The Legend of Zorldo Group M

Andrew Mele, Robert Davis, Richard Muri, Daniel Garcia

Project Requirements:

A set of rooms which contain:

* At least one exit:
  + We had rooms which can be entered and exited by standing next to the doors and saying ”open door”
* Items and rooms with descriptions:
  + All items and rooms can be inspected by saying “examine <item>” or “examine room”
* The rooms and items should not be fixed:
  + Our map had a set destination (the boss room), but the user may explore the dungeon in any order and navigate the maze to reach the end.
  + Each room (except for special cases) has a random number and assortment of enemies and pots, each with random items contained.
  + In addition, the user may return to any previous state of the game using the save and load commands.

An adventurer who can:

* Pick up and store items
  + Lank gains items such as rupees and hearts by killing enemies and smashing pots, which are added to his pocket.
  + Lank gains more important items such as new weapons through opening chests
  + Items are always displayed in the pocket
* Get rid of items in some way
  + Certain items are expendable, for example, the arrows are decreased by one each time the bow is used. Once all arrows are used, the arrows are completely removed from the pocket and the bow is unusable.
* Use items where appropriate
  + Various items are used to fight enemies by equipping them such as swords and bows, or indirectly such as the boomerang to stun the enemy.
  + Items such as hearts and fairies can heal Lank.
  + An error will be displayed when the user does not use the items as required.
* Interact with the environment
  + As explained previously, the use can examine any item to gain more knowledge about it.
  + Additionally the user interface is a much more intimate interaction between the environment and the user than any text would be.

You must be able to configure an initial state of the game:

* We implemented a save and load function that allows the user to return to any previous save state.
* The user may also manually edit the XML file to change this configuration.

Extra details

* A large user interface and a maze set-up to the map make the experience more immersive for the user.
* The further into the game the user goes, the greater weapons he gains, therefore increasing his fighting capabilities.
* The map is large enough that it warranted a more responsive movement system than typing “move north” so we implemented the use of arrow keys for movement within the room. To move between rooms still requires additional commands.

Main Program:

Main: initializes game then drops into a while loop where it continues to define a command based on user input then executes the command and updates the room and characters in the room. After each loop game checks if end-game pre-requisites are met, and if they are drop into those lines of code.

Items: creates all the items in the game and returns them as a dynamic list.

Chests: creates all the chests in the game and returns them as a dynamic list.

Commands: creates all the commands in the game and returns them as a dynamic list.

Characters: creates all the characters in the game and returns them as a dynamic list.

Ganon: creates the boss Ganondalf, and returns him as a dynamic list.

Pots: creates all the pots in the game and returns them as a dynamic list.

UCommands: creates a restricted list of commands that only the user can access.

GetString: gets a specific string from the user.

GetYN: returns either true or false depending on user input.

GetCommand: using a regular expression to parse user input, then returns the desired command from the command list.

Items Class:

Item: constructor for item type.  
  
Use: A switch case statement that takes the Item’s Name and executes a specific set of instructions depending on that name.

Chest Class:  
  
 Chest: constructor for chest type.

Command Class:

Command: constructor for command type.

Movement: moves Lank based on user input.

Use: redirects to Item.Use sending user input as temporary strings.

Examine: prints an item or room description to the screen utilizing the Interface class.

Help: displays the help page where the user can look up the game controls.

Open: opens either a chest or door depending on Lank’s location relative to the chest or door.

Fight: redirects to Lank.Fight and Character.Fight sending user input as temporary strings.

SaveGame: redirects to Save.SaveGame.

LoadGame: redirects to Save.LoadGame.

Map: redirects to Room.Map.

Execute: executes the chosen command based on the commands name, (taken from GetCommand in the main program).

Temp class:

This class was primarily used to pass temporary variables into different classes, without having to instantiate a whole new variable. Because these variables were static, we could use them in nearly every method or class.

Save class:

SaveGame: creates a save file called savegame.xml.

Loadgame: loads the saved file into the game.

Character class:

Character: constructor for character type

Die: once the character drops below 0 health, the character is removed

Equip: an item is equipped to the character

Fight: the character attempts to do damage to another character

Stalk: the enemies will follow Lank and not collide with each other.

Lank Class:

Lank: constructor for Lank

ChangePocket: adds of removes items from Lank’s pocket.

The static class Interface does not inherit from anything but object. The reason for this is that the class serves to print the correct things to the console and not to fully keep track of the game state. Most of the variables in Interface are used to store the state of the interface so that when the refresh function is called, the game will be the same.

The methods are:

* CleanOutput - Cleans the output queue by writing spaces. Dynamically writes spaces the exact number of characters to reduce write time.
* Clear Pocket - Wipes the entire square of the pocket with spaces
* DrawCharacter - uses oldx, oldy, and character to wipe the old location with a space and draws them at their current location.
* DrawChest - Wipes the old location of chests and writes their old location
* DrawGameWindow - Draws the window the size of windowHeight and windowWidth as well as the number of hearts the character has and the rupees
* DrawInput - Draws the input at the bottom of the window
* DrawLank - Draws specifically the character lank from the main class at his new location
* DrawPocket  - Draws the pocket using the I variable or the pocket variable from Lank
* print - takes a string, then dynamically dequeues or enqueues based on the size of the buffer from the windowHeight variable.
* Refresh - clears the console, then calls a few other functions along with locally stored variables to recreate the interface state
* RefreshHealth - clears the space above the game window and then reprints the main characters health and rupees
* StartDraw - Calls other functions to initialize the game at the start.

Room Class

The room class is used to draw the rooms, and fill them with things including pots, chests, and enemies. It has ten fields; an integer x location and integer y location used for drawing the map of the rooms, a list of characters, list of pots, and list of chests to fill the room with, and five bools, one to check if Lank has been in the room before and the remaining four a description of north south east and west door locations.

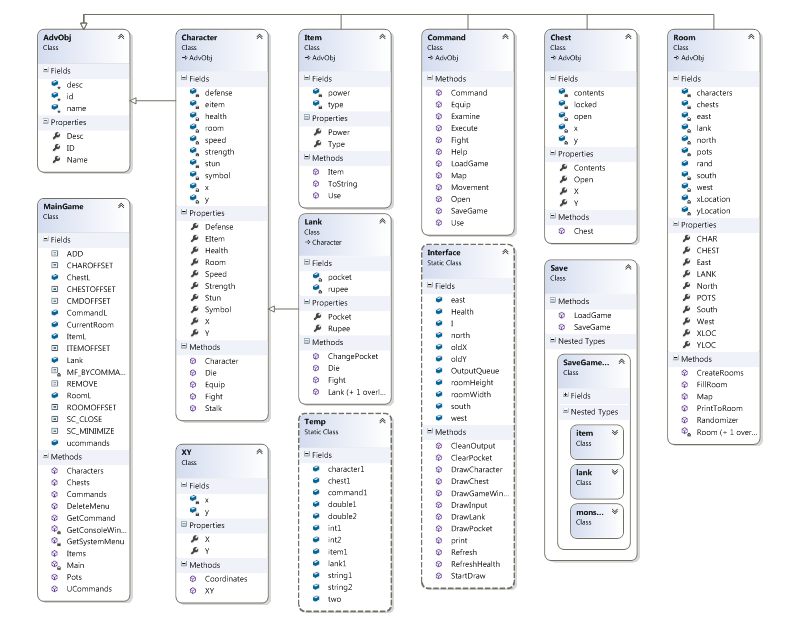
The methods are:

* CreateRooms(): This function defines every room using the brute force method, and stores each of them in a list that gets returned.
* Map(Lank LANK): This functions prints a map to the console whenever the user types map. It operates by printing various color squares at the x and y locations of a room defined by the x and y fields.
* FillRoom(int lowMonster, int highMonster, int lowPots, int highPots): This function checks if Lank has been in the room before; if he has, it immediately calls the PrintToRoom function. If so, then it calls the Randomizer function to fill the room with a number of pots and monsters between low and high, at random x and y coordinates within the room, then it calls PrintToRoom.
* PrintToRoom(string type): checks for an object to print, and then prints that object at the x y coordinates contained by the object x and y fields in the room.
* Randomizer(string monstPot, List<Character> Pots): checks for monster or pots as a string argument, and returns a list of random monsters or pots.

XY Class

The XY class is used exclusively to generate a list of all the xy coordinates within a room, and then choose a random element from that list and assign it to a character or pot for printing. The class has one function called Coordinates which returns that list.

We have tested every aspect of our game. There still remain a few bugs, however nothing is game breaking. We primarily tested the game by actually playing it and observing if everything was working as intended. We urge you to test out all of our commands and try using several different items to test the games ability.

UML Diagram:

Flow chart:

Division of work:

Room Design: Richard Muri

Combat/enemies: Robert Davis

User Interface: Daniel Garcia

Commands/Items: Andrew Mele