You will each be making a big individual semester project. You will each be developing a role-playing game. It will be a simple dungeon game with several rooms. There will be interactions between entities. The minimum requirements are that you have a 25 room dungeon with a starting and ending point and interactions with at least 4 Non-Player Characters (NPC).

You can have prizes and puzzles. You can have fighting and talking. You can have a final boss. The requirements are given below as a guide to how you approach the tasks you will be given along the way. You should refer to these requirements often as you write code for your game and even as you write code for the other assignments given.

The rubric (or grading criteria) are given on a separate page. The pages following it contain thorough explanations of each criteria.

Major Project 1 (RPG) Requirements:

|  |  |  |
| --- | --- | --- |
|  | Points | Earned |
| Standard comment section appropriately used, including name, pseudocode, notes, maintenance log. Comments are used throughout program as appropriate. (comments) | 4 |  |
| Introduction and clear instructions. Easy to use interface. (I/O) | 4 |  |
| Simple dungeon with a minimum of 25 rooms. (0.16/ room) max 4 pts | 4 |  |
| Minimum of 4 NPCs (1 / NPC) max 4 pts | 4 |  |
| Interactions with NPCs effective and consistent (I/O, branching, looping, classes) | 4 |  |
| Option mechanics are effective. (branching) Game loop and other loops effective. (looping) | 4 |  |
| Use strings to hold appropriate data, including but not limited to addressing the user by name, holding NPC responses, other canned output that might be used multiple times. (strings) | 4 |  |
| Functions used appropriately, including passing correct arguments and returning correct values (at the very least for intro and interactions). To get full points, make sure to have functions for rooms and interactions. (functions) | 8 |  |
| Game works (has a beginning, middle, end) (2). Game is replayable and has entertainment value (1). Game has error checking (1). | 4 |  |
| Game uses structs for your player statistics and other places as appropriate. (struct) | 8 |  |
| Game uses vectors (arrays) as appropriate. At the minimum, you will save your NPCs in a vector. You might also choose to use vectors in other ways. (vector) | 8 |  |
| Game uses object-oriented programming. The NPCs will go from being structs to classes. Interactions with the NPCs will be via methods associated with the classes. (classes) | 8 |  |
| Game will use files for saving the game state. The user will be able to save the game at checkpoints or at will depending on how you design your game. The user will be able to start a new game or continue a saved game when the game starts. (File I/O) | 8 |  |
| Game has been thoroughly tested by at least four others (peers and possibly others) who also fill out a 5 question questionnaire (that you create). | 4 |  |
| Other | ±4 |  |
| Total points (competency grade) | 80 |  |
| Total points (converted / not just normalized) | 100 |  |
|  | Points | Earned |

Many of the words in parentheses are concepts you have learned along the way so far. Branching and looping and I/O are a big part of any game and you see them in several places in the scoring rubric. The notes below are meant to help explain what they mean. (And thus how to ensure you maximize your score.)

Make sure you have the standard comment section at the beginning of your code. If you are missing your name or pseudocode or maintenance log, you will be missing some points. If your maintenance log is not complete or has a simple beginning and ending date with little between or little in the way of explanation of what you have done, you will not receive full points. And if you do not have comments throughout your program, you will not receive full points. These are simple things, so maximize your points.

Your game must have instructions and an introduction. Tell the user how to play the game in those instructions. Talk about movement. Give any back story that you need. Tell who wrote the game. And get the user’s name. You will need it later to address the user. Also, make sure that your user interface is clean and simple to understand. Even text games need to have a nice user experience.

You must have a minimum of 25 rooms (areas) in your game. You may have more, but you will not receive any extra credit for having more. The rooms (areas) will be counted and if there are less than 25, the number of rooms will be multiplied by 0.16 to get the score out of 4.

You are required to have 4 NPCs, who either wander around the rooms or may be in specific places. There are no extra points for more. The number of NPCs will be counted and points given up to 4, one point per NPC. The NPCs must be object-oriented. That is, they need to be 4 objects created with one class. If you decide that your NPCs will be structs instead, you will not receive all of your points. The NPCs need to be stored in a single vector.

Make sure to have interactions with NPCs. This could be fighting or talking or running away (the three usually implemented in RPGs). These interactions should be done in a consistent manner (which probably means you will implement them with methods). And if you implement fighting to the death as a feature or option, make sure that you give some good output as your game loops through the steps toward doom (of the user or the enemy NPC).

Make sure that your game loop and options work properly. You should not exit the game until the user chooses. You should keep fighting or talking or doing whatever in a manner that makes sense to your game. You should make sure that your game is internally consistent. Be true to your story. Be true to your genre. And even though it is technically part of a different section (in terms of being spelled out), make sure to have error checking. Do not let the user choose something and go off into neverland (unless that is supposed to happen). Make sure that every option has some output. Even if it is a chiding remark that the user cannot do something the he or she chose.

You need to use strings. At the very least, you need one string for the user’s name (which will be in the user struct). Address the user by name whenever you talk to him or her. This will mean that you will pass the username or more probably, the user struct to functions. You may also want to hold different responses to the user when different situations arise. You can use functions similar to the fortune cookie problem to give different responses. Know that these responses can be particular to a situation (they do not need to be random).

Make sure to use functions and object methods appropriately. This means that you have the appropriate number of functions for rooms (it may actually be less than one per room, and probably ought to be), movement, interaction, and whatever else you might need functions for. Also, know that you can call functions from other functions besides just calling them from main(). However, do not call a function from itself, especially main(). Not unless you already know recursion and how to properly define your ending conditions. (But still do not call main from itself. Ever.)

Make sure that you save the user’s game state in a file. When the program begins, you should give the user a choice to start a new game or continue with an old game. Let the user pick the file that starts the new game. The game state will contain all the stats of the user and the NPCs. Including the locations of each.

Make sure your game works, with a definite beginning, middle, and ending. You should test it often. Not just when you think you are through. Test it at many junctures of programming. Make sure it is replayable and has entertainment value. This can be shown by having other people test your game. In fact, you will show that you have had at least 4 people test and review your game by creating a 5 question questionnaire and having your testers test and review your finished game. These four printed and filled out questionnaires will be attached to your printed code.

There are also some aspects of coding that do not quite fit into the neat categories in the rubric but that are taken into account. Most of those things are objective and are discussed here, but there is certainly a small degree of subjectivity in judging code. Understand that they can add or subtract points (up to 4 either way).

Do not use global variables. Period.

Make sure to be neat and use whitespace to your best advantage to make your code readable.

Use comments. Not every line needs to be commented, but you should have comments on certain lines and sections. Remember, one reason you are commenting is so that you can remember what you did when you look at your code a year or two from now.

Write code that shows you know what you are doing.

Write elegant code rather than spaghetti code. Spaghetti code is fragmented, sloppy, and shows little organization. Elegant code uses readable identifier and function names, follows a specific naming convention, uses indentation and whitespace, organizes in functions, objects, structs, etc., and has a clear arrangement.

Write code that is yours. Copying other people’s code is not acceptable in the major program (or any program really). If you take a segment from a website or modify somebody else’s code, make sure you document those things. Include a URL. Show how you made it your own.

**Understand that using somebody else’s code is more than just an 4 point issue. You and whoever you copy from will be in danger of failing the project and subsequently the class.**

You only get one shot at the major project. In other words, I only grade it for points one time. Have your peers review your code and test your project. And check your code against the rubric and the explanatory notes below it (these). You ought to even be able to go through the rubric and grade yourself or at least have a good idea what your grade will be.

Final note. If you are interested in making use of more advanced techniques in the major project or any projects, you may refer to the chapter which has some advanced topics on User Interface (UI) and other fun stuff. You do not need to have any of these extra features in your code to receive full points on your project.