The project is based on x86 architecture, because I've chose to import 32bit version glfw library.

The project has structured by MVVM paradigm. We have models, views and viewmodels. I created two views for the program (console and window). You can initialize whichever you like. View can only talks to viewmodel layer and have no idea what's going on in the model layer. Viewmodel is like a middleware that gets data from model layer and transfer data to view. Model layer just contains data and cannot talk to other layers.

I've used command pattern to execute specific commands like buy stuffs or complete a job. For example, in the console application, if you press 'I' to see inventory, a command will execute to show the inventory, or if you press 'buy' button in the window application, a command will execute to do purchasing operation.

I've used ImGui and opengl to create the window application. Although ImGui has lots of awesome features for UI, I'd tried to keep things as simple as possible.

I'd created an Id for each entity to make it easy to use it with a database. I'd created a storage class to deal with data, because I'd planned to use sqlite to handle data and create a save/load system for that, but my time was over.

I didn't use event handling system because of time-limit, but it's better to use a stable library (like boost) for rising and handling events.

I didn't get the exact meaning of this part of document that says "There is a 50% change that the Tool, used for a Job receives some damage.", so I decided to damage the tool after each usage by decrease its amount divided by 2.

Please let me know if you have any issue with working/running the application.

Improvements:

1. I've used singleton design pattern instead of using global variables
2. I've used command design pattern to handle every command we give to the program
3. I've Used MVVM paradigm for the project
4. It's better to keep header files as clean as possible, so I created related cpp file for headers
5. #pragma once is compiler specific and it's better to write a portable header guard
6. It's better to use constructor uniform initializer instead of assignment operator
7. I've created namespace for everything to handle naming as clean as possible
8. I've tried to use encapsulation to prevent errors and data misuse
9. Add Id for each entity
10. I've tried to take advantage of the standard library

Fixed Issues:

1. Prevent memory leak by taking advantage of smart pointers
2. The initialized jump cable was $50, but it should be 20$ based on document.
3. the "fixCosts" property name in the Malfunction constructor is not right, it should be "reward", so the programmer will consider this as final reward to give to player, not a cost that reduce from player!
4. In the Player::PrintInventory(), iter->first should be tool* and iter->second should be int