In the course project you will implement a significant extension to your minigolf game. The extension will be in addition to the requirements for Assignments #1 through #4 (that is, completing unfinished requirements from the assignments cannot be used to satisfy the course project requirement).

We offer four separate suggestions for the course project below. Fully implementing any one of the suggestions will satisfy the course project requirement. Note that we provide much less detail about how to implement each suggestion. Part of the course project’s increased scope is an expectation that you will complete both the design and the implementation necessary to satisfy the suggestion’s deliverables.

Alternatively, you can propose and implement your own minigolf game extension. In this case, you must first provide a write-up that explains what you plan to do for the course project, and details exactly what “deliverables” you will provide in your final minigolf game. We will approve plans that are of an appropriate size, and will mark the final result based directly on the deliverables you were able to complete.

**Suggestion 1: Network Multiplayer**

In this extension, you will implement the ability for multiple players to compete across the network. Games of this type normally work in the following fashion:

1. The first player starts the minigolf program, and offers to serve a network game. We name this player the *host*, and call his/her machine the *local server*. The process of waiting for remote players to identify themselves and join the game hosted on the server is called the *join phase*.
2. Remote players start the minigolf program on *remote client* machines, then join the game being served by the host by sending a request over the network to the server machine.
3. Once all the remote players have joined, the first player ends the join phase, and the *game phase* begins.
4. Once the game is completed, all players are shown the final scorecard and are allowed to enter high scores (if necessary). At this point the game phase ends, all players (on both the local server and remote client) revert to a “no game in progress” state.

As each player takes their turn, all the other players must be able to see what the active player is doing (*i.e.*, the current shot direction and shot strength as it is selected). Once the ball is hit, it should roll within the current minigolf hole in the same way on each player’s machine.

Some obvious issues that need to be considered include:

- What underlying mechanism should be used to allow the programs to communicate with one another (*e.g.*, sockets)?
- What types of messages need to be passed between the programs during the join phase?
- What types of messages need to be passed between the programs during the game phase?
- Do the programs need to communicate to guarantee consistency (*i.e.*, to ensure the players’ balls are all in the same place on the local and remote machines after each shot is completed)? If so, how should this be done?
- How should loss of communication be detected and managed?

The following deliverables are required for the Network Multiplayer extension:

1. Allow up to three remote players to compete from client machines connected over the Internet. IP addresses should be used to identify and communicate between players.
2. Ensure each player’s game runs in a consistent manner on both the local server and the remote client machines.
3. Provide some mechanism to allow a remote client to exit the game.
4. Provide some mechanism to allow the local server to terminate the game.
5. Provide some mechanism to detect and terminate a remote client if they become unavailable (*i.e.*, is no longer responding to communication requests).
6. Provide some mechanism to detect and terminate a game if the local server becomes unavailable to a remote client.

**Suggestion 2: Computer-Based Opponents**

In this extension, you will implement the ability to designate any of the players to be a computer opponent (this will occur in the “New Game...” dialog when the number and names of players is entered).

The main difficulty in this option is implementing appropriate AI logic to allow a computer opponent to plan an efficient path from the tee to the cup. On holes where you can see the cup from the tee, this is easy. On holes where you cannot see the cup, this is more difficult. You should use tile-connectivity information to plan your path. That is, you should try to find a connected path from the tile containing the tee to the tile containing the cup. This search is much more tractable, since explicit tile connectivity information is provided in the minigolf hole layout. For example, a brute-force search would compute all tile sequences that start in the tee’s tile and end in the cup’s tile, then select the shortest sequence as the path to try to follow to move the ball from the tee to the cup. Obviously, there are other considerations you might want to include when the path is selected and/or executed (e.g., how many turns does each path require, do some paths contain more flat tiles than others, how many tiles along the path contain dynamic objects, and so on).

A second consideration is how accurate to make the computer opponent. Theoretically, it is possible for the computer opponent to select a perfect shot direction and shot strength for each shot it makes. It makes more sense, however, to have different levels of computer opponent (e.g., beginner, advanced, or expert) that have different amounts of “accuracy” during game play.

The following deliverables are required for the Computer-Based Opponents extension:

1. Update the “New Game...” dialog to allow any player to be designated a computer opponent.
2. Implement path-planning logic to construct a ball path whenever a computer opponent starts a hole.
3. Implement the appropriate extensions to the interaction loop in your program to allow the program to manage computer opponents by selecting an appropriate shot direction and shot strength.
4. Provide at least three “levels” of computer opponent: beginner, advanced, and expert. This choice should affect the effectiveness of the path planning algorithm, and also how accurate a computer opponent is (i.e., how “perfect” the shot direction and shot strength are for each shot the computer opponent makes).

**Suggestion 3: Advanced Tile Types**

In this extension you will implement additional tile types to increase the complexity (and, hopefully, the enjoyment) of your minigolf game.

Currently, the minigolf game has a single tile type: grass. Other classes of tile types exist, and can be implemented to change the makeup of a minigolf hole layout. For example, the following game properties could be affected by certain types of tiles:

- **rolling friction**: different tiles might change the rolling friction coefficient, for example, “deep grass” or “sand” would have a higher rolling friction; “ice” would have a lower rolling friction.
- **shot direction**: certain tiles might not have a smooth surface, causing small, random variations in the shot direction as the ball moves across them, for example, “gravel” or “sand.”
- **hazard**: some tiles are hazards, and entering them causes an immediate end to the current shot (plus a “drop” of the ball to reposition it prior to the next shot), for example, “water” or “hole.”
- **ball position**: tiles can also be designated as transporters, for example, entering one tile could automatically transport the ball to some other position on the course; some type of visual cue is needed to identify transporter tiles, and to denote where the player’s ball will be moved when it enters the tile.

A key requirement for this extension is appropriate rendering to make the different tile types appear realistic. In particular, deep grass, water, ice, sand, and gravel all require a rendering technique more sophisticated than simply colouring the tile differently (e.g., texture mapping, translucency, ripples, specular highlights, and so on).

The following deliverables are required for the Advanced Tile Type extension:

1. Implement at least one new tile of each of the following types: rolling friction, shot direction, hazard, ball position.
2. Update the minigolf course file format to allow tiles to specify their “tile type” (along with any other properties, if necessary, to properly manage that tile type within the game).
3. Update the rendering engine to display different tiles types in a realistic fashion.
4. Update the physics engine so the ball rolls properly over the new tile types.

**Suggestion 4: Course Terrain**

In this extension you will construct a 3D topography in which to place the holes in a minigolf course layout. This is commonly referred to as a *height field* in computer graphics. A flat plane is subdivided into squares or triangles using an underlying regular grid. Each grid position represents a vertex on the terrain. Next, vertices are moved up or down to raise or lower the terrain in that region. The goal is to produce a topography with smooth hills and valleys, of an appropriate size, and with appropriate locations to place the individual minigolf holes.

Before the terrain is built, positions for the minigolf holes should be selected. This allows the minigolf holes to be “packed” onto the finished terrain. It also defines how big the terrain needs to be to fit all of the minigolf holes. If you look at how a real minigolf course (or a regular golf course) is laid out, you will see that the end of hole \( i \) is close to the beginning of hole \( i+1 \). The tee for hole 1 and the cup for hole \( n \) (where \( n \) is the number of holes in the minigolf course) are also co-located close to one another.

Some minigolf hole layouts have a cup or a tee that rise up above the rest of the minigolf hole. In this case, you will need to place external objects (e.g., ladders, ramps, or platforms) to “provide access” to that part of the hole. The goal is to create a seamless, believable minigolf course.

Finally, when players move between minigolf holes, the scene should no longer “jump” by removing the current hole and displaying the next one in a single step. Instead, a camera animation path should be built to move the players from the end of the current hole to the beginning of the next hole in a smooth, movie-like fashion. This will allow players to “see” the terrain and the minigolf holes that lie within it.

The following deliverables are required for the Course Terrain extension:

1. Design and implement a packing scheme to take a set of \( n \) minigolf holes and pack them into an appropriate sequence of positions on an underlying terrain model.
2. Implement a method to construct a random terrain model with a user-defined size.
3. Update the rendering engine to display all of the minigolf holes together with the terrain as each hole is being played.
4. Design a method to animate the camera from the end of one hole to the beginning of the next as each hole is completed.

**Hand-In Requirements**

Use Wolfware (the online assignment submission software) to submit the following files:

- `minigolf.exe`, a Windows 2000 executable of your finished course project,
- `README.txt`, a text file that explains how to perform interaction in your course project, and
- all associated source code files (these can be called anything you want).

There are two important requirements that your course project must satisfy.

First, your executable and instruction files **must** be named exactly as shown above. The program will be run and marked electronically using a script file, so using a different name means the executable will not be found (and subsequently will not be marked).

Second, your program **must** be compiled to run in the Multimedia Lab environment. This means that you should logon to a Multimedia Lab machine, compile your code, and test it to ensure it executes properly. You are free to complete your initial work on another architecture (e.g., a home computer, or a Linux or Solaris workstation); this should be relatively easy to do, since C/C++, OpenGL, and GLUT are all portable across different platforms.

It will not be a valid excuse to say “I forgot/didn’t know that I was supposed to submit that kind of executable file.”