Project 4: A game

In this project you will be making your own game. It is a text adventure game, where you walk around in some sort of world, exploring or accomplishing a task. You can pick the game's theme (what sort of world it is, etc.), what sort of activities are most important (fight monsters? trade resources?), and what the goal is.

I have provided a very simple world to start with. As a game it is lackluster to say the least, but the code that runs it should provide a framework from which to build your program. There are several files included in the program, each defining classes that help make up the world. You should start by playing the game and looking through the code to see how it works. (When playing, the command "help" will give you a list of other commands.)

Once you get comfortable with the code, you should begin changing and building on it to make your own game. The list of possible changes later in this document describes many improvements you could make and assigns a number of points to each change. (In general, bigger/harder additions are worth more points.) To get a passing score on this assignment, you must earn at least 20 points, though you should aim to earn at least 40. You can pick which improvements to make, but remember that your goal isn't just to check off boxes by making the improvements, but rather to make something that is actually a good game. (So for example, it is probably necessary to choose the option that expands your world beyond 4 rooms…)

The improvements are not explained in the sort of specific detail that would tell you exactly what changes they include. You have to add this functionality in a way that you think makes for a good game. For example, you have to figure out how to modify the user interface so that the new functionality can be used. (If you are adding doors that require keys to unlock them, do you just make it so you have to be carrying the key to go through the door? Or is there a specific "unlock" command? If so, you should update the "help" description to include information on how to use it.)

You should create a document that lists all the improvements you made, and a brief explanation of what specifically you did for each improvement. (Sometimes there will be very little to say, but if the improvement gave you freedom to choose how things would work, you should say what you chose. If you have to add a new type of monster, for example, say what type you added and what if anything is special about it. You should also have a quick note about what code in what files was added/modified to make this work.) You are also free to make improvements that aren't on the list of options, and if you do that you should describe them here as well. You should also state explicitly the total number of points you have earned.

Your project will be evaluated in several ways. These are:

- How many points did you earn from your improvements?
- How well did you make the improvements you made? Did you do the minimum to make it count, or do something very challenging? Were there bugs? Were the improvements made in a way that was easy for the user to understand? Were they incorporated well into the game?
- How good is the overall game? You should be trying to make a game that is actually fun to play. Pick improvements that help you achieve this and try to design the game well. Make your improvements well-integrated into the game. (For example, if you add weapons to your game, you should make it so that there are various different weapons that exist in the world so that finding a good weapon is part of playing the game. Just having a single weapon on the floor in the starting room that you pick up at the beginning of the game doesn't add much.) Make sure someone who is new to the game can figure out how to play.
- How well documented is your code and how understandable is it? This applies both to comments in the code and to the document you're submitting. Your code should be well-written and readable, and I should be able to figure out what you did and how you did it.

Some helpful notes:

- The starter code has each class in a separate file. It is not necessary to make a file for every class. For example, if you have classes defining particular types of items, it might be natural to put them in the items file. However, it can often be good practice to keep some things in separate files as a way of making the code more manageable.
- You should think ahead and have an idea of what you want your game to end up looking like before you start adding things, but you should also add things one at a time in a way that can be tested as they get added.
- Leave time for general testing of the game at the end. Even once all of the technical improvements have been made, you will want to change things around a bit to make the game better. Feel free to have a friend play the game and see if they enjoy it, if they can figure out how to play, etc.
- The starter code is meant as starter code. It is ok to change things. You should also not assume everything is done perfectly. You ideally want your program to work gracefully even if, for example, a user makes a typo while inputting a command. The starter code is reasonably good about this sort of thing, but it can certainly be improved. However, you should not just throw out the starter code and start over (at least, not without talking to me first). I expect you to make a game of the general type that the starter code prepares you for.

Working with Partners

You are allowed (even encouraged!) to work with a partner on this project. If you do so, you must both contribute roughly equal work to the project. I also expect a slightly more ambitious game from you – you will need 25 points to pass and should aim for at least 50 (as opposed to 20 and 40 for single individuals).

You should also submit a brief description of who did what work. (This should be no more than a page, probably substantially shorter.) Did you sit at a computer together working? Split things up? Test each others' code? You should describe briefly what parts each person worked on, what was worked on together (and what "worked on together" meant for you) and how you coordinated your efforts.

This is a small enough project that coordinating the efforts of two people should not be overly complicated, but if you're interested, I would recommend using a git repository. (These can be set up for free online, and a quick google search will give not only the needed resources but also instructions on how to use git.) Let me know if you need help with this.

Deadlines

This project will include several intermediate deadlines.

Nov. 3 Submit a brief paragraph to Gradescope. This should say who you will partner with (if anyone) and the general game idea. (The game idea is just a very basic description – 2 or 3 sentences should be plenty.) If you are working with a partner, you should both separately submit the same thing.

Nov. 16 Project check-in. By this day you should have 20 points worth of improvements implemented. You should show these improvements to a professor or TA. You can do this in TA evening hours or in professor office hours.

Dec. 1 You should send a working beta version of your game to the classmate(s) who has been assigned as your tester. They will play/test your game and give you feedback.

Dec. 3 You should give you feedback on the game you tested.

Dec. 7 Final deadline for submission of the entire finished game.

Menu of possible improvements

"drop" command (1 point)
Make it so that the player can drop items, so that their inventory doesn't become unreasonably large.

"wait" command (1 point)
In the game as it is, time passes only when you move to a new room. Make it so that you can type a command that just lets time pass. (Harder version: let the user type a number so they can wait for a number of turns with only one command.)

"me" command (2 points)
Create a command that lets the player see their current status, including their health and/or whatever other attributes you've added.

Bigger world (2 points)
The game starts with four rooms. That is not enough for much to happen in. Make the world bigger and more interesting. Give descriptions to rooms that say more than just a room number.

Inventory maximum size (2 points)
Make objects you're carrying have a size or weight, and put a limit on how much you can carry.

"inspect" command (2 points)
Create a command that lets the player look at items. This should display the item description. The player should be able to look at both items they see in their current location and items in their inventory.

Weapons (2 points)
Implement a type of item called a weapon, which the player can use to make themselves more formidable in battle. (This will require modifying how fights against monsters work.)

Armor (2 points)
Implement a type of item called armor that makes it harder for a monster to kill the player.

Auto-generating monsters (2 points)
Make it so that over time new monsters appear in the world.

Victory condition (2 points)
Create a way for the player to win the game.

Healing items (2 points)
Create items that players can use to restore health. These items should only be usable once each and should disappear when used.

Locked chests (2 points)
Create "chests" or other containers that cannot be opened without a key, and also the relevant keys.

Locked doors (2 points)
Create doors that cannot be used unless first unlocked with a key, and also the relevant keys.

Containers (2 points)
Make containers (bags, boxes, etc.) that can hold other objects.

Stacking items (2 points)

Make the inventory display "<name> x2" if there are two identical items (and so on, if there are more), rather than listing the item name twice.

Regeneration (2 points)

Have the player regenerate in some way over time. (The easiest thing is that health regenerates a little each turn, but there are other things you could do as well.)

Random world (3 points)

Make the world be generated in a random way when the program is run so that it's somewhat different each time the game is played.

Loot (3 points)

Make it so that defeating a monster causes useful items to appear. There should be some amount of randomness about what those items are.

Events (3 points)

Make it so that interesting things sometimes happen. These can be based on a timer or somehow random.

More monsters (3 points)

Create several different types of monsters. They should not just have different names, but really be different in important ways.

Special rooms (3 points)

Make rooms that have some effect on the player or monsters when they are entered. (Maybe there's a hospital that heals you when you enter it? That sort of thing.)

Complex rooms (3 points)

Give rooms some additional attributes or behavior. Maybe they get dirty? Or full? Or different types behave differently?

Player attributes (3 points)

By default the only characteristic of the player is their health. Add more attributes ("stats" in most games) that affect what the player can do.

Command abbreviations (1-3 points)

All the typing needed to play a text adventure game can get tiresome. Traditionally there are a lot of abbreviations allowed. On the simple end this could be just some short alternate commands ("inv" instead of "inventory", etc.), or on the more ambitious end, this could do something like accept any string that is the start of only one command. (It can even allow items, monsters, and other things to be abbreviated in some consistent way.)

Helper (4 points)

Create an autonomous character(s) in the game that is on your side and does something helpful. You should have some ability to control/influence how it behaves.

Currency (4 points)

Create a currency for the game. There should be a way for the player to gain currency, and also useful ways for them to spend the currency. (A merchant? a store?)

Leveling up (4 points)

Add a mechanic for "leveling up" your character. That is, as they accomplish things in the game, the player should get stronger in some way.

Crafting (4 points)

Create a system where the player can use some items to construct other more useful items.

Magic (4 points)

Create a system where the player has some "spells" (or "abilities" or whatever other name fits your theme) that they can use (possibly learned during the game). There should be some restriction on how often spells can be cast.

Characters (4 points)

Characters are neutral beings in the game that you can talk to and interact with. (The "talking to" can be a matter of picking one of several options of things to say from a list.) They should interact with you or the world in some way when you talk to them.

Building (4 points)

Let the player modify or add rooms/locations. The changes they make should affect the game in some way.

Choose your own (? points)

You can make whatever changes you want beyond your required points if it helps make the game better. If it's important to you to get some points for those changes, talk to Adam and we'll figure out how many points they are worth.

Bonus options

The two improvements below require very serious effort to implement. They are above and beyond the expectations of this assignment, and they may require looking up things we have not learned in class. Nevertheless, you should certainly feel free to implement them if you're feeling particularly ambitious.

Saving the game

Let a player choose to save their game to come back to later. This should create a file in the folder where the program is stored (or better yet, in a "saves" subfolder of that folder) with a name specified by the user. On starting the program, the user should have the option to continue from the point at which they saved, instead of creating a new game.

Graphical interface

Typing is annoying. Make the program work through a graphical interface instead typed commands. You don't have to create pictures that actually look like the world. You can have a window, with portions dedicated to displaying the description of the current room, what monsters, are there, what items are there, etc. Game actions should be accessed through clicking buttons with the mouse, rather than through typing. (You will need to use a graphical user interface (GUI) package for this. Tkinter is the most common.)