

Dungeon Game Reflection

by Elliot William Short

FILE	25363_ELLIOT_WILLIAM_SHORT_DUNGEON_GAME_REFLECTION_312866_718243357.PDF (1.03M)		
TIME SUBMITTED	11-DEC-2020 04:26PM (UTC+0000)	WORD COUNT	2827
SUBMISSION ID	139286940	CHARACTER COUNT	12783

University of Wales Trinity Saint David

Software Dev and Testing/Agile SEP/Database Systems

ACCA7000/ ACCA7001/ ACCA7002

Elliot Short - 1702797

Contents

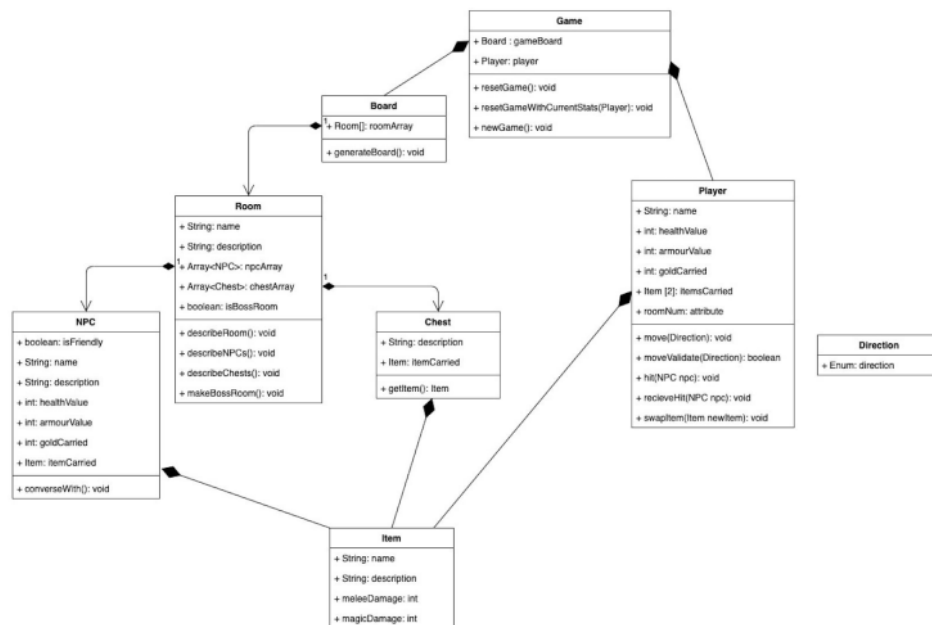
Introduction	3
Design Evaluation	3-5
User Stories	5-8
My Contribution	8-11
Agile	11-12
Conclusion	12

Introduction

As a team we were tasked with creating a text based dungeon adventure game that allowed the user to move between rooms in the dungeon, talk and fight non playable characters (NPCs), pick up items that can be used to fight and pick up spells that can be casted. The aim of the game is to collect as much gold as possible before escaping the dungeon. The game was to be developed in java with a database connected to it so that the program could have persisting data. These were the requirements and the rest was up to us as a team for how we wanted to develop this application. We followed the agile methodology throughout the entire project, from the sprint planning at the start to the sprint retrospective at the end.

Design Evaluation

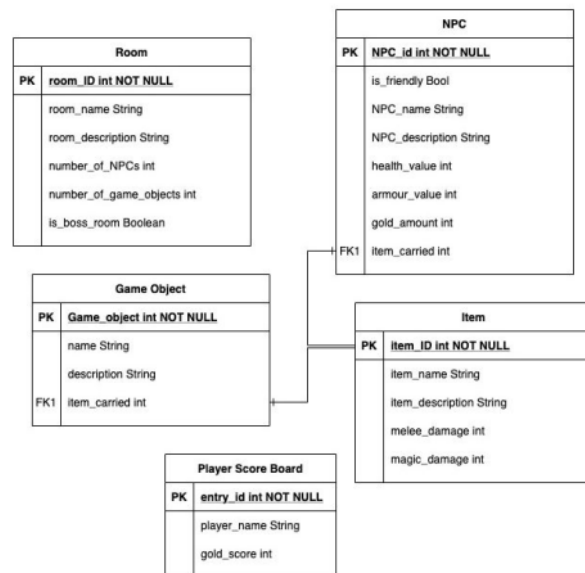
At the start of the project during sprint planning we discussed extensively how the gameplay would work, all while taking notes. Once we had decided how the fundamental gameplay would work we discussed what attributes each aspect of the game would have. For example, we decided the attributes that the player would have were: name, healthValue, armourValue, goldCarried, itemsCarried and roomNum. From the discussion most of the attributes for different classes were decided, with some being expanded on further when Finchley created this UML



UML diagram of Dungeon Adventure Game

diagram to visualise the design. We followed this diagram during development of the game when creating classes and objects. Under the attributes listed in the diagram are the functions that each class would need. For example, NPCs had the converseWith function that allowed the player to converse with a NPC, asking them questions like “What is in this room?”. Now that development has finished we can see that our final product followed this diagram almost exactly as it was designed. This initial diagram was extremely helpful, reminding the team what had been decided so that we weren't changing how the final product would look day by day. When developing a program blind you run the risk of ending up with a product vastly different from that which you had originally envisioned. While this may not always be a bad thing, it is better to have a rigid idea of what the program will look like, and then make modifications to that as a team rather than on the fly making adjustments to your original plan. We followed a similar process when initially deciding what our database would look like. In the end we used object-relational mapping for our database solution, this allowed us to not use as many queries when calling on the database, this made the overall program more efficient and quicker to use.

During the initial sprint planning we decided on what attributes would need to be in the database. An example of a discussion that took place during this planning was the item database. The criteria stated that the player should be able to use an item to attack a NPC and they should be able to cast a spell. Instead of creating two separate tables for physical and magical items, we created a single item table that included melee and spell damage. We decided that the difference between a magical and physical item would be their melee_damage and their magic_damage. So a sword would have a melee damage of 7 and a magic damage of 0. A fire blast scroll would have a melee damage of 0 and a magic damage of 5. This gives the player the illusion that they are using two different kinds of items when in reality, they are the same item but with different damage attributes. This saved us from having to make extra



ERM Diagram of the Database

tables. This diagram allowed us to have a good plan of what the database would look like before development, similarly to the UML diagram.

User Stories

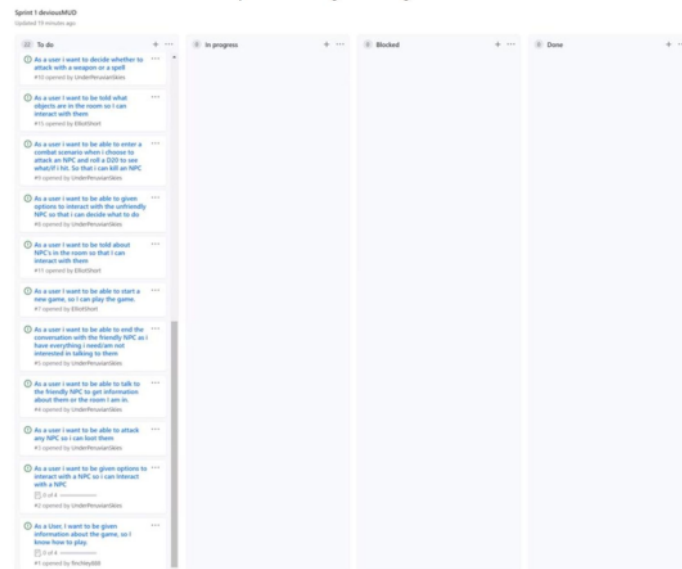
Day 1 of sprint planning, we created user stories to populate the product backlog. Before writing the user stories we firstly wrote the user activities, these included activities like starting a new game and winning the game. Once we had these user activities we filled them with user stories.

Once we had decided on what user stories were going to populate the product backlog, we converted them to items on a Kanban board in github. Not all the user stories we wrote made it into the Kanban board because that board acted as the sprint backlog. We only put user stories into github that we felt could be completed in our 3 day sprint. For each of the user stories we wrote subtasks included in that story and a time estimate. During this sprint planning I acted as scrum master to ensure that agile and scrum principles were adhered to.

One of the user stories that was worked on was "As a user I want to be able to talk to the friendly NPC to get information about them or the room I am in." This story was worked on by me and was completed on the 3 day of the sprint. The story required me to create a function in the NPC class called `converseWithAndAttack`, this function was an if statement that took in the users input and either



Sprint Planning: Creating User Stories



Kanban Board with User Stories

printed dialogue or called other functions. For example, if the user selected dialogue option 2 then the NPC would use the describeRoom function to tell the user about what is in the room. This function requires the room class and integer dialogueOption to be inputted before it can work. The room class allows the function to describe the correct room, the one the user is in, before it describes the room to the user and the dialogueOption indicates which option the user has decided to pick in the NPC dialogue. In the game class a function called talkToNPC prints the options to the user, takes in the users input and then uses a function called isIntInRange to check that whatever the user inputted falls in the range of the options. This fulfilled the criteria of the user story and subtasks which were to "Write NPC dialogue

Option: "What's your name?" return string name

Option: "Tell me about the room" returns extra info about room"

The last part of the criteria is the test. The function isIntInRange is tested by making sure that the user input gives the expected result when inputted, and by making sure the user input is lower than the amount of options given. For example, if the user inputs option 3 in the dialogue, the test makes sure that when 3 is inputted that is the answer given through the function, and then it checks that 3 is not higher than the max option, which is 4. This user story is then finished and moved to the done section of the Kanban board.

```

public boolean converseWithAndAttack(Room room, int dialogueOption) {
    if (dialogueOption == 1) {
        System.out.println("I am a " + this.name);
    } else if (dialogueOption == 2) {
        room.describeRoom();
    } else if (dialogueOption == 3) {
        System.out.println("The " + this.name + " says 'Goodbye'");
    } else if (dialogueOption == 4) {
        System.out.println("You have decided to attack the " + this.name + ", prepare yourself");
        return true;
    }
    return false;
}

public boolean talkToNPC(NPC npc) {
    while (true) {
        System.out.println("What would you like to say to the character?");
        System.out.println("[1] What are you?");
        System.out.println("[2] What's in this room?");
        System.out.println("[3] I have nothing more to say");
        System.out.println("[4] Attack");
        int selectAnOption;
        String answer = stdin.nextLine();
        int integerAnswer = isIntInRange(answer, numberOfItemsInInventory, 4);
        if (integerAnswer > 0) {
            return npc.converseWithAndAttack(gameBoard.getRoomAtIndex(k player.getCurrentRoom()-1), integerAnswer);
        } else if (answer.equals("q")) {
            return false;
        } else {
            System.out.println("Please enter a valid option or type q to quit");
        }
    }
}

@Test
void testIsIntInRange() throws Exception{
    Game game = new Game();
    assertEquals( expected: 1, game.isIntInRange( answer: "1", numberOfItemsInInventory: 2));
    assertEquals( expected: 1, game.isIntInRange( answer: "1", numberOfItemsInInventory: 1));
    assertEquals( expected: -1, game.isIntInRange( answer: "3", numberOfItemsInInventory: 2));
    assertEquals( expected: 2, game.isIntInRange( answer: "2", numberOfItemsInInventory: 2));
    assertEquals( expected: -1, game.isIntInRange( answer: "0", numberOfItemsInInventory: 2));
    assertEquals( expected: -1, game.isIntInRange( answer: "test", numberOfItemsInInventory: 2));
    assertEquals( expected: 4, game.isIntInRange( answer: "4", numberOfItemsInInventory: 5));
    assertEquals( expected: 23, game.isIntInRange( answer: "23", numberOfItemsInInventory: 23));
    assertEquals( expected: -1, game.isIntInRange( answer: "-5", numberOfItemsInInventory: -5));
    assertEquals( expected: -1, game.isIntInRange( answer: "@", numberOfItemsInInventory: 2));
    assertEquals( expected: -1, game.isIntInRange( answer: "5", numberOfItemsInInventory: 2));
    assertEquals( expected: -1, game.isIntInRange( answer: "-abcdefg", numberOfItemsInInventory: 2));
}

```

User Story : As a user i want to be able to talk to the friendly NPC to get information about them or the room I am in.

An easier user story that was the first one completed, "Generate UML diagrams". This user story was assigned to Finchley and the subtasks were "Create UML diagrams" and "Commit to Github and wiki". During the sprint planning we took notes about how we felt the game would progress and then Finchley used these to draw the UML diagrams of the classes we would need for the game. The subtasks were then ticked off and it was moved to the completed section of the Kanban board.

The next user story is an example of a user story that got blocked mid-way through being completed. At the start of the sprint I felt that we had a good idea of what commands we would be giving the user, things like "move north" or "examine room", so I started writing the help page. The problem here was that after I started writing it I realised there were some commands that we had not yet decided if we were going to use. An example of this was the "inventory" command. Originally we felt that we would not need this command as the idea was that if the user picked up an item that was better than what they were currently using it would automatically equip it. But once the help screen had been written we realised that this might change, so we had to move the user story to the blocked section of the Kanban board, and it remained there until we felt we had a final decision on all the commands the user would do. These 3 user stories show how the average user story went, whether it was short or blocked they would follow a path through the Kanban board, getting moved to "in progress" when they got started, "blocked" when another user story was blocking them and finally "complete" when they were finally tested and implemented.

My Contribution

I felt that while my programming skills may not be as strong as other members of the team, that I put all of current knowledge to test and was able to significantly contribute to the project. Out of the 21 user stories I feel that I have contributed in some way to an estimated 11 of them and have heavily contributed to roughly 7 of them. The user stories that I either completed individually or heavily contributed to are the following:

- As a user I want to be able to end the conversation with the friendly NPC as I have everything I need/am not interested in talking to them #5
- As a user I want to be able to talk to the friendly NPC to get information about them or the room I am in. #4
- Create Database Entries #27

- As a user I want to be given options to interact with a NPC so I can Interact with a NPC #2
- As a user I want to be told about NPCs and objects in the room so that I can interact with them #11
- As a User, I want to be given information about the game, so I know how to play. #1
- As a user I want to be able to attack any NPC so I can loot them #3

Here are some extra features that I helped develop that aren't covered by these user stories:

- Adding the gold that a dead NPC had to the players inventory and then displaying that gold when they type the "inventory" command
- Database styling, when the descriptions of items, chests, NPCs and rooms are printed that the format that they have is correct with what we expect.
- Describe functions that read the descriptions of items, chests, NPCs and rooms and print them to the player, whether it's when the player types "describe (item,chest,NPC,room)" or whether the player asks the NPC the same question.
- When the player uses the check inventory, attack, or talk command.
- Inputting all of the data into the database tables such as names, descriptions, item attack values, enemy health etc.
- Writing the original names and descriptions of half of the items, chests, NPCs and rooms.
- Writing tests for the high score screen to make sure that correct score is inputted into the high scores database table when the player wins, loses or quits the game.

When it came to the database I've already mentioned aspects that I was tasked with. After the initial database was set up using NoSQL I was tasked with filling it with data that we already had such as names and descriptions, but also making the values for things undecided such as attack values and health points. Along with this I had to make adjustments to fields data types when different aspects of classes needed to change. For example, early on we discovered a problem when the descriptions we had written for each item was over the limit of varchar, 255. So I had to find a solution to this issue which was to change the data type in the database to longtext. Whenever an issue like this arose I would make those adjustments. When working on the high score functionality I was tasked with testing these in JPAUtilTest. The way I did this was by inputting a test name and test values into the database. A test score would be inputted alongside the name, the test would then check that this score had been inputted correctly into the database by looping through an array looking for the inputted value.

```

@Test
void testScores() throws Exception{
    // enter a new score
    JPAUtil.setScore( playerName: "FinchleyH", scoreValue: 10);

    // get the score array
    ArrayList<Score> scores = JPAUtil.getScore();
    int indexOfAddedScore = -1;
    for (int i = 0; i < scores.size(); i++) {
        if(scores.get(i).getPlayerName().equals("FinchleyH")){
            indexOfAddedScore = i;
            break;
        }
    }
    assertEquals( expected: 10, scores.get(indexOfAddedScore).getScore());

    // enter a new score
    JPAUtil.setScore( playerName: "Elliot", scoreValue: 100);

    // get the score array
    scores = JPAUtil.getScore();
    indexOfAddedScore = -1;
    for (int i = 0; i < scores.size(); i++) {
        if(scores.get(i).getPlayerName().equals("Elliot")){
            indexOfAddedScore = i;
            break;
        }
    }
    assertEquals( expected: 100, scores.get(indexOfAddedScore).getScore());

    // enter a new score
    JPAUtil.setScore( playerName: "Harley", scoreValue: -20);

    // get the score array
    scores = JPAUtil.getScore();
    indexOfAddedScore = -1;
    for (int i = 0; i < scores.size(); i++) {
        if(scores.get(i).getPlayerName().equals("Harley")){
            indexOfAddedScore = i;
            break;
        }
    }
    assertEquals( expected: -20, scores.get(indexOfAddedScore).getScore());

    // enter a new score
    JPAUtil.setScore( playerName: "Adam", scoreValue: 10000000);

    // get the score array
    scores = JPAUtil.getScore();
    indexOfAddedScore = -1;
    for (int i = 0; i < scores.size(); i++) {
        if(scores.get(i).getPlayerName().equals("Adam")){
            indexOfAddedScore = i;
            break;
        }
    }
    assertEquals( expected: 10000000, scores.get(indexOfAddedScore).getScore());
}

```

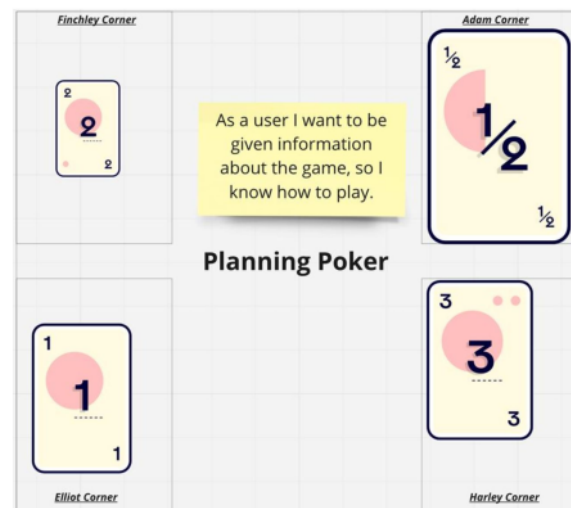
Testing of the Highscores

There were 62 story points assigned over the 21 user stories, after we finished the project the group discussed how we felt about everybody's contribution. While there were aspects of the project that a few of us were less well versed in where our contribution may not have been as great as others, we all agreed that everybody put in equal amounts of work and effort despite this fact. My completed story point estimation is around 16-17 for myself and it would be around the same for all the other members as well.

Pair and mob programming was used a lot in this project as I and another member did struggle to understand a few concepts initially, in the first day or two of the sprint. However, once these obstacles were overcome the team really worked well together, staying on call together for the majority of the day in order to provide help and support to each other. The last 2 days of the sprint were the most productive and where most of the groups work got completed and where I completed the majority of my work.

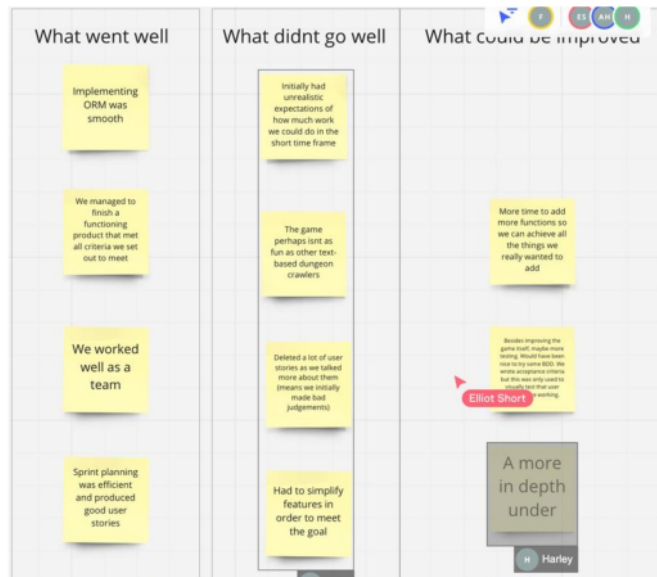
Agile

Throughout the project we used agile and it's process to ensure that we worked efficiently through this short project. The first day was spent entirely on sprint planning, starting with outlining the project. We had been given a criteria by the product owner, and now we as a sprint team discussed what we wanted the final project to look like. I was assigned the role of scrum master so I would organise and conduct most meetings, whether that was the sprint planning on the first day or the daily stand up. As mentioned before, we populated the sprint backlog in the first day on the Kanban board in github. I would split up each story into subtasks, then as a team we would use planning poker to assign each task a time to complete, 1,2,3,5,8 or 13. Before the sprint started I made sure that we had a sprint goal which was " Create a simple text-based RPG, connect this to a database for storing maps and recording player scores." Each day of the sprint I would conduct a daily stand up where I would get each team member to talk about what they did the day



Planning Poker

before and what task they were going to work on that day. I was also tasked with making sure that people kept the Kanban board up to date in github so that members could keep up with the progress of the project. On the final day of development once we completed the project and felt that the sprint goal had been fulfilled we closed the project on github and conducted a sprint retrospective. Here each scrum team member would write what they thought went well, what they thought didn't go well and what could be improved. Normally a sprint review would happen before this step where I would present the work we had done to the product owner. From here the product owner would decide whether the product that we have provided is ready for release, or whether another sprint is needed in order to finish development. As this wasn't possible in the short amount of time we had we went straight to the sprint retrospective.



Sprint Retrospective

Conclusion

Overall I'm happy with how I handled the role and responsibility of scrum master, I think I kept the team following the agile and scrum process as close to intended as possible which allowed our project to run efficiently with no user story, sprint backlog or any other agile related issues. The daily stand up allowed us to keep up to date with each other's progress and keep in constant communication on what parts of the project needed work next. On my overall contribution as mentioned before, the first few days I had to be caught up to date with certain technologies we were using such as object oriented programming and ORM, but in the end once I understood these concepts I was able to work on the team with minimal issues and felt that my contribution to the project in the end was valuable and substantial. I feel this for all the other team members too. Nobody worked substantially less than anybody else and everybody put maximum effort in.

Dungeon Game Reflection

GRADEMARK REPORT

FINAL GRADE

80/100

GENERAL COMMENTS

Instructor

Software design evaluation (30): 26

Database evaluation and implementation (30): 20

Coding and Testing (30): 27

Reflection (10): 7

Ensure that your figures are numbered and have a description - and that you refer to each figure by its number in the body of the text.

Good evidence of teamwork, clearly you worked together well and formed good practices.

A very impressive implementation overall - clearly a significant amount of effort went into this and it was evidently a labour of love.

Ensure when creating user stories that the user is given a persona, e.g. call them "player" instead of "user".

The team has clearly put significant effort into learning about ORM and this has produced a nice real-world type solution. While we were expecting traditional database queries, this is an appropriate (and better) alternative.

Some duplication of code which could have been separated into functions, e.g. the try/catch blocks in

game.java.

Some of the game logic which is currently hard coded could have benefited from being in the database, this might have allowed extensibility, e.g. allowing different levels.

You have evidently made some effort to ensure the code is testable and, therefore, the tests are detailed and appropriate.

Very well done on an excellent implementation!

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

This user story is missing its justification and thus does not have a return on investment. It should have a "... so that..."

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12
