Textadventure Moria Mines – Milestone 3

Note that this milestone is stretched over 2 weeks!

Contents

Learning goals

- Practice object references even more
- Practice inheritance
- Practice interfaces
- Improve your software architecture even more
- Trying out unit testing

Specifications overview:

Read the whole assignment before you begin – you are supposed to begin with UML.

Monsters

You must add enemies to the dungeon. Monsters are basically like a player, but are controlled by the system and not by the user.

But why?

Like items, practicing object references is helpful to improve your OOP skills. Also, this increases the size of your project, which should demonstrate your need for good software architecture.

Combat / interaction

The player must be able to fight the monsters. If you have chosen a non-combat game, there must be another kind of interaction between player and NPC instead, like chatting or trading.

But why?

This forces you to bring together many parts of the code, for example checking player health vs. monster health, checking if the player deals extra damage because of an equipped sword item, etc. Keeping your code clean and readable will be an interesting challenge.

The monsters drop loot

When a monster / NPC dies or is defeated, they must (at least sometimes) drop loot.

But why?

Making sure objects are instantiated and ready to be assigned to the players' inventory is yet another exercise in object references.

Sort the high score file

The high scores must be sorted so the highest scoring player is displayed first.

But why?

This is an algorithmic challenge to sort the high score file.

Make an end room / improve end room

Make sure there is an end state – a place where the player can win the game (and record his score to the high score file). There could be an end chest that contains all kinds of treasure items, or an end boss that the player must beat. Maybe also make a little epilogue about our hero's fate?

But why?

This is just for making the game more fun.

Design principles

Keep these in mind – they still apply

1) Make sure you still have a controller class
   https://en.wikipedia.org/wiki/GRASP_(object-oriented_design)
2) Follow at least **S** and **O** of the 5 SOLID principles.
   https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)
3) Incapsulation of attributes
   https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)

But why?

Because practicing these guidelines will enhance your object-oriented design skills and make it easier to avoid making spaghetti code! Your code will become more maintainable and reusable.

UML

Before you add monsters and handle combat, you must make a class diagram and sequence diagram of how you plan to implement the next features.

Then do the code.

When your code is finished, you must update both diagrams, but keep a copy of the pre-code versions. Hand in all four diagrams

One more time for emphasis: first the diagrams then the code and then the diagrams again.

But why?

Because this will help you get a design perspective before you get too focused on the details, and in the long run your code will be better.

Test

Make a JUnit test (or more than one) to test the central part of the combat system. This test should be complete – That is both negative and positive tests and all equivalence classes.

But why?

Because testing improves the quality of software immensely.

Game overview – the 3 milestones combined:

Your game should at the end of this sprint consist of a combination of all 3 milestones:

- A player
- That walks through a dungeon made of interconnected rooms
- He can pick up gold and items
- And meet monsters along his way
- He can fight the monsters
- And maybe get some phat loot (item pickup)
- He has an inventory for items
- And he can win the game by reaching an end room

Optional extras / suitable bonus challenges

- Load / save the game (first consider only making it possible to save/load 1 single game, not a list of games).
- Make the dungeon be randomly generated at every new game start, instead of using a "hardcoded" dungeon of rooms.
- Make your own interface and implementation somewhere in the code
- Make Javadoc documentation of your code

Hand In

Hand in on Fronter no later than Thursday 10th November 2016 at 23.59.

- Please hand in only a single .zip file (you can drag 'n drop documents and pictures into your Netbeans project, and then export the project as a zip)
- Please hand in as a group.

But why?

This will make it easier and faster for your teachers to give you feedback

What to hand in:

- Code in zip format
- UML diagrams
- All test cases

On Friday 11th November you will do a peer review of the work done by another group. Your work will of course also be reviewed by another group. You will receive further information before that day.

Appendix A: Recommendations and extra info

The info below is not strict requirements, but some info, for example the combat system part, could help you get started on how to code the combat.

Player

A player must have a name, hit points, damage and a level. All players must start in level 1 and with 30 hit points (see Player start-up for content and weapon).

If the player's hit points are reduced to 0 or less, for any reason, he/she is dead and the game ends. This is a single player game.

The player has a list of items and can carry an unlimited number of things. This list must be possible to see, either always visible or on request only. Items that can be used up, such as healing potions must be removed from the list when used. Otherwise, items cannot be dropped or removed from the list.

Level:

Whenever a player defeats a monster he gains a level. When a player gains a level he also increases his maximum allowed hit points by 10. If the player is at 100% hit points he should immediately have his current hit points set to his new maximum. If he is wounded there should be no change in his current hit points only his maximum allowed hit points should be adjusted.

Starting

Starting the game:

Before the game starts, the dungeon must be created and populated with monsters and items. The 'End Treasure Chest' must be placed in a room (preferably in a room at the end of the dungeon; not the first room). Secondly a new player must be created, complete with an item list (see Player startup for the item list). The game must start in the starting room.

Starting room:

There must be a starting room (displayed as a hallway/corridor on the map). This room must, in addition to the normal description, also contain an extra description with the adventure background and history: Why does the player enter this particular dungeon? Why is it so interesting and/or rewarding? Leaving the starting room by moving out of the dungeon must end the application.

Player start-up

Make a starting content for the player: backpack, 1 healing potion (returns hit points to max), 1 short sword (6 damage/strike).

All players must start in level 1 and with 30 hit points.

Ending

The game ends in two ways:

1) There may be several treasure chests in the dungeon; however one chest must mark the end of the game. Here it is referred to as the “End Treasure Chest”. When a player picks this treasure chest up, he has won the game and it ends.
2) If the player's hit points are reduced to 0 or less, for any reason, he/she is dead and the game ends. The player must then be given the option of ending the application or starting over.
Treat the restart option as starting a brand new game with a new player character.

In both ending conditions, the game must display all of the treasures collected and the total amount of gold coins. The game must also display the players ending level, hit points and maximum damage.

Action

The games must start the player in the starting room. For each room a description must be displayed. A player must be able to select an action in every room, for instance moving ‘north’ or ‘west’. The possible actions need not be immediately visible. It is part of the gaming experience that you must guess what actions can or should be taken in a room.

The actions must be text based, and use standard input. The game executes the action, for instance by moving the player north to the next room and showing him the description of this room. If a player does not write the correct command or writes a command that cannot be executed, the game must respond appropriately and keep the game running.

At all times it should be possible to write ‘help’ and then be shown a list of **all** commands that the game accepts in its current state, and their exact spelling, even though only some of them is possible to execute in the players current room. This list should not show what commands are possible and impossible in the room you are in.

The following are commands that the player can select:

-**"help"**: shows all the commands that a player can type.

-**"attack"**: uses the player's best weapon + level to attack a monster.

-**"pickup"**: Picks up all items in the room. If there is a monster in the room it must be defeated before any items can be picked up.

-**“load”**: loads the saved game. There is only one save game.

-**“save"**: saves the game. There is only one save game. See the save/load description below.

-**“new”**: creates a new game and player and starts from the beginning.

-**“quit”**: ends the game. Display an appropriate end message.

-**“use”+<item_name>**: attempts to use an item that you have picked up. Ex. “use healingPotion” will use a healing potion that the player has picked up.

-**"west", "east", "south" & "north"**: moves the player to the room in the given direction. If it is not possible to move the player, the game must respond with an appropriate error message and let the player type a new command. You cannot move past a monster, only back the way you came; you must slay the monster to move past it.

Items

All items must, as a minimum, have a name and a description. When entering a room with an item the player must see it; the item must be added to the description of the room.

When an item is picked up, using the "pickup" action, it must show up in the player's item list. The monster in the room must be dead before any items can be picked up.

Weapons must have a damage number as well. The weapon that gives the most damage is always the one a player uses.

Potions must be useable via the "use" action. Treasures must have a gold coin number that describes their value. Items can *not* be dropped (but some might be used up – e.g. lotions).

When creating the dungeon, insert the following items in the rooms of your choice (the End Treasure Chest should be hard to reach):

- 1 End Treasure Chest
- 1 Bastard sword, 14 damages
- 1 Long sword, 10 damages
- 4 x Healing Potions. Returns a player's hit points to maximum
- 1 x Weapon poison potion, permanently increase weapon damage by 5 when **"use"**.
- 3 x Rubies, 200 gold coins each
- 2 x Peals, 50 gold coins each.
- 1 x Silver necklace 800 gold coins.

(Ambitious students may insert more items to make their dungeon more balanced or enjoyable to play.)

Monster

- A monster must have a name, hit points, damage and description (= looks, smells and sounds when first encountered).
- Only one monster per room.
- A Monster cannot move away from the room they occupy.
- They never chase the player however they always attack the player as soon as he enters the room.
- You cannot move past a monster, only back the way you came; you must slay the monster to move past it.
- There must be at least 6 monsters in the dungeon (ambitious students may insert more monsters to make their dungeon more balanced or enjoyable to play).

Combat system

The combat system is a simple, turn-based, auto-hit system (= you always hit the opponent):

Monsters and players always hit each other. Only damage is deducted. Monsters and players take turns attacking each other, with the monster starting. When entering a room, a player is immediately attacked by the monster and damage is deducted from the player's hit points. It is now the players turn. In his turn he may use 1 action and then it is the monsters turn. The player can for instance use the **"attack"** command to attack the monster. This command always takes the weapon with the highest damage number plus the

player's level and deducts the sum from the monsters hit points. Then it is the monsters turn to attack the player and after that the players turn and so on until one of them is dead. If the player's or monster's hit points are reduced to 0 or less he is dead.

However the player may also choose another action in his turn, for example he could choose to use a potion to bring his health to maximum or he could choose to leave the room (he will not be attacked when leaving and the monster will not follow the player).

- If the player is dead the game ends. If the monster(s) is dead the player gains a level and gets to loot the room: only when all monsters are defeated can the **"pickup"** command be used and all items be picked up by the player.

- The player must be shown how much damage he/she takes and gives to the monster.

- When a player defeats a monster he/she gains 1 level. Increase hit points by 10, see Level under Player.

Play test

Test your dungeon and make sure it is possible to complete it.