Drops the capacity of the steel container by 5% (i.e. the robot would have only 95% of the capacity that it previously had) |
| Quantum Electronics | Consumption rate of electronics is reduced by 2% |
| Dirty Plastics | The consumption rate of plastics is increased by 3% |

Note that these materials act exactly like the “normal” materials, in that the robots can’t tell them apart; it is only when they consume them that these effects take place.

The effects are cumulative, so if a robot keeps consuming Super Aluminum, its aluminum consumption rate will impact the current rate, exponentially.