COMP 303

Winter 2025

Milestone 1

Posted: Monday, January 27^{th}

Due: Monday, February 10^{th} by 11:59 p.m.

Introduction

In this milestone, you will propose a space that will exist in our class's shared virtual world. You will describe your idea, the design patterns you will use and how you will attempt to implement them. The aim of the proposal is to give yourself a roadmap for the project that you can use for the remainder of the term, as well as a chance for the teaching staff to review your idea and propose changes if needed. (You can always modify parts of your idea after submitting the proposal, but please consult with the teaching staff before making significant changes.)

Requirements

- **Main Concept**

  In your proposal, describe the theme and purpose of your virtual space. For instance, is it a forest of monsters that you can catch or battle? A haunted mansion with puzzles to solve? A concert hall where different instruments can be played? Be creative! We provide some small beginnings of ideas later in this PDF that you can use for inspiration if you like.

  Your idea should be described in at least half a page (regular margins, 12 pt font, etc.). Make sure to describe in detail all the different actions that a player can take in the room, and explain the general flow or order of the actions the player can take once they enter the room. For example, in a concert hall, the player may walk over to an instrument, then interact with it to play certain notes, then walk over to another instrument, etc., and maybe talk with an NPC to save a recording of their song.

  Your idea must have a consistent theme. If you want to implement a bunch of different mini-games, for example, then you could use a casino or arcade as your main theme. Make sure to describe your theme in the proposal. You should keep your idea and theme consistent with the framing of our virtual world (i.e., a fantasy world without politics).

  Your idea must comprise both a back-end and front-end. The back-end will be the actual logic of your room, e.g., the board for a TicTacToe game, or the code to handle new map objects like pressure plates. The front-end will be the actual space into which players can move, including background tiles, decor map objects like plants or trees, map objects that the player can interact with or talk to (pressure plates, buttons, NPCs, boulders that can be moved, etc.). In your proposal, please discuss how you would want your room to look. You can also provide a sketch of it if you like.

  Note that the front-end must be interactive; that is, the player should be able to go into the room and do something by moving onto tiles (or some other mechanic that you could implement, like pushing or pulling objects). It shouldn't be that a player moves into a room and stands still, or only types commands into the chat window. See our class examples of a TicTacToe game (not interactive) vs. the Trivia House (interactive).

- **Design Patterns**

  In your proposal, you must list four design patterns you will use and how they relate to your idea.

  - Two of the patterns must be concretely explained: you must specify exactly how they are to be used for your idea. Please also provide a class diagram, extending the one that will be seen in class on Tuesday, January 28, and show what classes you will define for your idea and how they will integrate into the existing project architecture. (Please annotate on the diagram to indicate which classes will use which design pattern, if and when applicable.)
  - For the other two design patterns, give a brief argument (no more than two or three sentences per pattern) as to why they are appropriate for your particular idea.

  We provide at the end of this document a list of the design patterns from which you can choose as well as examples of how each can be used, and also a list of general project ideas and the design patterns that could be used for each of them.

  Notes on choosing patterns:

  - At least two of the patterns must be interconnected. For example, a Singleton controlling the state of multiple Strategy objects.
  - At least two of the patterns must be used in a way that significantly affects the player's experience (not just a superficial usage). All the project examples we list later in this PDF discuss patterns that can be used in a significant way to implement that example.
  - You can and should use the iterator and null object patterns if and when appropriate for your idea, but they will not count towards the four that you choose.
  - By the time the proposal is due, we will have covered four or five design patterns in class, including the Strategy and Singleton patterns. These two latter patterns come up often, and since we will have discussed them and you will thus have a bit of experience with them, it may be easier for you to choose these as your first two patterns (for which you must go into detail). For the other two, you could choose two of the remaining patterns that we will discuss later in the term (I suggest the observer pattern for one of the two, as it also comes up a lot). These are just suggestions, though -- if you have a great idea and don't think these particular patterns would be appropriate, you are free to choose from the others listed in this PDF, which we will discuss later in the term.

  Note: When actually writing the code for your project, you should keep in mind more than just the design patterns. We will also be looking at other things that we have/will discuss in class (encapsulation, unit testing, use of anti-patterns, etc.). We'll provide more detail about this in a few weeks once you receive the feedback for your proposal, at which point you can start writing your code.

- **Timeline and division of work**

  In your proposal, you must provide a brief timeline of what you will want to work on each week and which person in the team (if applicable) will work on what. The timeline you make here doesn't have to be strictly adhered to since things can come up during the term; you may have to work more in one week than another, for example, or you may find that one part of the project is a lot simpler than another and thus you may need to rebalance the work. But, by setting down a provisional timeline and division of work, it may help you to plan ahead. You don't have to give strict dates either, but try to indicate what will be happening per week.

  Here are some weeks with events that you should include in your timeline:

  - Week of Feb. 17 (late in week): Project proposal grades and feedback out; can start writing code.
  - Week of Mar. 17: Half-way point; check-in meeting with TAs.
  - Week of Apr 7: Final demos of project / code submission.

- **Brainstorming meeting**

  Your team must register for and attend a short, 10-minute brainstorming meeting with one of our TEAM Mentors before you submit your proposal. In the meeting, you will be able to talk about your idea and the design patterns you are thinking of using, and the Mentor can let you know what they think. If you are having trouble with your idea, they can also try to help you flesh it out. Please do arrive at the meeting with some idea.

  To register for a meeting, please leave a comment in an available, green cell on one of the two 'Brainstorming meetings' tabs in this Google spreadsheet.

- **Registration on server**

  Finally, you must also register your team name, emails and link to your private GitHub repository on our project server. We will show the steps for this in class. By following these steps, you will also be able to download the project code.

Grading

This milestone is worth 6% of your final grade, broken down as follows.

- 1%: Explanation of your idea.
- 3%: Explanation of two design patterns and class diagram (1.5 % each).
- 1%: Brief argument for two other design patterns (0.5 % each).
- 1%: Attending brainstorming meeting.

There is no grade for the timeline, but please make sure to include it or a penalty will be assessed.

We will also check the complexity of your idea when grading. When you receive feedback on your proposal, you may be required to add more complexity if your project falls below a minimum standard as decided upon by our TAs.

Submission instructions

We will use myCourses to submit. You will be able to go to the Assignments page and upload a PDF of your proposal. Only one submission per group is needed.

Help with the proposal

In addition to the brainstorming meetings, please feel free to come to our TAs or my office hours, and we would be very glad to assist you with your ideas. (Make sure to bring some kind of idea with you, though.)

If you have any questions about the proposal guidelines, please post on Ed.

Working with your team

This milestone is the first in which you will be working with your team (if you have one). I would highly suggest to try to be flexible with your team to get things off on the right foot -- that means to try to be accommodating with others' schedules and ideas for the project. If you have trouble coming to an agreement, there is always the possibility to come to a compromise. If you have difficulty with coming to a compromise, please see me or a TA/Mentor in office hours, and we can try to assist.

Next milestone

In the days after the proposal deadline, each student will receive the proposals of two other teams and will be required to submit a review of them within a one week period. You will receive the review guidelines at that time. Once all reviews are in, we will distribute them along with the TA's grades and feedbacks to each team; at that point you can start working on your project code. Do not start before that point, as we must first approve your idea.

Appendix: Survey of design patterns

Below is a quick reference of design patterns you may choose from and when we will discuss them in class. (Note that all dates are approximate.) We also provide an example of how each design pattern could be used as part of a project idea (note that there are many ways; these are just for inspiration).

- **Strategy** (Jan 21): Defines a family of algorithms, encapsulates each one, and makes them interchangeable from the client code perspective.

  Example: You could implement some kind of game where there are monsters chasing the player character, and there could be different strategies to do this (shortest path, random, etc.).

- **Flyweight** (Week of Jan 27): Uses sharing to efficiently support large numbers of fine-grained objects.

  Example: For any kind of objects, items or other content which may be duplicated, you may consider using this pattern.

- **Singleton** (Week of Jan 27): Ensures there is only one instance of a class.

  Example: You may need a global controller for your room's puzzle logic or special event management.

- **Composite** (Week of Feb 3): Composes objects into tree structures, allowing clients to treat compositions of objects the same as individual ones.

  Example: You may create a puzzle that involves objects that subclass MapObject; you may compose these objects into tree structures for flexibility.

- **Decorator** (Week of Feb 3): Dynamically attaches additional responsibilities to an object.

  You could create item objects which could be decorated with certain functionality.

- **Prototype** (Week of Feb 3/10): Creates new objects by copying an existing prototype instance.

  Example: When creating multiple NPCs with similar attributes, you can store a prototype NPC (with default image and dialogue) and clone it to quickly generate new NPC instances with slight modifications.

- **Command** (Week of Feb 3/10): Encapsulates a request as an object, supporting parameterization, queuing/logging and undo operations.

  Example: We already have the ChatCommand class, but a command doesn't have to be used only when the user types something in. They could step on a pressure plate, and that may activate a command, for example.

- **Template Method** (mid-March): Defines the skeleton of an algorithm, deferring some steps to subclasses.

  Example: A RoomGenerator could outline steps like 'place walls,' 'add decor,' 'insert NPCs,' etc., allowing subclasses to vary how each step is performed (e.g., a haunted mansion vs. a festive holiday room) but keeping the generation procedure consistent.

- **Observer** (mid-March): Establishes a one-to-many dependency so that when one object changes state, all dependents are updated.

Example: A Clock object might notify multiple objects in the room to update their displays (e.g., changing day/night modes, or triggering an NPC's schedule). All observers are updated whenever the clock's time changes.

- **Visitor** (late March): Represents an operation to be performed on object elements without changing the classes of those elements.

  Example: In a game world with different entities (items, NPCs, decorations), a CollisionVisitor might be used to handle how each entity type reacts to collisions. Rather than coding collision logic in each entity class, the visitor handles it for each entity in a separate place.

Appendix: Examples of project ideas

These are starting points for possible projects. Feel free to start off by choosing one if you like, and then expand it and flesh it out by providing the details mentioned in the requirements section above.

- **Monster forest**: Your space doesn't have to be an interior room! It could be a new map like a big forest, in which there are monsters or animals, perhaps themed after a popular video game, which you can catch and battle against each other!

  Design patterns: strategy (different ways to trigger an encounter with a monster, whether based on time of day or tile type); singleton and observer (day/night cycle singleton object could notify tiles to change).

- **Time puzzle**: A room with various objects in it that must be activated and/or collected/used in some order to solve a puzzle. By pressing a certain button, the player can toggle between two or more different time periods (could be day/night, or different historical eras, etc.); in each time period, the objects look and behave differently.

  Design patterns: observer (MapObjects are notified when the room shifts time periods); command (for the player's time-shift action); singleton (managing the time state); strategy (NPCs could behave differently depending on the time).

- **Arcade**: A room that has different mini-games, with a scoreboard and/or currency system that keeps track of the players' scores/money across all games.

  Design patterns: singleton (for the management of scores/money); strategy (different scoring mechanics or game loops, one per arcade game); observer (to notify the score/money manager).

- **Underground sewer**: Full of narrow walkways and flowing water; the water can rise and lower, opening or closing passages and creating a maze-like environment with different puzzles to be solved. The water can also affect objects in the sewer in some way (for example, it could cause objects like rocks, or maybe rats, to be swept away in the water to different places, or maybe it can cause certain objects to degrade, like paper).

  Design patterns: observer (water controller could notify sewer objects to block paths, etc.); strategy (the water could flow in different ways); commands (opening or closing a valve, etc.).

- **Festive simulation**: A room of a house that is themed after a holiday. For example, if you choose a Christmas holiday, you could have Santa Claus in the room, stockings along the mantlepiece which could be displayed in different orderings, and so on. (This idea was thought up by one of our TAs :)).

  Design patterns: singleton (one Santa Claus), strategy (stocking orderings).

- **Concert hall**: a large hall where players can make music by interacting with different instruments on the stage. Players can play by themselves or with others, and record their song for later playback.

  Design patterns: observer (a conductor observes each instrument being played and stores it away for later playback); flyweight (each instrument type references shared sound files or metadata, while individual instances store pitch modifiers); template (to define the steps which a player may take in

the room, e.g., choosing instruments, playing them, etc., and this can then be varied or specialized in subclasses).

- **Music Lounge**: Players can enter artist and song titles and listen to them and browse information about the newest releases by artists; if multiple players are present, they can vote on what track to listen to, etc.; playlist support, etc. There can be games like guess the song or the artist, and a scoreboard could track which players have the highest scores.

  Design patterns: Singleton (to manage the API requests); observer (to notify players when the current song changes or when a vote is queued up); strategy (in what order to play the songs in a playlist); command (user commands to skip a track, to vote, or to like a track and add it to their saved likes, e.g.).

- **Chatbot Lounge**: A room that interfaces with an LLM API so that players can ask questions or get whatever information they like (within reason). Different NPCs could walk around the room being controlled by the LLM. Different objects could appear in the room and their image and description could be generated on the fly by the LLM.

  Design patterns: Singleton (to manage the chat service); observer (to update the room's objects in some way according to the API's response to the player's question); command (to send the chatbot a message or regenerate an answer); decorator (to format the AI's text in some way); strategy (different chatbot prompting behaviours).