Chapter 1

CS488/688 W12 A1: Introduction

> *"I think this assignment should have 8 objectives but we should still mark it out of 10."*
>
> – The Mean TA.

This assignment is due **Wednesday, January 18th** [**Week 3**].

1.1 Topics

- Exposure to OpenGL
- Callback-based program structure
- `gtkmm` user interfaces

1.2 Statement

This assignment will get you started writing graphics applications using OpenGL. It will also familiarize you with the set of languages and APIs we will be using for subsequent assignments. You will be writing a user interface in C++ using the `gtkmm` UI toolkit. The user interface will be wrapped around a window in which graphics will be rendered using OpenGL from the C++ application.

In particular, the program is a game in which the user must manipulate falling tetrominoes so as to make complete lines at the bottom of a well. When lines are completed, they are removed from the game. When the well fills up, the game is over. Any resemblance between this game and a popular arcade game of Russian extraction from the 1980s is purely coincidental.

You will also implement some graphical functionality not directly related to game play, but that will help you develop OpenGL skills needed in later assignments.

1.3 The game

The game takes place in a U-shaped well of unit cubes enclosing a grid of width 10 and height 20 in which tetrominoes can fall. The blocks occupy discrete positions in the grid (they don't fall smoothly, but jump from position to position).

Tetrominoes start in a four unit tall stripe on top of the well. Every time a predetermined interval elapses, the current tetromino falls one unit. A value of 500ms is a good novice interval; 100ms is more challenging. At any time, the current piece can be moved to the left or right, rotated clockwise or counter-clockwise, and dropped the rest of the way down the well. When the piece can fall no further, it stops, and any rows in the well that are completely filled are removed from the game. The game ends when a piece cannot clear the starting stripe.

As the game progresses (ie, as more filled rows are removed from the game), the pieces should fall at a faster rate. You should have at least three different speeds.

The game should be drawn from unit cubes. The well and the various piece shapes should all be drawn in different colours.

1.4 The interface

The user interface should be written in `gtkmm`, a C++ wrapper for the popular `GTK` user interface toolkit. You will need to implement the following functionality:

- A `File` menu with the following menu items:
  - `New game`: Start a new game. Keyboard shortcut `N`.
  - `Reset`: Reset the view of the game. Keyboard shortcut `R`.
  - `Quit`: Exit the program. Keyboard shortcut `Q`. (This one should already be implemented; be sure not to break it.)
- A `Draw Mode` menu with the following menu items:
  - `Wire-frame`: Draw the game in wire-frame mode. Keyboard shortcut `W`.
  - `Face`: Fill in the faces in the game. Each different piece shape should have its own uniform colour. Keyboard shortcut `F`.
  - `Multicoloured`: Similar to `Face` mode, but each cube is required to have a different colour on each of its faces. In this mode, you're allowed to assign colours however you like, as long as no cube has two faces of the same colour. Keyboard shortcut `M`.
- A radiobutton `Speed` menu with at Slow, Medium, and Fast speeds that sets the rate at which pieces fall. The game may use additional speeds, but you need to be able to set the speed to one of the three, and as the game speed increases, it should automatically update the speed radiobuttons as approrpiate.
- A `Buffering` menu with the following menu item:
  - `Double buffer`: Toggle double-buffering. This should be a check item.

  Note that the `Draw Mode` menu should use radio buttons to indicate which state is selected.

  The keyboard shortcut `B` should toggle the buffering mode.
- Mouse movements:

- Mouse operations should be initiated by pressing the appropriate mouse button and terminated by releasing that button. Only motion in the horizontal direction should be used.
- The left mouse button should rotate the game around the X axis.
- The middle mouse button should rotate the game around the Y axis.
- The right mouse button should rotate the game around the Z axis.
- When the shift key is pressed, all mouse buttons should uniformly scale the game (both the board and the pieces). When the mouse moves to the left, the game should become smaller, and when the mouse moves to the right the game should become larger. The maximum and minimum scales should be restricted to a reasonable range.

You must make reasonable decisions about how much to scale or rotate for every pixel's worth of mouse motion. For example, if the mouse isn't moving, there should be no scaling or rotation.

You are also required to implement a feature sometimes known as "persistence" or "gravity". If, while rotating, the mouse is moving at the time that the button is released, the rotation should continue on its own. This decision should be made at the time of release; after that, it should persist independently of mouse movement, until the next button press.

- Game play:
  - The left arrow key should move the currently falling piece one space to the left.
  - The right arrow key should move the current piece one space to the right.
  - The up arrow key should rotate the current piece counter-clockwise.
  - The down arrow key should rotate the current piece clockwise.
  - The space bar should 'drop' the piece, sending it as far down in the well as it will go.

1.5 Gtkmm and OpenGL

In this course, user interfaces are written in `gtkmm`, a C++ wrapper around the C-based GTK toolkit[1]. We also use `gtkglextmm`, a library providing OpenGL support from within GTK, and again a C++ wrapper around a C library.

In the C++ code you write for this assignment, you won't need to use any OpenGL commands apart from these:

- `glEnable`, `glDisable`
- `glBegin`, `glEnd`
- `glVertex`
- `glNormal` (maybe)
- `glRotate`

[1] `gtkmm` used to be called `gtk--`, but has since changed its name due to search engine indexing problems.

- glScale
- glTranslate
- glDrawBuffer

(Of course, you may find that you will want to use additional OpenGL functionality to add extra features. Note that the Matrix4x4 class stores its matrices in row-major order, but OpenGL expects matrices in column-major order.)

1.5.1 Double buffering

Your program should support both single and double buffering. In single buffering, graphics commands draw directly onto the screen. In double buffering, drawing happens on a "back buffer" that is then swapped with the "front buffer" when drawing is complete. Because the swapping happens quickly, the user never sees a half-complete rendering.

The OpenGL command glDrawBuffer lets you choose which buffer to draw into. The gtkglextmm widget allows you to swap the buffers, and this is already implemented for you in viewer.cpp

1.6 The Skeleton Program

If your account is correctly set up, you will find a skeleton program in the cs488/A1/src subdirectory of the source distribution. The program creates a user interface with an OpenGL window. As a test, it draws triangles where the corners of your game (not including the well) should appear. The camera is set up so that the triangles appear centered and correctly sized. You need to modify this code to render the current state of the game and respond to user interface events. Here's a to-do list, with a suggested order that will help you make your way through the assignment.

- Write a function to draw a unit cube using OpenGL. You can test it by modifying Viewer's on_expose_event member function to draw a hard-coded set of unit cubes. There are two ways to draw cubes:

  - If you're already familiar with OpenGL, you can write a single drawCube function that draws a cube at the origin, and use glPushMatrix, glPopMatrix, and glTranslate to place each cube in the game.
  - You can also just write a drawCube function that takes the cube's location as a parameter and draws the cube directly in that location.

  Note that although the GLUT library includes a function to draw a unit cube, you can't use it in this assignment. Your cube should be composed from calls to glVertex.

- In your render function, draw a U-shaped border for the well out of cubes.
- Implement face rendering and wireframe rendering.
- Implement rotation and scaling. You should be able to see the effect on the well.

- Implement single- and double-buffering. You should be able to see the difference when rotating the well.
- Add code to draw the current contents of the game. Each piece type should be drawn in a different colour; the colours are up to you.
- Hook up a simple timer event (using the `Glib::SignalTimeout::connect` method) that calls down to the game's `tick` method and re-renders. You should be able to see pieces falling.
- Implement the rest of the controls for game play and the remaining user interface details.
- Acceleration. The more rows you complete, the faster pieces should fall. After every n rows, you can simply decrease the delay between successive calls to `tick`.

1.7 Donated code

The skeleton program comes with the following files:

- `main.cpp` – The entry point for the program.
- `viewer.hpp`, `viewer.cpp` – The OpenGL widget. All of the OpenGL-related code is here.
- `appwindow.hpp`, `appwindow.cpp` – The application window code. Most of the UI-related code (menubars, etc.) is in these two files.
- `game.cpp`, `game.hpp` – An engine that implements the core of the falling blocks game.
- `algebra.cpp`, `algebra.hpp` – Some basic geometry and colour classes useful throughout the course.
- `Makefile` – Used to compile the program with `make`.

You should be able to get the skeleton program running using the commands `make; ./game488`.

1.8 Deliverables

These executable files should be put in the directory `cs488/handin/A1`:

- `game488` – The program executable.

All source files should be in the directory `cs488/handin/A1/src`.

1.9 Hints, tips, and ideas

There are lots of ways this simple application could be modified to enhance playability and attractiveness. You are encouraged to experiment with the code to implement these sorts of changes, as long as you have already met the assignment's basic objectives. Here are some suggestions:

- A scoring mechanism.

- Head-to-head networked play.
- Modified cubes for pieces. The blocks look much better if individual cubes have their edges slightly beveled.
- Animations for certain events. The board can spin around when you lose, for example.
- Add lighting.

If you make extensive modifications to the game, you should make sure to run in a "compatibility mode" mode by default – you should support at least the user interface required by the assignment. You can activate your extensions either with a special command line argument or a menu item.

1.10 Objectives: Assignment 1

Due: Wednesday, January 18th [Week 3].

Name: ____________________

UserID: ____________________

Student ID: ____________________

___ **1:** Wireframe mode works.

___ **2:** Face colour mode works.

___ **3:** Multicoloured face mode works.

___ **4:** Acceleration works at three or more speeds.

___ **5:** Both single- and double-buffering are supported.

___ **6:** The user interface works as specified (menus, mouse interaction, etc).

___ **7:** The game can be rotated.

___ **8:** The game can be scaled.

___ **9:** The game is playable (i.e., you can move the pieces as described under ”game play” of the assignment specification).

___ **10:** Persistence works for rotation.

Declaration:

I have read the statements regarding cheating in the CS488/688 course handouts. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.

Signature: