CPSC 3710 Computer Graphics Project

"Daft, the Group" Members: Jordan Peoples, Chad Klassen, Adam Shepley Jason Racine

How to compile and run the project

1. Copy the game folder from the CD onto your desktop.
2. Open the folder in a terminal.
3. Type in make (ensure that you have G++ installed on your test machine)
4. Wait for the program to compile.
5. Type in "./detroitrockrobo", and press enter.
6. Program should open and feel free to start testing.

Distribution of work among members (referencing our SVN log)

It's important to note that we did not assign specific roles to group members in this project; we went for a more agile development scheme wherein we'd announce what we intended to work on and then report our progress or ideas at the next time we met.

Adam Shepley:

1. Modified city scale and implemented an early version of diffused lighting
2. Changed redundant road rendering to 1 flat pane for entire city side.
3. Refined the direction arrows to work in all directions.
4. Added road markings across road to indicate the possible rail directions
5. Added a numerical identifier and object-directed name to every building, along with accessors and the algorithm to assign them uniquely at object creation time based off of city block and building creation number.
6. Experimented with early lighting and defined a base material specification
7. Modified and expanded upon this document (secretarial work)
8. Fixed the 'turn' arrows at crossroads to rotated when the robot rotates

Chad Klassen:

1. Created and rendered the robot class using a variety of primitives.
2. Created, implemented and repeatedly refined a camera class for both implementations based off of the robot (the main one that remains) and those with the camera as the focus (deprecated and not present anymore).
3. Designed the camera rotations and their multiple applications.
4. F1 – F8 functionality; ensured that the camera movements and placements corresponded to the proper keys at the proper times, as well as creating movements and animations for the robot (camera movement & placement)
5. Added rotations to the robot itself.
6. Corrected and ensured that the movements and rotations of the camera and robot corresponded to each other in all instances and camera angles

Jason Racine:

1. Building colors – created and tweaked the individual color settings for the buildings and building types, and material settings where needed
2. Established the lighting setup for the city such that the existing colors for the building types were respected
3. Created the “rectangle building” class, which was then used as a basis for the majority of buildings (via scale) of a rectangle/square type
4. Created the City Block classes, which contain their ground patch and can be slotted with up to 4 buildings (technically more, but we have a logical limit of 4)
5. Created the equivalence (==) and assignment (=) operators for the greater Building class
6. Added details to the robot (the flat triangle polygons, the reflective coloring)
7. Implemented arrows that appear at the “moveable” crossroads (and later, resized them properly)
8. Created an algorithm to ensure the random assignment, creation and placement of buildings per block
9. Corrected the final keyboard key mappings for the program
10. Re-factored the building creation into individual objects, rather than single objects with a pointer per block

Jordan Peoples:

1. Created a “city manager” singleton-like class (basically the City as a whole abstract piece) which allowed the attachments of City Blocks and facilitates the adjustment and access of particular building information from anywhere in the program
2. Created a block-based road system (later replaced by a single tile for simplicity's sake; though Jordan's method is technically superior)
3. Fixed up the Robot class' rotations after it was implemented in our 'main' file
4. Attached the Robot to our roads/railing and created an indicator for it reaching a crossroads (making the skybox turn blue)
5. Added backwards movement to the robot (though not in the description, it greatly assists in navigation and bug testing)
6. Created multiple Building subclasses (these are what you see actually being rendered) such as a “Pylon” (inside joke building, but it looks cool), Pyramid, Sphere and Torus (Torus is not used, but functional. We were going to use it to make “bridge”/tunnel buildings)
7. Created the base version of this document based off of SVN repository comments
8. Fixed the final placements of the roads (boundaries) and implemented the 'reset' functionality

All:

1. For the picking and selection, everyone made some attempt in unique ways to correct the issue, but most of the tries netted no large result
2. General bug tracking and correction; a large amount of the functionality was white boarded before and during implementation for the group to dissect and correct design problems

3. This document was typed and revised by the general group during discussions
4. General research and program logic ideas were derived from group discussions and dissections primarily

Special data structures used

Our buildings are housed in a special Building vector type, and are created using a virtual abstract Building superclass and one of its (specified only at creation time, and not when calling member functions) children classes.
We also have a “City Manager” (city) type (not a data structure so much as a class that holds other classes) that...manages the city.

OpenGL learned outside of class

- We learned that OpenGL has specific pre-requisites for compilation that require specific libraries from repositories (versions of glut)

- We learned that implementing pick and select in a 3D environment is several orders of magnitude more difficult that doing so in a 2D environment; (at that, we could not complete the implementation properly)

Assumptions made on/about the project

- User must be running Linux, and be familiar with commands in the Linux terminal.
- User will not drive the robot out of the city limits, as no boundaries are in place (not a requirement).
- User will have the G++ C++ make/compiler installed
- Users will not purposely try to disrupt the movement animations
- (Our animations are deliberately slow to make them look better, but we did not want to restrict freedom of movement, so it's possible to move while rotating)

Other documentation

When the robot is able to turn in the intersection, the sky will turn red as a indicator, as well as arrows will appear near to the robot indicating which was you can turn.

These indicators will appear OUTSIDE the city as well, and we allow the robot to move backwards with the '**S**' key as this makes it far easier to navigate the city (as the 1-pixel requirement specified makes it really easy to miss a turn point and having to go all the way to the next block is quite difficult for a novice user).

We could not finish the “building destruction” pick and select option; We can somewhat destroy buildings or groups of buildings, but the issue is with which buildings are being selected. Also, there is immense slowdown when pressing.

You can rotate the robot with LEFT and RIGHT arrows independently of the movement across and near the crossroads.

Our rotation markers are sometimes perturbed by the Return option key/'r'.