CS 162:

Introduction to Computer Science II

Final Project Report

By Elliot Bates

Contents

Discovery

- I am to design and make a text based game.
- The game should take the format of a room/compartment based structure with 4 directions leading from each room to another room (These don't have to lead anywhere in every room, but the links must exist).
- The player must have some goal to achieve that requires them to collect something from the rooms.
- There must be at least 3 different types of room, each with something different happening in them.
- There must be some sort of time constraint on the player.
- The rest is up to me!

Design

Initial Design - Story

The player will play as a hero you must try to save the world by defeating an evil monster. The world map will be divided into 4 regions: forrest (South), field (central), mountains (East), and Swamp (West). The player will start in the forrest. They must then head to the castle and meet the princess. She will tell them to go to the mountain and seek help to find the amulet. In the mountains the player must find a character (who will give them a ladder) so that they can reach the top of the mountain. At the top of the mountain the player must defeat a mountain spirit to obtain the amulet. They must then return to the castle and speak to the princess. She will tell them to go to the swamp where the player must find a demon. The demon will set a hangman-style game. If the player wins the game they will get the spell that they need. If they lose they must fight the demon to get the spell. Finally the player must return to the forrest where they started to fight the final boss.

Overarching Program Design

There will be an abstract Location base class which the world location types will be derived from. The base class will contain a pure virtual function called 'resolve'. The main body of the program will function as follows:

```
Do {
        Resolve events at current location
        Player chooses direction to move in
        Move to new location
} while game end parameters are not met
```

When the player chooses which direction to travel in they will also have the option to view their hero, use items, look at the map, view the help page, and change game settings. Each time this loop iterates the ‘day’ in the game will increase by 1. If the player loses all their hit points or runs out of days before the end of the game they will lose. Brief descriptions of some of the more important classes are listed below. A class hierarchy diagram containing all classes can be found in appendix A.

Locations

The game will contain 4 different types of location: random encounters, shops, healing fountain, and quest related locations.

Random encounters

At these types of location the hero will have a random chance to run into a creature that they can either fight or run away from. If the player does not initially run into a fight they can choose to look around. Looking around gives a random chance to either run into a fight, find an item, or nothing.

Shops

The shops will be populated with random lists of items that the hero can purchase.

Healing fountain

The healing fountain will restore the players hit points. There will be a random chance that a stranger gives the player some hint or tip whilst they are resting in the fountain.

Quest location

These are where all the events that affect the story will take place. Event at these locations will be unique to each location.

Creatures

The creature class will be used for all characters in the game, including monsters, bosses, and the hero.

Hero

The hero class will be derived from the creature class. In addition to the functions needed for combat, variable and function will be added to facilitate movement and interaction with the game (such as an item inventory).

Development and Testing Plan

Development Plan

Stage 1 – Write creature, hero, item, location, and all location sub class files. Implementation of all non-get/set member functions can be left blank for now. Create locations in main function and basic game movement loop. Test stage 1.

Stage 2 – Modify fight class from previous assignments. Add fights, story and other necessary functions to quest locations. Add fights to encounter locations. Test stage 2.

Stage 3 – Write item sub-class files. Add items to quest locations. Add functionality to shop locations. Add items to encounter locations. Add hero inventory interaction functionality. Test stage 3.

Stage 4 – Add remaining functionality to main program (main menu, game settings, instructions, graphics, map display). Block off sections of map until certain requirements are met. Add time constraint. Test stage 4.

Test Stage 1

Testing	**Variations to be Tested**	**Outcome**
Open program → use navigation menu to travel between locations → exit program	Make sure every link (direction) from every location is tested	Selecting directions that should connect to other locations cause the player to move to that location. Selecting directions that have no location to lead to causes a message to be displayed and the player is able to choose again. This is as planned.

Test Stage 2

Testing	**Variations to be Tested**	**Outcome**
Open program → use navigation menu to travel between locations → travel to quest locations in order → exit program	Repeat the mountain top battle choosing each of the options to check the correct opponent is being made. Try to pick the same item that you use on the mountain spirit to take after the battle. Test what happens if you fail to guess the password or if you do guess it.	The story at each location displays correctly and in order. In the mountain top battle all possible opponents are created correctly. Choosing the item you already used on the mountain spirit causes a message to be displayed and another can be chosen. Choosing other items is allowed. Guessing the password correctly causes the location to be resolved without a fight. Failing to guess it forces you into the fight.
Open program → use	Continue travelling around until all	The encounter locations function as

navigation menu to travel between locations → travel to random encounter locations → exit program	encounter locations have been visited and all possible creatures have been encountered. Test the fighting mechanic.	designed. The current fight option makes each combatant attack once before the fight menu is displayed again. This can be arduous to interact with in a long fight you expect to win, so an additional option has been added to allow players to make the fight run through to its conclusion.

Test Stage 3

Testing	**Variations to be Tested**	**Outcome**
Open program → use navigation menu to travel between locations → visit shops → exit program	Buy all the items, buy none of the items, fill up your inventory, empty your inventory, use every item.	The shops, items, and interaction with the heroes inventory work as designed. Arriving at a shop with a full inventory means that you cannot buy any of the items so an option has been added to access your inventory in the shop and use up items if your inventory is full.
Open program → use navigation menu to travel between locations → travel to random encounter locations → exit program	If a fight doesn't happen look around in locations. Continue until all possible items have been found.	Items are found and added to inventory as expected. The same problem as with the shop (full inventory means you can't get the item) is found so a similar option is added to the encounter battle.
Open program → use navigation menu to travel between locations → travel to quest locations in order → exit program	Check all quest items are added to/removed from inventory at the right time. Try to get rid of quest items.	All quest items are added/removed at the correct time. They cannot be used or removed from the inventory, as designed.

Test Stage 4

Testing	**Variations to be Tested**	**Outcome**
Open program → change game settings → play game → exit program	Test different game speeds. Try with/without god mode.	A suitable range of game speeds is allowed. Enabling god made adds extra gold and stats as designed.
Open program → play game → view instructions → view map → go to other locations → view instructions → view map → exit program	Test at many locations. Test instructions from inside main menu.	The map and instructions display correctly whenever they are called.

Open program → play game → try to travel to location in the wrong order → exit game	Try every possible route into areas of the map that should be restricted at that time.	There is no way to get into areas of the map until you are supposed to go there.
Open program → play game → run out of days	-	Game over message is displayed

Test Stage 5

The final stage of testing involved playing the game myself and having others play it and observing them. This was very useful in balancing out elements of the game such as attack damage, hit points, and gold. This stage of the testing also revealed to me that perhaps the game should be a little more guided. I added a 'current objective' string to the hero class which could be seen in one of the main navigation menu options. This will help the player to remember what they are supposed to be doing next.

Implementation

Source code for this program has been submitted to TEACH along with this report.

Reflections

What problems did I face?

With so many files I faced an error towards the end of the project that I eventually worked out was being caused by circular #including. This was the first time I had come across this problem, but with a little reading a was able to learn when to #include and when a declaration (e.g. 'class Item;') is sufficient. There were also a few times where I found that my design didn't allow me to do something I wanted to. This was caused by the data I wanted access to not being available in the current scope. This was solved by modifying the classes a little (e.g. by adding the hero pointer to the Location class).

How did the program evolve?

I made lots of small tweaks to the program (for example by adding the functions mentioned in the testing sections). But the overall running of the program did not really evolve. I think this stemmed from having a (fairly) well thought out design. Once I have completed and tested the program I found that the items side of the game was quite bare and so in order to make the game more interesting I added two more item sub-classes – Sword and Shield. Swords and shields were added to the shops and encounter areas and the equipping/discarding of unwanted swords and shield was extensively tested.

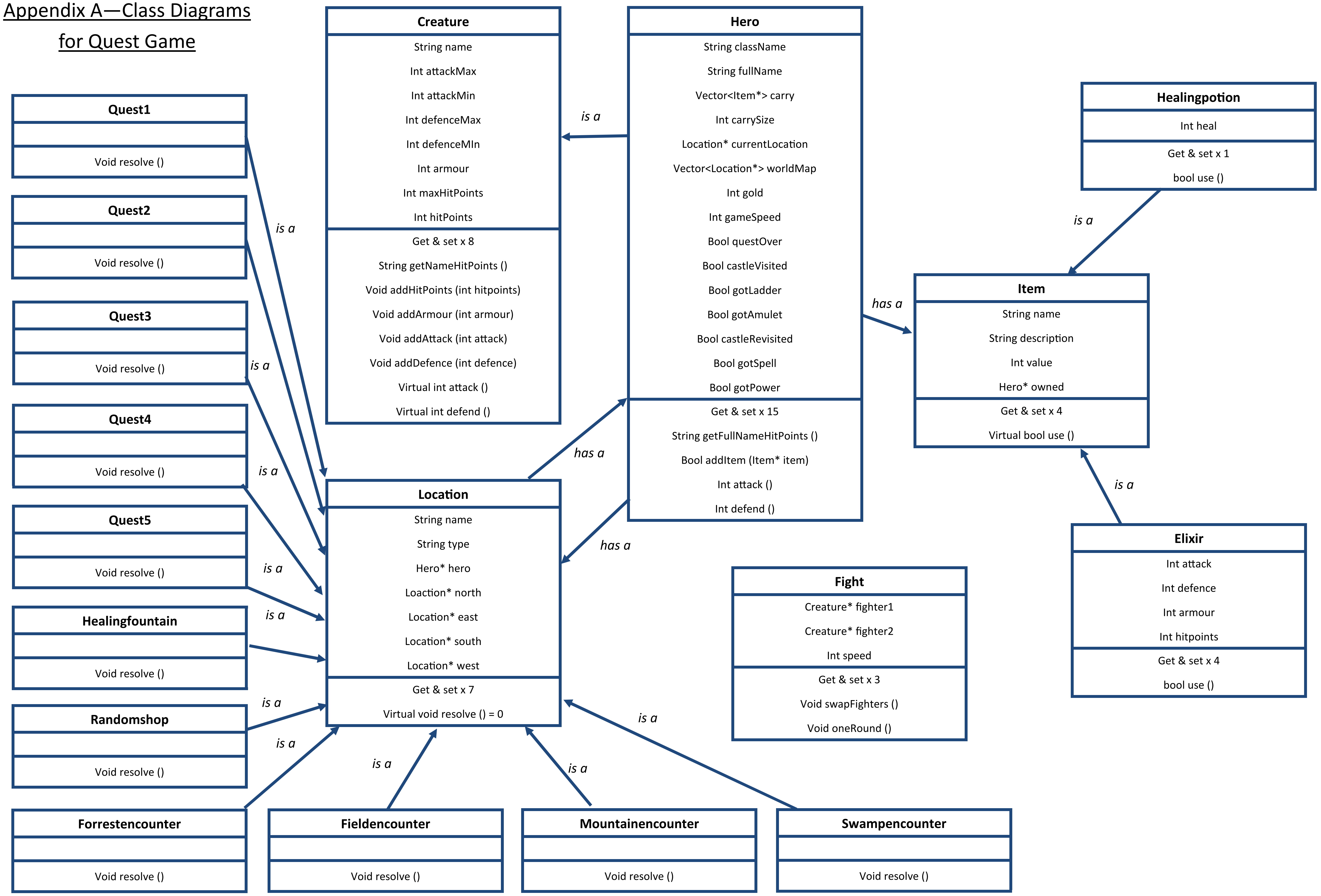
What did I learn?

I feel that the project really reinforced what I know about using pointers and using inheritance and polymorphism. For example I found that the following declaration is valid (provided Sword is a sub-class of Item):

```
Item* thisItem = new Sword (arguments)
```

Polymorphism can be a very useful tool in this way and was a vital element in the design of this program. This was by far the largest single programming undertaking I have done and really tested my planning and time management skills. I found myself constantly making ‘to-do’ lists so that I didn’t forget one of the steps needed to implement what I was working on, or to make sure that I didn’t forget a little tweak that I realised the program needed.

Appendix A—Class Diagrams for Quest Game


Creature
String name
Int attackMax
Int attackMin
Int defenceMax
Int defenceMIn
Int armour
Int maxHitPoints
Int hitPoints
Get & set x 8
String getNameHitPoints ()
Void addHitPoints (int hitpoints)
Void addArmour (int armour)
Void addAttack (int attack)
Void addDefence (int defence)
Virtual int attack ()
Virtual int defend ()
Hero
String className
String fullName
Vector<Item*> carry
Int carrySize
Location* currentLocation
Vector<Location*> worldMap
Int gold
Int gameSpeed
Bool questOver
Bool castleVisited
Bool gotLadder
Bool gotAmulet
Bool castleRevisited
Bool gotSpell
Bool gotPower
Get & set x 15
String getFullNameHitPoints ()
Bool addItem (Item* item)
Int attack ()
Int defend ()
is a
Healingpotion
Int heal
Get & set x 1
bool use ()
is a
has a
Item
String name
String description
Int value
Hero* owned
Get & set x 4
Virtual bool use ()
is a
Elixir
Int attack
Int defence
Int armour
Int hitpoints
Get & set x 4
bool use ()
Quest1
Void resolve ()
Quest2
Void resolve ()
Quest3
Void resolve ()
Quest4
Void resolve ()
Quest5
Void resolve ()
Healingfountain
Void resolve ()
Randomshop
Void resolve ()
is a
is a
is a
is a
is a
is a
has a
has a
Location
String name
String type
Hero* hero
Loaction* north
Location* east
Location* south
Location* west
Get & set x 7
Virtual void resolve () = 0
Fight
Creature* fighter1
Creature* fighter2
Int speed
Get & set x 3
Void swapFighters ()
Void oneRound ()
is a
is a
is a
is a
is a
Forrestencounter
Void resolve ()
Fieldencounter
Void resolve ()
Mountainencounter
Void resolve ()
Swampencounter
Void resolve ()