Zorkish Adventures: Phase I & II

Overview

Create text-based CLI game, in two phases, with a focus in phase one on game state management, and in phase two on useful OO patterns (specifically the command pattern) and resource (adventure details) loading.

The history of the great "Zork" games is interesting and educational (although please don't try to recreate the Zork interpreter the "Z-machine" for this blatant replicant). See for example
> Zork history at Gamasutra http://www.gamasutra.com/view/feature/1499/the_history_of_zork.php
> An interview with two of the Zork developers http://www.armchairarcade.com/neo/node/1383
> Specifications for the Z-machine http://www.inform-fiction.org/zmachine/standards/z1point0/

Concepts

Game states are a very useful concept. You might think about the term game "scenes" (instead of states), much like a scene in a theatre show, where the stage is setup (initialised) and a show presented (update and display), and finally the stage cleared (cleanup), possibly for the next scene.

For game programming, defining a consistent model to handle all scenes is a sensible game framework approach.

Note: The use of OO techniques (in C++) is expected for this game and its features. Although you can create the same structured effect without OO specific language features, this is a good chance to use OO C++ to its full advantage.

Phase I :
The game world is very simple (hard coded data) and only two simple commands are required. This phase is designed to focus on the successful implementation of the overall game state (stages) ideally using OO techniques, but has no real gameplay.

Phase II:
The overall game states ("stages') remain the same, however the Command Pattern is used to implement a richer set of game commands (and add some actual "gameplay*"), and a world description (data) is loaded from a text file.

Top-level Game State Map

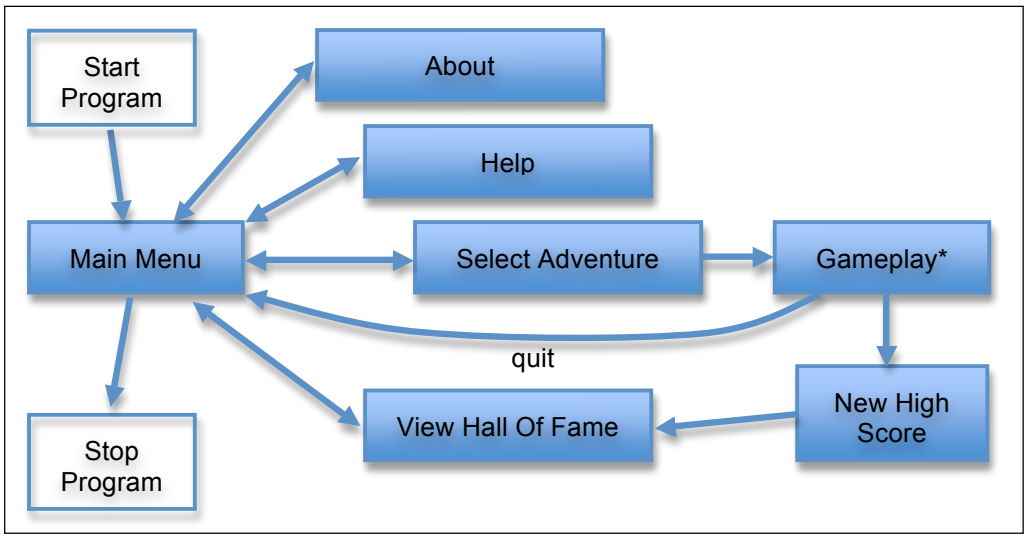

Figure 1. A state map to represent the different game "stages" required in Zorkish.

Game State Details

Main Menu

```
Zorkish :: Main Menu
-------------------------------------------------------

Welcome to Zorkish Adventures

   1. Select Adventure and Play
   2. Hall Of Fame
   3. Help
   4. About
   5. Quit

Select 1-5:> _
```

About

```
Zorkish :: About
-------------------------------------------------------

Written by: [your name here]

Press ESC or Enter to return to the Main Menu
```

Help

```
Zorkish :: Help
-------------------------------------------------------

The following commands are supported:
```

If this is phase 1 development show the following:

```
   quit,
   hiscore (for testing)
```

if this is phase 2 development show the following:

```
   [go] _, (or just n, ne, e, etc)
   look at _,
   look in _,
   inventory,
   open _ [with _],
   close _,
   attack _ with _
   take _ [from _]
   put _ in _
   drop _
   quit
   [up arrow] to repeat last command

Press ESC or Enter to return to the Main Menu
```

Select Adventure

```
Zorkish :: Select Adventure
-------------------------------------------------------

Choose your adventure:

   1. Mountain World
   2. Water World
   3. Box World

Select 1-3:> _
```

Note: the adventures shown by your program should depend on what you decide to create. Initially, you will need to add (hard code) a simple testing world for phase 1. Later, you might want to scan a directory for valid adventure files, or list the valid adventures files in a text file and load the list.

View Hall Of Fame

```
Zorkish :: Hall Of Fame
-------------------------------------------------------

Top 10 Zorkish Adventure Champions

   1.  Fred, Mountain World, 5000
   2.  Mary, Mountain World, 4000
   3.  Joe, Water World, 3000
   4.  Henry, Mountain World, 2000
   5.  Susan, Mountain World, 1000
   6.  Alfred, Water World, 900
   7.  Clark, Mountain World, 800
   8.  Harold, Mountain World, 500
   9.  Julie, Water World, 300
   10. Bill, Box World, -5

Press ESC or Enter to return to the Main Menu
```

New High Score

```
Zorkish :: New High Score
-------------------------------------------------------

Congratulations!

You have made it to the Zorkish Hall Of Fame

Adventure: [the adventure world played]
Score: [the players score]
Moves: [number of moves player made]

Please type your name and press enter:
:> _
```

Gameplay

Depends on phase. For representative samples see the later sections.

Phase 1 Gameplay Sample

Phase 1 does not contain complex gamplay -- it is simply to test the game stage management. So, a simple phase 1 “adventure” experience for a human player might look something like this:

```
Welcome to Zorkish: Void World
This world is simple and pointless. Used it to test Zorkish phase 1 spec.
:> quit
Your adventure has ended without fame or fortune.
```

Which would take you back to the main menu. Alternatively, you can end the phase 1 game with

```
:> hiscore
You have entered the magic word and will now see the “New High Score” screen.
```

Which provides an easy way to test the new high score game “stage” transition without the need for gameplay command complexity.

Phase 2 Gameplay Sample

In phase 2 gameplay commands (using a command parser and a command pattern) can be more complex.

```
Welcome to Zorkish: Stone World
This world is made up of several locations, objects and entities to interact with.
Used it to test the Zorkish phase 2 spec.

:> look
You are in a square room.
There is a single door leading North.
There is a bag on the floor

:> look at bag
The bag is small and tied closed with a piece of string.
You could probably manage to undo the knot if you tried.
There is something in the bag.
```

```
:> open bag
Sorry, you don't have a bag.

:> take bag
You have taken the bag. It has a nice weight to it.

:> look in bag
The bag is closed. You can only guess what is inside.

:> open bag
Your skills are poor, but you do manage to undo the knot of the bag.

:> look in bag
There are 3 gold coins in the bag.
Score: +30

:> go north
Ouch. The door is closed.

:> open door
The door opens easily.

:> n
You walk north. After a short walk you arrive at a junction.
There are rocks scattered on the ground.
There are paths heading East, West and South.

A troll approaches. He looks rather fierce but has no weapons.
The troll yells "Gimme ya stuff or 'ol smash ya!"

:> attack troll
You swing a punch at the troll and HIT. He looks annoyed.
The troll swings a punch at you and MISSES. Lucky you.

:> take rock
You have taken a rock. It's heavy.

:> attack troll with rock
You throw a rock at the troll and HIT. He falls to the ground.
The troll is unconscious.
Score: +200

:> quit
Thus ends another grand Zorkish adventure.
```

Note: All commands and names are treated case-insensitively.

Implementation Recommendations

State Pattern for Game Stage Management:

The management of the different overall games stages can be handled in many ways. A simple type of control loop using magic "mode" values (or enumerated values) and a switch statement would suffice. You could also use function pointers and a generic process loop.

However one of the best, and most agreed upon approaches, is to use the state pattern. Essentially, each of the different game stages ("Main Menu", "Hall of Fame" and so on) is considered a game "State" which is defined as a class and managed by a StateManager object which either knows about all the states it has to manage, or the states are "registered" with it (perhaps by an overall game or application object).

The term "Stage" and "StageManager" are suggested as they seem to be a bit clearer than just "State" which is easily confused with the idea of managing the in-game "states" of units and characters. However, as always, use names that ***you*** *think best describe form and function and don't be afraid to change as you go.*

The state pattern also makes it easy to manage states in stacks or queues, and is commonly used for "undo" control in programs, however this feature is unlikely to be used in this game specification example.

One of the most relevant online articles is http://www.codeproject.com/KB/architecture/statepattern3.aspx which specifically relates the state pattern to managing game stages as discussed here.

Game World Representation:

The game world can be represented in many ways. It is, essentially, a graph (in the sense of vertices and edges, not a "chart" with squiggly lines). Locations can be nodes (vertices) in the graph and directional edges connect each location. (This is quite different from a simple grid world model.)

NOTE: The world is NOT at GRID! Support any direction you want – including "up" "down" if you want!

A world should also contain entities that have identifiable names, descriptions and may contain other entities. Some entities are dynamic characters that can change the environment or change their location. Other entities can be altered ("open bag") or used to alter other objects ("attack troll with rock"). However, as a whole the environment still needs to be unified into a system that we can be loaded from file and then its state "altered" as we process user commands and update the environment.

You will need to select and implement an appropriate game world model. Careful planning will pay off here – plan twice, discuss with someone else **BEFORE** you code anything! Review before you code. Seriously.

Think also about how you will represent a world (or "adventure") in a simple text file. Such a file will need to contain both locations in the world and entities, along with names, descriptions and properties.

Consider the following in your design:

- Think of all the possible properties (name, description) and uses ("open _", "put _ in _") your entities might need to support. Do you need different code to support each, or a single coded "entity" (class) that can use data to define its behaviour?
- Entities can contain other entities. Consider the "composite pattern".
- Given that locations can also contain "things" are they just another entity?
- Each location needs to be linked with another location in some way. "Bi-directional" links might be easier than a model with separate individual links. For example, a path between the forest and the house is one "link" (N and S) or two separate links (one for "N" travel and one for "S" travel).

A tree can work as a good structure for the entire game world, but make sure you understand how this is true before you try it. Each node in the tree (which is technically a graph still) can contain other nodes (or be an empty leaf node), can know who its parent is, and who its siblings are. You will also need to be able to modify the tree (get an entity, put it somewhere and so on). Your own research required!

Command Processing:

Again, there are several ways you might represent and implement command processing. In its most basic steps, you will need to break up the text entered by the player into pieces. Command "words" are typically the first piece, and based on that may need to process single or multiple additional values. Consider the following

```
[go] _, (or just n, ne, e, etc)
look at _,
look in _,
open _ [with _],
attack _ with _
take _ [from _]
```

It makes sense then to identify the "first" command or action, and then pass all the processing of any remaining pieces to a specific method (or object) that knows what to do and how to do it. (This is known as a classic "verb-noun" grammar.) You should also be able to support command "aliases," so that "put" is the same as "drop", take/get, go/move, look/examine etc. without radical changes to your code. Adding new commands should not be too hard.

Short version: the **command pattern** is the way to go. Use a command processor to take a string (or similar) select from the command objects it manages, and then "runs" the appropriate command and gives it the string (as it might need it). A command would also need to know about the world (or the current world "context" at least) because a command may need to know what game entities are "visible" to the player.

A command "dictionary" would be useful, as you can "lookup" the command object instance using a string "key" such as "put" or "attack". A dictionary also makes supporting aliases quite simple (two keys point to the same command object instance). Try to avoid creating multiple instances of the same command class.

You might, for example, have a base Command class, and then something like GetCommand, PutCommand, AttackCommand etc all inheriting as "a kind of" Command and each able to process.

Of course wikipedia is one place to start http://en.wikipedia.org/wiki/Command_pattern and David Cumps also has one of his nice articles on design patterns applied to games (in C# though) http://blog.cumps.be/design-patterns-command-pattern/