

24783 Advanced Engineering Computation: Problem Set 3

(*) In the following instruction (and in all of the course materials), substitute your Andrew ID for where you see *yourAndrewId*.

START EARLY!

1 Check Out or Update Base Code and Libraries

Please make sure you have up-to-date libraries and course files before starting an assignment.

If you have not done working-directory set up as described in the first assignment (like in case you need to work from a different computer), please see Problem Set 1 and set up the working directory.

I assume you created the working directory called *24783* under your home directory and you checked out your Git repository in there.

Home directory is typically *C:\Users\username* in Windows, */Users/username* in macOS, and */home/username* in Linux, where *username* is the user name in your local computer.

First, open command-line (Developer PowerShell or Terminal), and move to your working directory by typing:

```
cd ~/24783
```

You need to check out (or clone) Git repositories once. If you have not checked out yet, do the following:

```
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/course_files.git
git clone https://yourAndrewId@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

You need to replace "yourAndrewId" with your Andrew ID. You'll be asked to type in credentials.

Also we are going to use two additional repositories:

```
git clone https://github.com/captainys/MMLPlayer.git
git clone https://github.com/captainys/public.git
```

If you are successful, you should have the following directory structure under your home directory.

```
Your User Directory
├── (Other files and directories)
├── 24783
│   └── course_files
```

```
|
├─ public
├─ MMLPlayer
└─ yourAndrewID
```

If you already have checked out these repositories (most likely you did for Problem Set 1), you need to update (or git pull) in those repositories. By change directory to the location where you checked out repositories and then type:

```
git pull
```

To update all four repositories, you can type the following commands in a sequence:

```
cd ~/24783/course_files
git pull
cd ~/24783/yourAndrewID
git pull
cd ~/24783/public
git pull
cd ~/24783/MMLPlayer
git pull
```

2 Copy Base Code and Add to Git's Control

Do the following to make a copy of the base code to your directory.

```
cd ~/24783
cp -r course_files/ps3 yourAndrewID/.
```

Adding `./` after the destination directory is a weak defense to prevent files from copied to wrong location in case you misspelled the directory.

And then type the following to add them to the Git's control:

```
git add yourAndrewID/ps3
```

Once you add ps3 sub-directory to Git's control, you can do commit and push as many times as you want to send your files to the server with no penalty before the deadline.

It is recommended to make frequent commits so that you can go back to earlier version in case you mess up.

3 Make a CMake Project

3.1 Top-Level CMakeLists.txt

Write a top-level CMakeLists.txt under ps3 sub-directory, so that:

- it enables C++11 features,
- it includes ps3 sub-directory,
- it includes MMLPlayer library (optional), and
- it includes public libraries.

Since public-library sources are located outside of your source tree, you also need to specify where the build files are written. Therefore, the lines for including the MML player and public libraries should look like:

```
add_subdirectory(../../public/src ${CMAKE_BINARY_DIR}/public)
add_subdirectory(../../MMLPlayer/ym2612 ${CMAKE_BINARY_DIR}/ym2612)
add_subdirectory(../../MMLPlayer/mmlplayer ${CMAKE_BINARY_DIR}/mmlplayer)
```

3.2 CMakeLists.txt Files for Executables

Write a CMakeLists.txt file in ps3_1 sub-directory. It must define an executable called ps3_1, which uses ps3.cpp, data.h, map.cpp, ports.cpp, and tiles.cpp. Don't forget MACOSX_BUNDLE keyword even if you are doing it in Windows or Linux to make it cross platform.

Once you set up CMakeLists.txt files, test-build ps3_1. First you create a build directory somewhere outside of ps3 directory, and then run cmake, and then type:

```
cmake --build . --target ps3_1 --config Release
```

to build the executable. If you omit `-target ps3_1`, you will need to wait for libraries unrelated to this assignment. If you run the ps3_1 at this time, you will see Fig. 1:

3.3 Add to Git's Control

After writing these files, make sure to add the files to Git's control.

4 Refactor ps3.cpp in Event-Driven Programming Style

Let's practice event-driven programming style one more time.

Refactor ps3.cpp in event-driven programming style as introduced in class. The main function should look like:

```
int main(void)
{
    FsOpenWindow(0,0,800,600,1);

    ApplicationMain app;
    while(true!=app.MustTerminate())
    {
        app.RunOneStep();
        app.Draw();
    }

    return 0;
}
```

Therefore, you need to write ApplicationMain class, which has at least MustTerminate, RunOneStep, and Draw member functions.

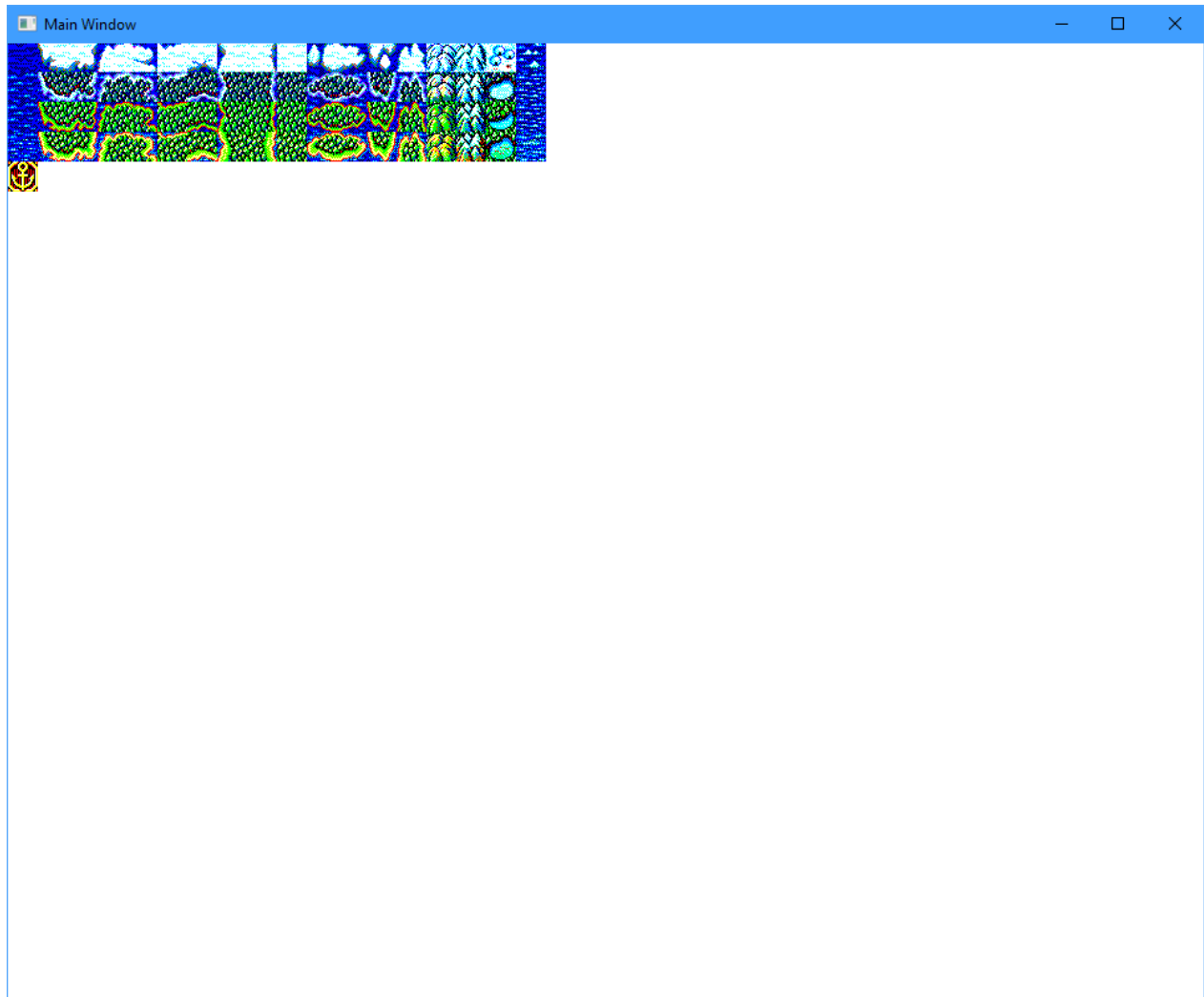


Fig. 1: Sprites used for drawing map in Daikoukaijidai 1 (Uncharted Waters 1) by KOEI Tecmo Games 1991

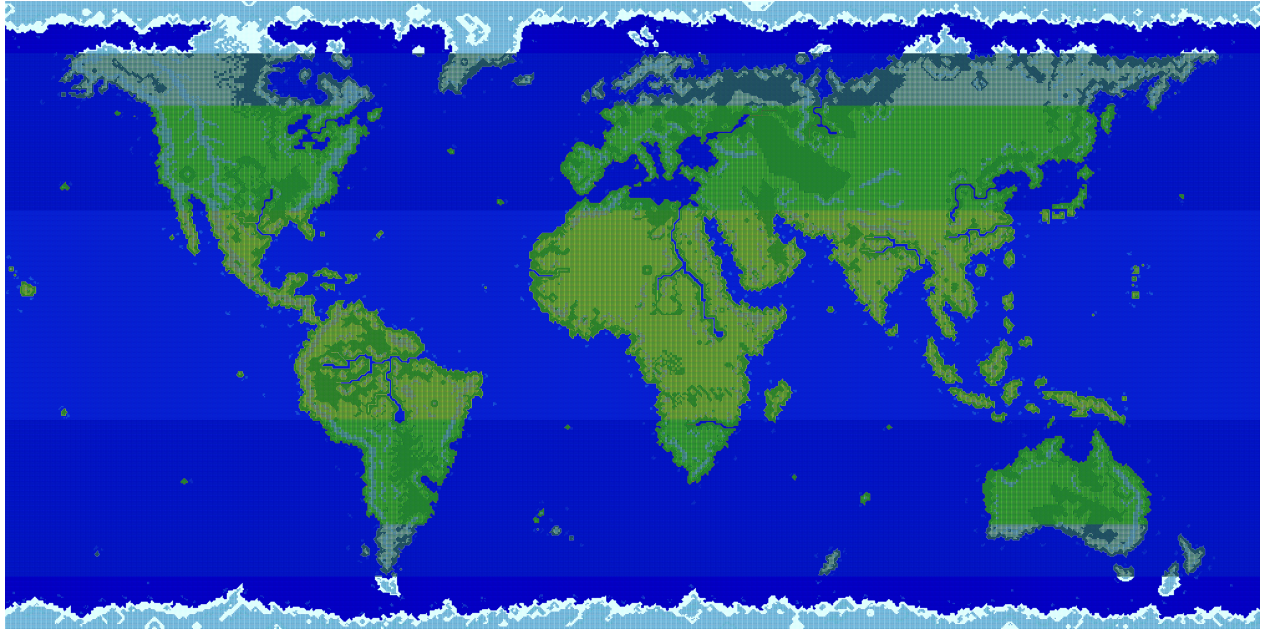


Fig. 2: World map of Daikoukaijidai 1 (Uncharted Waters 1) by KOEI Tecmo Games 1991

5 Map Viewer

Fig. 2 is the world map of a classic game called Daikoukaijidai 1 (Uncharted Water 1) by KOEI Tecmo Games. It was the 1st of a very successful series. The game was ported to many different platforms, re-made, and the latest version is (I believe) Daikoukaijidai 5. The goal of this assignment is to make a world-map viewer.

All necessary data are provided in `map.cpp`, `tiles.cpp`, and `ports.cpp`. By including `data.h`, you have access to all data.

The map consists of 72×36 pages of 14×14 blocks, and each block is 24×24 pixels wide. Therefore, overall, 1008×504 blocks.

The map uses four sets of sprite patterns. One for north and south polar regions (top and bottom 42 blocks), sub-polar regions (next 42 blocks), and mid-latitude regions (next 84 blocks), and 168 blocks across the equator. You can see different colors from the picture below.

In the base code, you are given:

- (1) An array of block codes for the entire map (1008×504). The array is tightly packed, therefore a block at block coordinate (bx, by) is `world_map[by*1008+bx]`.
- (2) An array of sprite patterns (24×24 pixels each) for each map-regions. Each bitmap is tightly packed so that you can directly give it to `glDrawPixels`. Sprite pattern for block code N ($0 \leq N < 18$) is `tiles_arctic[N]` for arctic regions, `tiles_subArctic[N]` for sub-arctic regions, `tiles_midLatitude[N]`, and `tiles_equator[N]` for across the equator. Also a sprite pattern for ports is provided in `tile_port`.
- (3) 70 port locations in block coordinates in an array `port_locations`. `port_locations[n*2]` is X, and `port_locations[n*2+1]` is Y. ($0 \leq n < 70$)

The base code opens 960×768 pixel window. Therefore, you can put 40×32 sprites on the window.



Fig. 3: Lisbon, where the game starts.

Your program needs to retain a coordinate of the block shown at the top-left corner of the window, and must draw 40×32 sprites of the map starting from this coordinate. I suggest to start with this top-left coordinate as $(0,0)$. Then, once you are confident in your rendering code, make initial coordinate $(434,112)$ so that Lisbon is visible when the program starts.

Also, draw port symbols when a port is within visible range. The port symbol is also 24×24 pixel sprite defined in array `tile_port[]`.

See the base code for how to draw a bitmap using `glRasterPos2i` and `glDrawPixels`. Note that `glRasterPos2i` specifies lower-left corner. So, in the sample, `bitmap height minus one` is added to the Y coordinate given to `glRasterPos2i`.

If your rendering code is correct, and you set initial top-left block coordinate as $(434,112)$, your window should look like Fig. 3. Depending on your video driver's OpenGL implementation, you may or may not see a white vertical and horizontal lines, but that's ok.

Then, let the user control the top-left corner coordinate. Imagine you are controlling a satellite by arrow keys (`FSKEY_LEFT`, `FSKEY_RIGHT`, `FSKEY_UP`, and `FSKEY_DOWN`). When the user presses `FSKEY_LEFT` or `FSKEY_RIGHT`, top-left block coordinate X must decrease or increase by 7 respectively. When the

user presses `FSKEY_UP` or `FSKEY_DOWN`, top-left block coordinate `Y` must decrease or increase by 7 respectively. If the user presses `FSKEY_ENTER`, make top-left block coordinate `(0,0)`.

The world must be connected left and right. Therefore, if top-left `X` becomes negative, add 1008 to keep it $0 \leq x < 1008$. The rendering also must take it into account. If the block coordinate for a window location goes $1008 \leq x$, draw a sprite for map block of `(x-1008,y)`. If block coordinate `y` is $y < 0$ or $504 \leq y$, do not draw anything (keep it white background).

5.1 5-Point Bonus for Half-Size Rendering

By using:

```
glPixelZoom(0.5,0.5);
```

you can draw half-scale bitmap without changing `glDrawPixels`, and

```
glPixelZoom(1.0,1.0);
```

for scaling back. When you make it half size, the bitmap will be rendered as 12x12 pixels if you are give 24x24 pixels as parameters to `glDrawPixels`. Therefore you can render sprites half size.

Draw a map with 12x12 bitmaps using `glPixelZoom`. Therefore the program must show wider range in `X` and `Y` directions.

Your program must have a flag to switch between normal and half sizes. If you go for the bonus point, start your program with half size, and when the user presses the space key (`FSKEY_SPACE`), make it normal size. `FSKEY_SPACE` must toggle between normal and half sizes.

If you render half size, your window should look like Fig. 4. Depending on the OpenGL driver implementation, it may not sample pixels consistently and you may see color inconsistency, but that's ok.

Also, `glDrawPixels` is one of the slowest functions of OpenGL. You might feel the response of your program gets poor, but it is expected. To do faster rendering, you need to use texture mapping instead of `glDrawPixels`. But, not required for this assignment.

6 Test Your Code on the Compiler Server

Test your source files (`.cpp` and `.h` files) on the compiler server. Some assignment may not require `.h` files. You do not have to test files that you don't make modifications. The files you need to test are the ones you write or modify.

We have four compiler servers:

- <http://freefood1.andrew.cmu.edu>
- <http://freefood2.andrew.cmu.edu>
- <http://freefood3.andrew.cmu.edu>
- <http://freefood4.andrew.cmu.edu>

Compiler servers are accessible from within CMU network only. To access from the outside network, use VPN to connect to the CMU network.

Make sure you don't see red lines when you select your files and hit "Compile" button on the server.

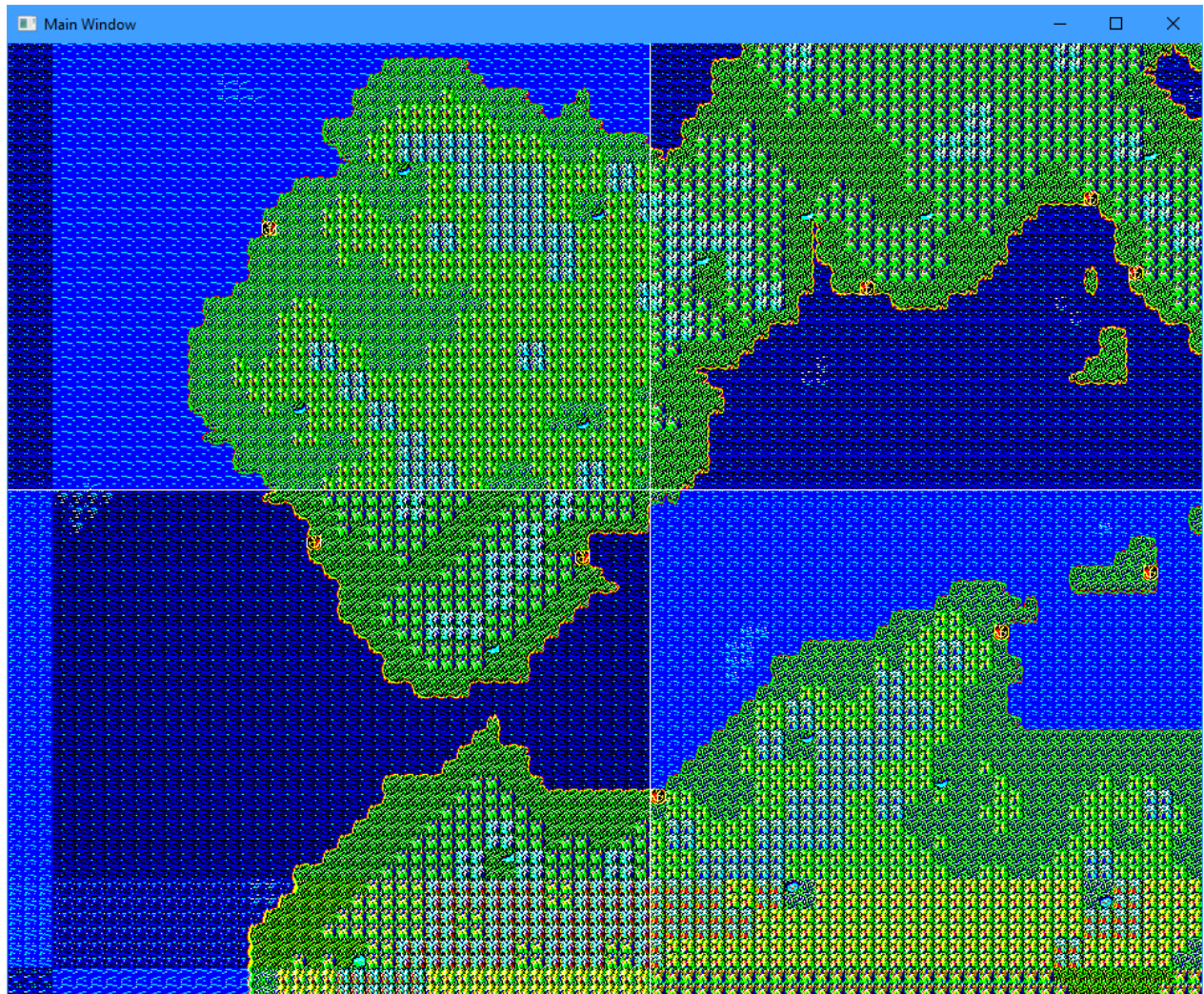



Fig. 4: Half-size Rendering.

 freefood1.andrew.cmu.edu

24-780 Engineering Computation Compile Server

Please make sure your source code can be compiled without error with this page before submitting to the Black Board.

This page helps you identify compiler-dependent features by compiling your program with Visual C++ and GCC. If you are seeing no error does not mean you receive full credit. You are responsible for understanding and complying with the specifications.

CAUTION: Test-compiling your source code does not submit your code to the Black Board!
After testing and making sure your code is error-free, submit your code through the Black Board.

[Go To Blackboard](#)

Source File 1 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	ps3.cpp
Source File 2 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	data.h
Source File 3 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 4 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 5 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 6 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 7 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 8 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.
Source File 9 (.c, .cpp, or .h)	<input type="button" value="Browse..."/>	No file selected.

Fig. 5: Freefood Server

We have multiple servers to make it less likely that all of them need to shut down for maintenance. If do not have to test on all of the servers. You need to make sure that your code passes on one of the servers.

In this assignment, you need to select ps3.cpp and data.h and click on the Compile button as shown in Fig. 5.

7 Submit

Lastly, you need to submit using git. What you need to do are two things: (1) add files to Git's control, and then (2) send to the git server.

7.1 Add Files to git's control

In this case, you want to add all the files under ps3 subdirectory. To do so, type:

```
git add ~/24783/yourAndrewID/ps3
```

This command will add ps3 directory and all files under the subdirectories.

7.2 Send to the Git Server

In Git, sending files to the server is a two-step process. The first step is local commit. You can do it by:

```
git commit -m "Problem Set 3 solution"
```

The message can be anything, but it is recommended to type something meaningful, at least you can see what changes you made to your repository.

Local commit is just local. Git server does not know about any local commit unless the commit is sent (or pushed) to the server. To do so, type:

```
git push
```

Make sure to do it in the CMU network. If you are working from home (probably most likely), use VPN to connect to the CMU network.

You can re-submit (commit and push) your solution as many times as you want with no penalty before the submission due.

8 Verification

Clone your repository to a different location and make sure that all of your files have been sent to the Git server.

You can do the following:

```
cd ~  
mkdir 24783Verify  
cd 24783Verify  
git clone https://yourAndrewID@ramennoodle.me.cmu.edu/Bonobo.Git.Server/yourAndrewId.git
```

Once you made sure all the files have been submitted, you can delete files and directories under 24783Verify directory.