CS 251: Code warrior: Makefiles and Box2D: Outlab

- Due: 11:55 AM 3/09
- Please write (only if true) the honor code. If you used any source (person or thing) explicitly state it. You can find the honor code on Piazza.

Overview

Small projects – those with a couple of modules – are easy to manage. Developers can recompile them easily by calling the compiler directly, passing source files as arguments. This simple approach fails when a project gets too complex; with many source files it becomes necessary to have a tool that allows the developer to manage the project easily and efficiently. That's where makefiles come handy.

But as we move from building software from one OS to another, we actually need to generate makefiles. cmake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner. Simple configuration files placed in each source directory (called CMakeLists.txt files) are used to generate standard build files (e.g., makefiles on Unix and projects/workspaces in Windows MSVC) which are used in the usual way. CMake can generate a native build environment that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations.

Talking of large projects, there will be times when you can't do everything from scratch. You'll use some of the existing software, and build on top of that. Ideally we would take something like Firefox or the Linux kernel and mess with it, but we would need a lot of other concepts. So We're going to take something you all know and love – physics. We will use an open source physics simulation library called Box2D, and create some nice simulations for ourselves. As you go along, think about the amount of work it'd have taken you had you started from scratch. Oh, Box2D was used, we are told, to create Angry Birds at some point.

A. Makefiles

In this exercise we look at a bit more advanced exercises in makefiles.

0. Write a makefile to compile into a single executable all cpp files in the current working directory. (The key optimising aspect is that only those cpp that have changed since the last build must be recompiled).

   Note: There can be any number of cpp files in the directory. A cpp file might have dependencies on any number of header files. You can assume that all the files are in the same directory. The task is to write a makefile that generates object files for each of these cpp files and links them together to give the executable **out**. The makefile should be efficient, i.e., it should recompile only the files that have been modified. Also it should work with addition of new cpp and .h files and changing the dependencies on header files. It should also have a target **clean** that cleans up the directory by deleting all the additional files that you generated as a part of `make`. Also explain your solution briefly in answers_A.txt

1. Write a CMakeLists.txt for the files given in task A of inlab. Running `cmake` should create a makefile that should be compliant with all the specifications given in inlab (up to `makefile_3`). You can assume that all the cpp files are in the same directory and only those files are in the directory. Attach a (as usual small in size but readable!) screenshot (image_A.jpg) of a working system for the Windows platform (your makefile should work across Linux and Windows) Sample screenshot for windows will be made available in the support folder (for reference – feel free to be creative.)

B. Box2D

In the inlab, you got to see what Box2D is all about. You worked on modifying a demo, and some of you might even have injected some keyboard controls into it. For the outlab, we will create something different, and try to do things that Box2D doesn't allow us to do inherently.

For this task, we want to simulate magnetism in nether Starwar space. You will create two kinds of materials: one magnetic material that can exert magnetic force, and a second metallic material that can feel the magnetic force. Strangely, the one exerting a force doesn't feel the force; the traditional Newton's third law doesn't hold for them. So for instance, in a scenario with A and B, A will move towards B, but B will not budge (instead of, as in real life, both A and B moving towards each other).

The force can be attractive, or repulsive.

A metallic object experiences a force of magnitude

$$F = \frac{k}{r^2}$$

due to every magnet in the environment where you can choose any value for k that suits your demonstration and r is the distance between the magnet and metallic material. A magnet can change character to be attractive or repulsive by a toggle. You can approach this problem however you want to, but keep in mind that the simulation has to be as close to the specification as possible.

Some notes: The force experienced by the magnetic material can be towards or in opposite direction to the magnet. If needed, you can limit the range of the force in which the magnet can exert the force to be $r0$, whose value can be decided based on your demo. It's up to you to decide the shape, size and other parameters of the objects.

0. Perform a simple simulation with the behaviour mentioned above.

   Submit the entire src folder (similar to the dominos folder i.e. do not submit the entire Box2D source) which when compiled by us will result in a demo of the 2 kinds of objects, and document every change that you made in the file answers_B.txt (no major change should be unaccounted for). See submission guidelines for more details. Try to make your design as modular and structured as possible. The demo can be very simple with just 2 magnetic objects and one metallic object. You can present it in any way you want, which makes it easy for us to verify all the properties.

1. Add a functionality to toggle on a click, the direction of the magnetic force exerted by the magnet. That means, if the force is attractive to begin with, a click will make it equally repulsive. Also, add a keyboard shortcut 'Q' to reset all the forces to attractive, and another shortcut 'T' to toggle the forces on all the materials at once. A sample executable is given for this task is given in the support folder.

   You need not replicate the behaviour provided in the sample, your demo just needs to fit with the description. Again account for the files that you have made changes with the modifications in answer_B.txt.

2. **[Extra Credit]** Now that you've learnt a bunch of different things, try putting it all together into a single piece. Make a small game with the following setup: There are objects which exert magnetic force, repulsive to start with, embedded on the screen (i.e., in the air). There is a horizontal slider near the bottom, which can be controlled using your Left/Right arrow keys. The magnetism can also be controlled in other ways described earlier. Once the game starts, a ball made of magnetic material drops from some random point near the top, goes around in the magnetic field, bouncing around because of collisions and magnetic forces, till it eventually falls out of it due to the gravity. At that point, you'll move to capture it. If you do, 2 balls will pop up near the top at two random points. Same things follow. Each time you successfully capture a ball, 2 balls appear again. The game resets when one of the balls escapes (your slider) on to the ground. The magnetism controls can be used tactically to help you in capturing the balls. This

might look a bit too tedious, but a very structured implementation of the earlier task will make this just creating multiple instances of the magnets. The slider can be borrowed from inlab. You just need to identify collision of the ball with the slider. Submit the src folder for this sub task.

Submission Guidelines

1. When you submit, please document individual percentages such as Student 1: 80%, Student 2:100%, Student 3:10% in the readme.txt. In this example, the second student will get full marks (10/10) and the first student will receive 8/10.
2. Do include a readme.txt (telling us whatever you want to tell me). (The reflection essay is not required for inlab, but is required for outlab.) Do include group members (name, roll number), group number, honor code, citations etc.
3. The folder and its compressed version should both be named `lab05_groupXY_outlab` for example folder should be named `lab05_group07_outlab` and the related `tar.gz` should be named `lab05_group07_outlab.tar.gz`
4. The submission folder should contain 3 sub-folders `taskA` `taskB` and `Extra Credit`
5. The subfolder taskA contains 4 files - makefile, CMakeLists.txt, image_A.jpg and answers_A.txt. answers_A.txt should contain the explanation of how your makefile works.
6. The subfolder taskB should contain the src directory of your project. If you have changed/added any other files add them to the subfolder and mention the instructions that need to be followed for handling them in instructions.txt. It should also contain an answer_B.txt that clearly specifies the files you have changed/added and the changes that you have done in each file.
7. The subfolder Extra Credit should contain the src directory of your project. If you have changed/added any other files add them to the subfolder and mention the instructions that need to be followed for handling them in instructions.txt.
8. Your submission folder should look something like this:

```
lab05_groupXY_outlab
|__taskA
|   |__makefile
|   |__CmakeLists.txt
|   |__image_A.jpg
|   |__answers_A.txt
|__taskB
|   |__src
|   |   |__callbacks.cpp
|   |   |__dominos.cpp
|   |   |__xyz.cpp
|   |   |__pqr.cpp
|   |   |__..
|   |__other files changed/added, if any (unlikely)
|   |__instructions.txt, if needed
|   |__answer_B.txt
|__Extra Credit
    |__src
        |__callbacks.cpp
        |__dominos.cpp
        |__...
```

```
│    │
│    │
│    │
│    ├── other files changed/added, if any
│    └── instructions.txt, if needed
└── readme.txt
```

How We will Grade You[80 + 20 Marks]

Extra credit points are available to you only if you get the basics right.

1. Task A [**30 Marks**]
   - `makefile` : 15 Marks
   - Makefile explanation: 5 Marks
   - `CMakeLists.txt` : 5 Marks
   - `image_A.jpg` : 5 Marks
2. Task B [**50 + 20 Marks (extra credit)**]
   - Magnetic - Metallic material implementation and answers_B.txt : 35 Marks
   - Mouse and Keyboard events : 15 Marks
   - Extra credits : 20 Marks (will be graded only if you have close to perfect marks in other parts)