Project 6
Designing a Better Medieval World

Two-day bonus: Submit all deliverables by 5:00 PM Wed. Apr. 15, 2009
One-day bonus: Submit all deliverables by 5:00 PM Thu. Apr. 16, 2009
Final Deadline for all deliverables: 5:00 PM, Fri. Apr. 17, 2009.

Notice:

Corrections and clarifications posted on the course web site become part of the specifications for this project. You should check the this page for the project frequently while you are working on it. Check also the FAQs for the project. At a minimum, check the project web pages at the start of every work session.

Overview

The purpose of this project is to provide further practice with design using the concepts presented in the course. Your task is to choose some "features" to be added to your Project 5 solution, work out a detailed specification of the features, and then design, code, and test their implementation. This project is to be programmed in Standard ANSI C++ and the allowed tr1 extensions, and take full and appropriate advantage of the Standard Library. More importantly, you are expected to use the design and programming concepts and approaches presented in this course. Since good design is much harder than good coding, you need to start thinking about the design of this project days before you start coding - procrastinate on the coding if you must, but procrastinating on the design will probably be fatal!

The project will not be autograded for output correctness because you control what the correct output is supposed to be. We will also not test your components because you control what the components and their interfaces are. However, you must submit a working version of the program and a demonstration script that allows us to use your new features and verify that they work. In addition, you have to supply some design documents that explain your design for the additional features. These *deliverables* are described below.

Each feature is only described at the level of an "idea" - like something the marketing people might come up with. You have to specify the feature - work out the details of the program feature and how it should behave - as well as how to design and code it.

You have to choose two features: One is simply an extension to, or elaboration of, the existing Project 5 structure, a *simple extension*. The second is a *major enhancement* that requires a significant amount of new or modified structure. You have to choose one of each type from the following lists. Teams have to choose more features (see below).

Simple Extensions

Option A. Another kind of agent. Explore the generality of the Project 5 framework by designing and implementing a new class of agent. Any modifications to the existing framework and classes should be in the direction of making similar future additions easier. This new class must have the following characteristics:

- It may not be simply a clone or near-clone of the existing three classes - it must behave differently in a significant way. For example, a new class that is simply a stronger Soldier will not do.
- It must display “automatic” behavior at least as much as Peasant or Archer do, and definitely more than the current Soldier, which is pretty boring.

Option B. Save/restore. Give the user the ability to save the state of the simulation to a specified file at any time. At any time, the user can restore the simulation from a specified file - the result is a simulation that is in the exact same state as it was at the time of the save. This must be done in such a way that if the program is expanded, e.g. by adding another type of Agent, it will be easy to include it in the save/restore system with little or no modification to existing code.

Option C. Destroyable structures. Make it possible for warrior-type Agents to attack and destroy Structures, which disappear from the world like killed Agents do. This must be done in a way that makes it simple to add new kinds of structures and have them be destroyable as well.

Major Enhancements

E1. Multiplayer game

Project 5 was in the form of a simulation, in which the user could issue commands to any Agent, and had available a single map view of the entire world and various local and other views. This new feature is to change the simulation into a multiplayer simulation game in which any number of players would take turns; the "tick" of the simulated time clock happens only after all players have said "go." Each player has Agents that only he or she can command, and Structures which belong to individual players. Each player has their own set of views which are open only during their turn (we'll assume that players do not try to look at the display while some other player is taking their turn). Players can leave or join the game at any time, and any number can play and have any number of objects or views. You can add additional game-like features if you wish, but the ones just described are required for this option. As examples of additional game features, players might have additional kinds of resources such as money, or players might have to have some amount of these money or food in order to build or train new objects.

Design goal: It must be possible to add additional Agent, Structure, or View types with no change to the multiple-player code. Even if you don’t add any additional game-like features, the framework must be in place to add such features or other capabilities to players

with little or no change to the rest of the code. While actually implementing a computer-controlled player is well beyond the scope of the project, a good solution will have an obvious place-holder for such a capability.

E2. Groups of Commandable objects

Provide ways for the user to put Agents into groups and control the groups as a whole, in addition to the existing individual Agent capability. Those of you who have played games such as Warcraft will recognize this sort of feature: You can form units into groups at any time and move or command the group as a whole, and any unit can be removed from the group and controlled individually, or groups disbanded, at any time. Any number, kind, or mix of units or groups of units can be included in the group. This feature requires you to make decisions about how agents in a group should work, move, attack, be displayed, etc. At present, only Agents are commandable, but in the future we might want to have commandable Structures, such as Forts that can fire at designated targets, or Training Schools that produces specified kinds of Agents.

Design goal. There should not be any arbitrary limits, either current or future, on what kind of Agents or commandable objects can formed into groups and controlled as a group. If a new Agent class is added in the future, it must be possible to treat it as a member of a group with no changes to the group control code. A good solution will easily extend to other kinds of commandable objects besides Agents.

E3. Multiple Object Families

Provide the capability to switch between different families of Sim_objects, while the program is running (this is expected to require that the previous game/simulation be terminated.) You must demonstrate this with at least one more family of Sim_object classes in addition to the P4/P5 set, and it must include at three kinds of objects. Because we don't want to force the user to learn a whole new command language, this other family should respond meaningfully to more or less the same set of commands, but otherwise be different in some interesting ways, not just straight "clones" of the current Structures and Agents.

Design goal: The mechanism to switch between families of Sim_objects must allow for easy extension to any number of future families of Sim_objects and classes within each family, within the constraint that the command language applies to each family.

E4. Selectively Responsive Agents

Currently either the Agents do not respond to each other, or interact with each other in very simple ways. We want to have Agents that interact with each other in more interesting ways. For example, suppose we have two Agents, and one attacks the other. Each Agent behaves differently depending on the type of its opponent. For example, if a Foot-Soldier is attacked by a Archer, it closes in to counter-attack, taking advantage of its superior strength and protection. But, if attacked by a Knight, it runs away to hide. Similarly, a Knight might always counter-attack a single attacker, but run away if there are multiple attackers. (Feel free to create a different set of relationships - this is just an example). All least three different kinds of Agents are required that can interact with each other in ways that depend on which types are interacting. Warriors interacting with each other is an obvious possibility, but you can choose another possibility. It is fine if one of the Agent types is the one added in Option A. You can certainly modify the behavior of existing Agents to provide a better demonstration.

Design goal: The approach taken to provide this feature must first, be general enough that any arbitrary pattern of interaction could be supported - for example a simple distinction based on a single attribute such as attack-strength or cowardice won't do. We want to be able to support completely arbitrary interactions. Second, the approach must easily extend to additional types of agents in a straightforward way. For the above example, if we add a fourth type of warrior, then it must be possible to easily implement its interactions with the three existing types of warriors. This needs to be done in a way that gives a good tradeoff in the various issues involved, and follows well the guidelines of good OO design.

E5. Agents that are Aware of What's Happening

Project 4 has agents that are aware of being attacked by other agent, thanks to a direct interaction between the two. Project 5 had an agent, Archer, that was aware of the object locations and so could pick a target or refuge. However, in general, the agents don't know what each other is doing in any more general way, and so can't behave very intelligently. This enhancement is to allow the definition of agents that know more about "what is going on" beyond what we provided for in Projects 4 and 5. For example, suppose that Peasants could automatically tell when a farm has an excess amount of food and start collecting from it. Maybe a Peasant could bring food to a Soldier who announces that it is hungry. Maybe if there is a fight going on in the vicinity, an Archer could move close enough and join in. Again, these are just examples to illustrate the concept, not specifications of what you have to do. The idea is that if a Sim_object announces a change in state, or an activity, then agents can respond as suits their type.

Design goal: The approach taken to provide this feature must easily extend to both additional types of Sim_objects, and additional kinds of state changes, activities, or announcements in a straightforward way.

What You have to Achieve in the Major Enhancements

The major criterion for a good design is whether the program can be easily extended, the hallmark of good OO design. This includes not just the clarity and simplicity of the code, but also whether additional classes or subclasses of the various types can be easily added to the extend the new capabilities of the program, and in a way that requires little or no modification of existing code - "adding functionality by adding code, not by modifying code."

Each option contains a concise statement of the design goal, which involves developing a general capability of some sort that will support future additions or modifications along the same lines. Your problem in the project is to (1) achieve this general capability with an extensible design, and (2) demonstrate that it works with some specific example implementations.

Two pitfalls to avoid: (1) Implementing a design that does your example cases in the minimal possible way without providing a clear extension pathway - in other words, failing the design goal. Such a project will get an extremely poor evaluation. (2) Running amok with example implementations, such as a dozen new kinds of objects for E3 - this will be a waste of time and won't make up for a defective design, no matter how clever the examples are.

So keep the priorities clear: The primary goal of the major enhancement is to *provide a general capability for extensions of a certain kind*. The specific implementations are secondary, and are required only to demonstrate the scope and power of your general capability. In other words, many clever specific implementations in a poorly designed framework will be worth very little, while a few well-chosen implementations that demonstrate the scope and power of a well-designed framework will be considered an excellent result. Please ask if this is not clear.

Other Rules

What You Can Change

The required design elements and components of Project 5 must be present in this project, because, as described for Project 5, this is the second part of a two-part project. To be clear, these are the Model Singleton, pervasive use of smart pointers, the MVC organization, the four kinds of views, and the two kinds of warriors. As long as these design elements are present, you are free to modify the Project 4 and 5 classes as you choose. Where applicable, the project should be fairly close in its behavior to Project 5 - this is not a new project, but an extension of the previous ones. All previous features should continue to be present, although details might be different and they might be implemented differently. Thus a complete rewrite is not called for, and is almost certainly not needed, but you can make any changes that will help you achieve a good design in this project. Ask for clarification if you aren't certain about this requirement.

Teams

Object Oriented Programming can work well with development teams. The team members first come to agreement on the responsibilities, collaborations, and public interfaces of the classes in the design, then divide the classes up between the team members, and then each team member develops the private implementation for his or her classes. If done properly, the separate classes will plug-and-play together, and changes and refinements to the design can be easily worked out and implemented.

If you want to try this out, you may form a team of up to three to perform this project.

Your team has to start with a Project 5 implementation. You can choose a single solution authored by one of the team members, or you can combine pieces from more than one member of the team. You should review and compare your solutions, pick the best or best parts, and fix any problems you identify, because the Project 5 problems will be human-evaluated as part of Project 6, and all team members will get the same score.

A team must supply an additional document on how the design, implementation, and documentation work was handled as a team. It is expected that each member will make substantially equal contributions to the project, and all will write code.

- An individual person must do one of (A , B, C) and one of (E1, E2, E3, E4, E5).
- A team of two people must do two of (A , B, C) and one of (E1, E2, E3, E4, E5).
- A team of three people must do either: three of (A , B, C) and one of (E1, E2, E3, E4, E5), or one of (A , B, C) and two of (E1, E2, E3, E4, E5).
- Teams of more than three are not permitted.

Choose one member of the team to be the "lead" for submission purposes. The lead member will submit the code to the autograder.

Autograder Deliverables

1. Your **source code and a makefile** will be submitted in a way that we will announce - we will be using the autograder simply as a way to send us your code and do a check compile of it. The lead team member will submit the code. To avoid confusion, other members should not submit any code. Your code must compile and run without error in gcc 4.x, using a makefile that you supply, and must be complete as submitted - we will not supply any files of our own.

2. A **command script input/output text file(s)** must be submitted along with your source code, to help demonstrate your new features. These must be suitable for I/O redirection, along the same lines as the sample files that have been provided in the course. These files should correspond to the annotated hardcopy console script file, explained below. We will use these to help determine if your program really does work "as advertised."

Our goal in running your program with (and without) your scripts will be to assess whether your program actually does the things you specified. You will lose credit if there are problems that cause us inconvenience or prevent us from compiling and running your code, such as compile errors due to non-standard code, missing files, or run-time errors that interfere with running the program. We will not attempt to fix your code, nor contact you about missing files.

Hardcopy Deliverables

You must submit some hard-copy paper documents in addition to your code, and in fact the quality of these documents can be more important than the reliability of your code - plan plenty of time to prepare them! They are:

1. A **hard-copy feature description document** which identifies the features that you chose (A, B, C, E1, E2, E3, E4, E5) and the details of your feature behaviors and how to use them. These are essentially your specifications for the new features. This should resemble the Project 5 specifications in level of detail and approach. In addition, this document must make it clear for the Major Enhancements what range of variation your new feature will support. For example, for feature E2, what are the limits on the kinds of actions that a group can be commanded to do? What is the scope of possibilities? Length: about 1 page/feature.

2. A **hard-copy annotated command script document(s)** similar to the *console sample*s in previous projects, showing the input and output of your program. The script should be annotated on the hard copy with explanations of what is happening - the annotations should be written by hand on the hard-copy. The submitted script files, described as Autograder Deliverable #2 above, are the corresponding input/output files that we can use for redirection. By running your program using the script files, and examining its behavior and this hardcopy document, we should be able to see your new program features in action, and get a complete demonstration of its capabilities, and understand how they work. You can have one script document per feature, or combine them as convenient. Length: as needed. The annotations should be hand-written on the hard copy.

3. A **hard-copy design document**, that explains the design of your program with regard to the new features. Your document can assume the parts of the design that are identical to Projects 4 or 5 as described in the specification documents; do not waste time explaining what is in those documents.

The purpose of this document is that by reading it, we should be able to easily understand the structure and organization of your code for the new features. The most important part is how your design supports easy extension. Your design document should include the following sections:

- *Design of Simple Extension*, including any changes needed from the Project 4/5 design. The behavior of the Simple Extension is described in the feature description document and must not be repeated here. Length: about one page.
- *Design of Major Enhancement*.
  - Design description, including any changes made to the Project 4/5 design. The behavior of the Major Enhancement is described in the feature description document and must not be repeated here. Required section: What would be needed to extend the project along the lines stated in the design goal for the enhancement. Length: 3-4 pages, not counting diagrams.
  - UML Class diagram showing structure of new classes in relation to Project 4/5 classes.
  - UML Sequence diagram showing an informative and typical interaction between objects in the new design.

The diagrams can be hand-drawn, and should follow the presented notation. For examples of the kind of information and level of detail expected in the design document, look at the "Basic Architecture" and "Class Responsibilities, Collaborations" and "Model-View-Controller Interaction" sections of the Project 4 specification document.

4. A **hard-copy team activity document** must be supplied for a team project. It lists the team members, describes whose version of Project 5 you chose as a starter, how you arrived at the project design as a team, and which team member was responsible for what work in the project. Length: 1 page.

Hardcopy Deliverable Submission Rules

The hard-copy document deliverables must be supplied in hard copy - soft copies of the documents will be ignored. Your project submission will be considered complete only when you have turned in all of the deliverables. Bonus days apply to score for this project, even though there is no autograder score component to the grade. Bonus days will be counted from the date when all deliverables have been submitted.

The hard-copy documents must be separate, each with its pages stapled together, and individually identified (if a team project, by the names of all team members, lead member listed first). The documents must be enclosed in a 9 X 12 clasp-closing envelope. University Campus Mail envelopes are Not Acceptable. Write on the outside of the envelope (1) your name and uniqname; for a team, all the team members, lead member first; (2) the identifiers for which features you have chosen (A-C, E1-E4) to help us sort the projects for evaluation.

You must deliver this envelope to either the professor or the GSIA *in person* at times and places to be announced.

If the hardcopy deliverables are not submitted according to these rules, we are not responsible for them and will refuse them. The requirement for a new envelope and stapled documents might sound mickey-mouse, but it is a serious one. This requirement is imposed because trying to keep track of dozens of loose or awkwardly packaged documents is a major hassle, unnecessarily driving up the effort and blood pressure of those doing a very difficult grading job under time pressure. Therefore, we will not be responsible for keeping track of the pages of unstapled documents, or submissions not given to us in person. Any submission not in an envelope, or inconvenient to handle, *will be refused - really!* Find a stapler now, and purchase an envelope now! Don't wait until the last minute and come with a grossly expensive presentation folder or a trashy envelope salvaged from a dumpster.

Project Evaluation

The quality of your solution will be based on your specific definition of features, and how well your program meets the design goals, as illustrated by your implementation and as explained by your deliverable documents, and manifested in the structure of the code. The general code quality will be evaluated similar to how it was done for Projects 1 and 3, but will count very heavily in the total score.

As already stated, your design and your code should reflect the advice and concepts presented in the course. A good selection of specific features, a good design that is general and extensible and takes advantage of the course concepts, good quality code, and clear and informative documents, should result in a high grade.

Warnings

How to get a poor grade on this project: new features that are minimal in scope, a design that will extends only poorly or not at all past the specific cases that you implemented, documents that are unclear or uninformative, code of poor quality, and a design that looks like you blew off the second half of the course.

Poor code quality will hurt more than in previous projects. Before finalizing your code, go through the Coding Standards document and the OOP design lecture notes and check each item or guideline against your code.

Good documents are critical - if we can't understand what you have done, or how it is supposed to work, we will not bother to fiddle, puzzle out, or guess. If the documents are inadequate and do not help us work with or understand your project, we will simply ignore the rest of your project work and give it a zero or poor grade. Check each document against the deliverables specification and make sure it includes the requested information.

It would be better to submit buggy code than inadequate documents, so allow time to develop the documents. Drafting the feature description and design documents before you start coding will actually help you work faster and better.