

Assignment2_RhysBeckett

by Rhys Vaughan Beckett

FILE	25296_RHYS_VAUGHAN_BECKETT_ASSIGNMENT2_RHYSBECKETT_312866_1417010543.PDF (1,002.08K)		
TIME SUBMITTED	11-DEC-2020 04:18PM (UTC+0000)	WORD COUNT	2438
SUBMISSION ID	139285441	CHARACTER COUNT	12914

Assignment 2: Block Test – Goblin want Gold!

Introduction

In this reflective report I shall be discussing & evaluating the performance of both myself and the team over this development cycle, this includes discussing the software design, evaluating the database design, considering the allocation of tasks and judging our utilisation of the Agile principles. Once all aspects of the sprint have been discussed, this report will reflect upon this past week, considered both positives and negatives before offering any potential improvements.

Software Design

Immediately after the project brief, we began a Teams meeting to discuss our chosen scenario. After a brief conversation, we unanimously agreed to develop the text-based adventure game. Once a decision had been made, work began on the project.

Though we wanted to follow the sprint format for the coming week, there were some issues with the given situation. Normally a sprint begins with the planning session, where individuals will allocate themselves tasks from the product backlog and add them to the new sprint backlog.

Unfortunately, we had no product backlog yet. Therefore, we accepted that Day 1 would be a low-code effort since we needed to design the product from scratch. Given our previous experience, we decided to utilise Miro for this collaborative design session.

Once everyone was in Miro, the session began by dragging in the scenario brief. We chose to consider this as the epic provided by the product owner. We discussed the epic and broke it down to smaller features, a relatively easy task given the bullet-point format of the brief. Some example features include: Room, Item, Dialogue & Combat.

After we populated the features, we moved on to the individual user stories. Some of these were relatively easy to fabricate given that they were so obvious, an example of this would be: room descriptions, inventory display & movement.

Other user stories required some thinking, especially those tied to the overall system and database. Since these features were too important to relegate into sub-tasks and were difficult to frame from the players perspective; we eventually decided to add an entire feature called 'System' where we could frame user stories from the perspective of the 'System Designer'. The system designer is a role within the DVLA that is utilised to overcome the lack of product owners in the public sector. They represent customers throughout the

DVLA systems, considering their needs and attempting to provide effective solutions. Since this was the method utilised within our recent DVLA project, we chose to carry this process forward.

As we finished fabricating all the initial user stories, we began estimation. For this process we chose to utilise the planning poker method. Though Miro offer an interactive feature for this purpose, we found it to be unwieldy and inefficient. Instead, since we were already in a shared meeting, we chose to utilise the chat system. Overall, our estimation process was quick and painless given that there were very few conflicting estimations. When there was a conflict in estimation, they were usually due to miscommunication and easily resolved.

During the process of fabricating and estimating these user stories, we began to realise that a 'Feature Driven Development' process would be ideal for our requirements. With this new understanding, we chose to separate the product backlog into distinct releases. None of these were necessarily the MVP/MLP since they focused on offering one distinct feature, rather than a partial implementation of the whole. This would allow for targeted testing and feedback of each feature, ensuring each aspect of the game is well implemented; hopefully eliminating the chances of producing the 'weak link' in our mechanics.

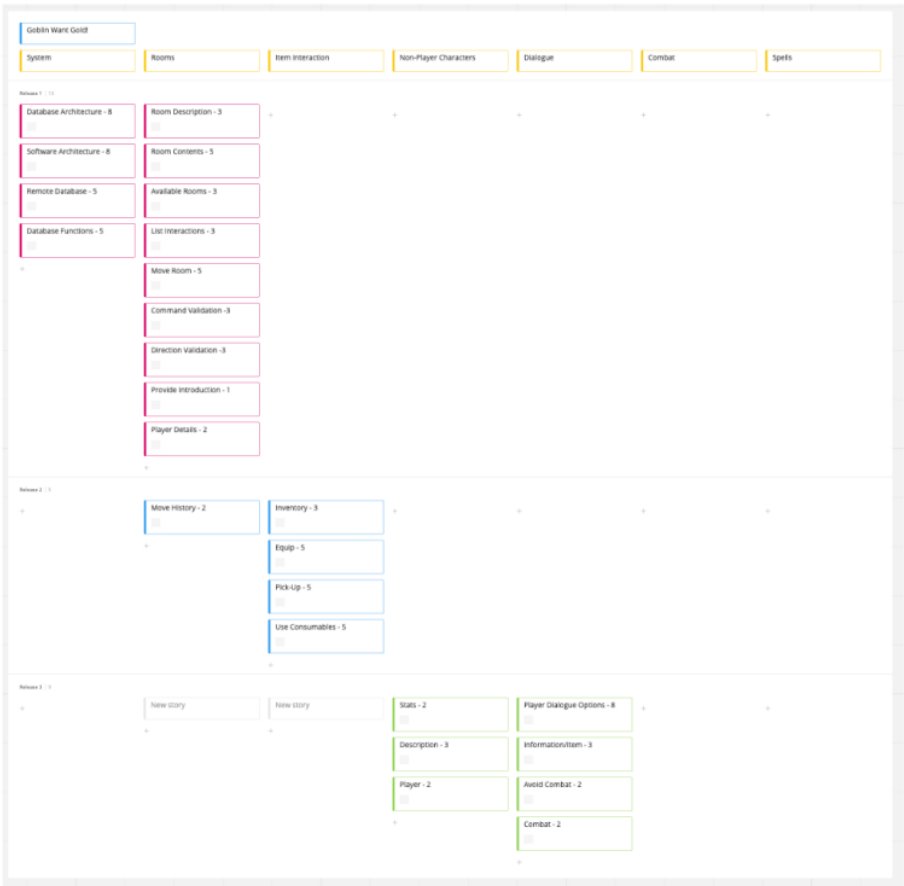


Figure 1 - Product Backlog

Once the product backlog was populated, we could finally hold a sprint planning session. This session was relatively short since we focused on the first release that covered the required systems and movement. Some of these stories needed to complete before meaningful progress could begin, these include: Database Architecture, Software Architecture and Remote Database. As such, we decided to assign everyone to those tasks and complete them in a mob session.

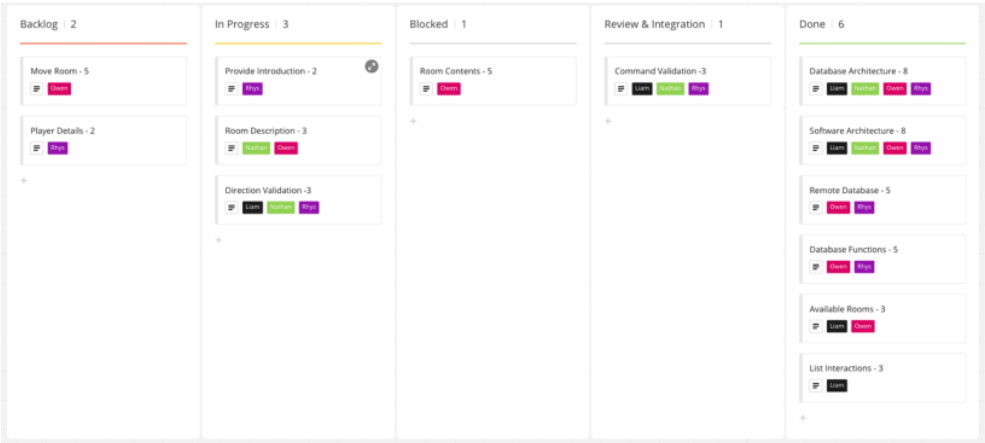


Figure 2 - Sprint Backlog

The database design was completed first and will be discussed later. After the database was populated, we designed the software architecture. Initially we considered planning the entire architecture required for the finished product, however we quickly that would inefficient and go against agile principles. Instead, we focused on the first release; planning the back-end systems and movement controls. This entire process was incredibly collaborative, we discussed the solution in-depth, shared opinions and drew visualisations of the functionality in Miro.

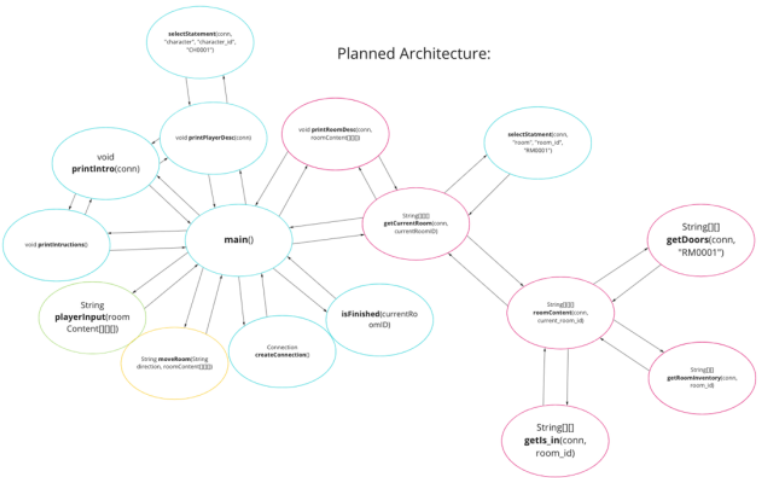


Figure 3 - Planned Software Architecture

Overall, I believe we followed good procedure and produced a strong design. It was certainly beneficial during the development process. As we continued through the development, we did realise that a few of the function on the diagram needed further granularity to keep to the 'Single-Responsibility' principle. However, I don't believe that invalidates our initial design; a good design should allow for changes as the solution becomes clearer. After the development cycle, we finished with a system that is more closely illustrated here:

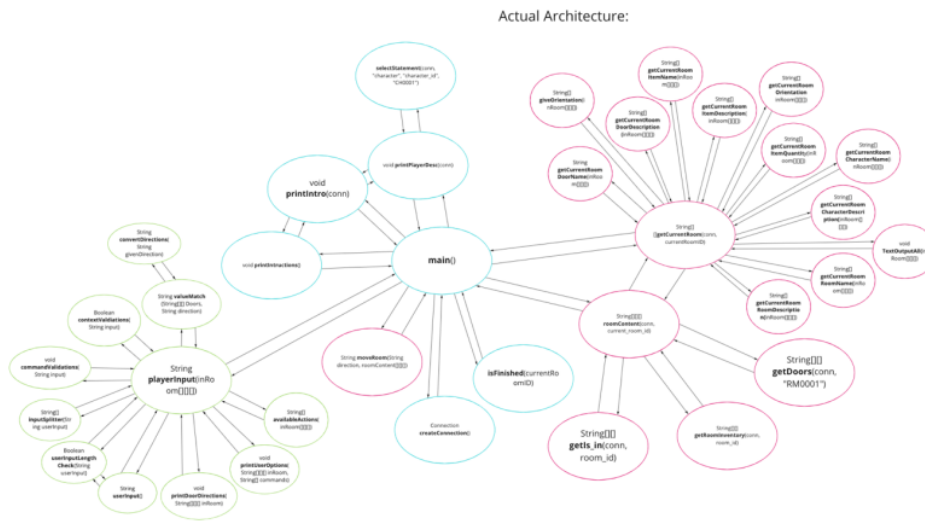


Figure 4 - Actual Software Architecture

Database Design

As previously mentioned, designing the database was the first task we completed. Given the nature of the assignment, everyone needed to contribute to the database; meaning we again held a very collaborative mob session. Similar to the software architecture, we discussed the solution, shared opinions and built a data model in Miro.

Figure 5 - Database Design Model

As we began to build the entity relationship model, we kept the normalisation principles in mind. Firstly, we considered which were the main entities; we chose: character, item, room, spell & speech_line. We chose these as the main entities since they seemed to exist in isolation of one another yet were all necessary for the finished game.

Once we had these entities, we considered some relationships. A few relationships were fairly simple being 'One-to-One', 'One-to-Many' or 'Many-to-One'; an example would be the 'item -> item_effect' relationship. Since many items might have one effect it was identified as a 'many-to-one', which is relatively easy to implement with a foreign key.

The relationships between main entities proved more difficult. Most, if not all of these relationships ended up being 'Many-to-Many', these relationships are harder to spot and trickier to implement. To solve this issue, we utilised joining tables. These table are commonly used to bridge a 'Many-to-Many' relationship in the normalisation process, our joining tables include: room_inventory, character_inventory, spell_tomes, character_spells, worn_by etc.

By following this principle, not only did we achieve better normalisation; we realised that these joining tables could eventually be utilised by a script to randomise the gameplay. This would be similar to the methods utilised by multiple AAA video games, including the Diablo franchise.

In a normal development environment, I would normally expect a single individual to produce a database of this size whilst the other developers would concentrate their efforts elsewhere. However, in this situation we were required to divide this work equally. Though it was awkward, I believe we succeeded in this effort.

Given the nature of constraints and keys, I initiated the implementation process by producing a few of the main entities. Once these were created, we were free focus on an individual section/branch of the model each. I personally implemented the tables that interacted with the room entity, specifically: room, door, room_inventory & door_description. Since we had such a detailed diagram to follow, the implementation of the database was relatively trivial.

Our database was created with PostgreSQL and hosted on Heroku. We chose to attempt implementing a remote database, since this would limit the number of conflicts and versioning errors. Fortunately, the research and implementation of this database process was completed relatively quickly. Though we all dedicated ourselves to the problem, I was lucky enough to find and implement a functional JDBC solution first.

As I was testing my proposed solution, I created a function that would make a SELECT call to the database on the passed parameters. Since the test was successful, I chose to further develop this function as it would've been useful as a layer of abstraction from the SQL. Rather than litter our code with multiple repeated SELECT statements, I thought it would be more efficient to call this function.

Given the nature of SELECT statements, I chose to utilise method overloading for this function. Meaning the developer can call the same function even if they want to filter a SELECT call on a String, Integer, Boolean or no filter at all.

Honestly, these functions were not as immediately useful as I expected. Many of the SQL calls we needed were not simple SELECT statement, most including some form of JOIN. However, I am reassured in knowing that they provided a good reference.

To test our SQL functions, I populated most of the tables with some form of test data. Since this test data is intended to be immutable, it should minimise failures in the unit testing process. Meaning the unit test shouldn't fail due to an altered value in the database.

Again, due to the nature of the assignment, we collaborated on the unit testing of these functions. Though I mostly wrote the functions myself, I thoroughly explained the process and divided the unit testing process. I personally wrote the tests for the SELECT All and String functions, which are available at the end of this report.

Task Allocation

As previously mentioned, multiple user stories were a collaborative effort. In fact, I would argue that the majority of our user stories were completed in collaboration, whether that was direct paired programming, mob sessions or refinement and review.

I've discussed a few collaborative user stories in-depth in the previous sections, these include: Software Architecture, Database Architecture, Remote Database & Database Functions. Even whilst collaborating, I would argue that these are some of the hardest stories that we completed this sprint.

Outside of those larger stories, I understood that my relative expertise would be best suited integrating and merging everyone's work. Though few, I did complete some stories in isolation. I assigned myself a couple of the smallest stories including Provide Introduction, Player Details & Victory Conditions. I chose these since they would be functionally close the main controller of the program/game, which is where most of my integration efforts would be focused.

Fortunately, these stories proved fairly trivial since their functions mostly just printed to console. In fact, only Victory Conditions provided a function with a testable return, being a Boolean. The unit test for this story is available at the end of this report.

Once I'd finished these stories, I had a decision to make. Either I would blaze ahead on more functionality or assist the others in completing their stories. Of the two, I chose the latter. I did this because I decided that rushing to add more features by myself was against the multiple agile principles we were honouring, principles like: Feature Driven Development, Kanban & Scrum.

Most notably, I assisted Liam and Nathan in reviewing and integrating the stories surrounding the user input. They did an amazing job building the structure of the functions, I simply joined them later and applied my relative expertise to trim and clean the code. Once we were satisfied, I made the final push and merge into the master.

Contribution

The sum of the story points that I am assigned on the sprint backlog is 38. However, most of these stories were completed in collaboration. Meaning, either everyone gets full credit for the completion of those stories, or we divide the points per contribution. I will admit that of those collaborative stories, I probably contributed more than the equal share.

This disparity in contribution is certainly not due to an individual lack of effort. Given that the assignment specification explicitly mentions: 'Each member must complete a number of tasks of a similar size and provide unit tests.', we made every effort to divide the work equally.

I believe the disparity was only possible due to the varying levels of expertise between the individual team members. Certainly, I found myself taking the lead during the collaborative sessions since I had the most expertise, I spent less time researching solutions and therefore completing my stories faster. Though as I previously mention I did not accept more tasks into the sprint backlog, my efforts in aiding the other team members should count towards my contribution.

Overall, I would suggest that my relative contribution was probably between 30-35%. I do believe that this number would have reduced given a longer timeframe as the individual expertise gap should reduce as we become more familiar with the task.

Reflection

Overall, I am very proud of the work this team has completed. Though the timeframe was limited, we held to correct procedure and produced a release built upon functional, well-tested, proven code.

Though I admit it is a shame that we couldn't develop more functionality, given that multiple days were required for design, testing & report writing, we understood that of the full week sprint, only a couple days were available for feature development. When compared against our predicted releases, we chose to focus on the quality of our first release rather than the quantity.

The short timeframe compelled the team to disregard a couple of the agile processes. Most notably, the 3-Amigos ceremony. Ideally, I would've like to have held this ceremony at least once in the early stages. Realistically, we could've invited a lecturer as an acting product owner the divided the team between the developer and tester roles.

Holding this ceremony would've facilitated the creation of acceptance criteria for the stories as they were being produced. With those criteria, we could've considered and potentially implemented a Behaviour Driven Development process instead of the FDD approach that we took.

In summary, I am very content with the team's overall approach to this challenge. Given our mixed expertise, I believe we have achieved the best possible result. Personally, I believe I focused more on the team than myself, facilitating their work instead of racing ahead. Though this might reflect in the quantity of code produced, I am confident in its quality. If given the same opportunity I would only attempt to further improve our agile methodology.

Evidence

```
@Test
void selectAllSuccess() throws Exception
{
    Connection conn = sqlFunctions.createConnection();

    String[][] all_expected =
    {
        {"TEST01", "Test1", "Test Desc 1", "true"},
        {"TEST02", "Test2", "Test Desc 2", "false"},
        {"TEST03", "Test3", "Test Desc 3", "true"},
        {"TEST04", "Test4", "Test Desc 4", "false"}
    };

    String[][] all_results = sqlFunctions.selectStatement(conn, "test_data");

    all_results[0][1] = all_results[0][1].trim();
    all_results[0][2] = all_results[0][2].trim();

    all_results[1][1] = all_results[1][1].trim();
    all_results[1][2] = all_results[1][2].trim();

    all_results[2][1] = all_results[2][1].trim();
    all_results[2][2] = all_results[2][2].trim();

    all_results[3][1] = all_results[3][1].trim();
    all_results[3][2] = all_results[3][2].trim();

    assertEquals(all_expected, all_results);
}
```

Figure 6 - SELECT ALL Unit Test

```
@Test
void selectStringSuccess() throws Exception
{
    Connection conn = sqlFunctions.createConnection();

    // room test
    String[][] room_expected = {"RM0000", "Test Name", "Test Desc"};
    String[][] room_results = sqlFunctions.selectStatement(conn, "room", "room_id", "RM0000");

    room_results[0][1] = room_results[0][1].trim();
    room_results[0][2] = room_results[0][2].trim();

    assertEquals(room_expected, room_results);

    // character test
    String[][] character_expected = {"CH0000", "Test Name", "Test Desc", "0", "0", "0"};
    String[][] character_results = sqlFunctions.selectStatement(conn, "character", "character_id", "CH0000");

    character_results[0][1] = character_results[0][1].trim();
    character_results[0][2] = character_results[0][2].trim();

    assertEquals(character_expected, character_results);

    // item test
    String[][] item_expected = {"IT0000", "Test Name", "Test Desc", "99", "false", "false"};
    String[][] item_results = sqlFunctions.selectStatement(conn, "item", "item_id", "IT0000");

    item_results[0][1] = item_results[0][1].trim();
    item_results[0][2] = item_results[0][2].trim();

    assertEquals(item_expected, item_results);

    sqlFunctions.closeConnection(conn);
}
```

Figure 7 - SELECT String Unit Test

```

@Test
void finishFalse() throws Exception
{
    Boolean expected = Boolean.FALSE;

    assertEquals(expected, program.checkFinished("RM0001"));
    assertEquals(expected, program.checkFinished("RM0002"));
    assertEquals(expected, program.checkFinished("RM0003"));
    assertEquals(expected, program.checkFinished("RM0004"));
    assertEquals(expected, program.checkFinished("RM0005"));

    assertEquals(expected, program.checkFinished("IT0001"));
    assertEquals(expected, program.checkFinished("CH0001"));

    assertEquals(expected, program.checkFinished("15"));
    assertEquals(expected, program.checkFinished("TRUE"));
    assertEquals(expected, program.checkFinished("disughbriven jGUEIDHNGVidgvndm"));
}

@Test
void finishTrue() throws Exception
{
    Boolean result = program.checkFinished("RM0006");
    Boolean expected = Boolean.TRUE;

    assertEquals(expected, result);
}

```

Figure 8 - Victory Condition Unit Test

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane shows a tree view of the database structure. The 'room' table is selected under the 'public' database. On the right, the 'Query Editor' pane shows a SQL query: `SELECT * FROM public.room ORDER BY room_id ASC`. Below the query, the 'Data Output' pane displays the results of the query in a table format.

room_id	room_name	room_desc
1	RM0000	Test Name
2	RM0001	Spawn
3	RM0002	the Mines of Moria
4	RM0003	the Library
5	RM0004	the Barracks
6	RM0005	the Throne room
7	RM0006	the Mickey mouse

Figure 9 - pgAdmin 4 Room Table

Assignment2_RhysBeckett

GRADEMARK REPORT

FINAL GRADE

60/100

GENERAL COMMENTS

Instructor

Software design evaluation (30): 18

Database evaluation and implementation (30): 14

Coding and Testing (30): 21

Reflection (10): 7

Some curious choices of diagram tools used for the database and software design. It would have been preferable to follow existing conventions for diagrams. Your diagram choice has made it difficult to interpret the database structure particularly.

Try not to solve the database requirements by thinking about the code-based transactions that you need to implement to make your procedures work.

Although the majority of this report is reflective, it is a little too informal in tone, even with this in mind.

A fairly good implementation overall - the code is clean, though would benefit from a few comments in places.

The database design is a little over-complicated and you haven't adequately discussed the need for such a complex set of joins - there's limited evidence of their testing or necessity.

Some good test functions generated, significant detail in granular testing of your functions.

A good code layout overall, separating functions into separate "classes".

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10
