

2013056 - Assignment 2 Block Test

by Owen Duncan Skilton

FILE	25302_OWEN_DUNCAN_SKILTON_2013056_- _ASSIGNMENT_2_BLOCK_TEST_312866_887816910.DOCX (737.6K)		
TIME SUBMITTED	11-DEC-2020 04:40PM (UTC+0000)	WORD COUNT	2800
SUBMISSION ID	139289660	CHARACTER COUNT	13844

LEVEL 7 ASSESSMENT SPECIFICATION

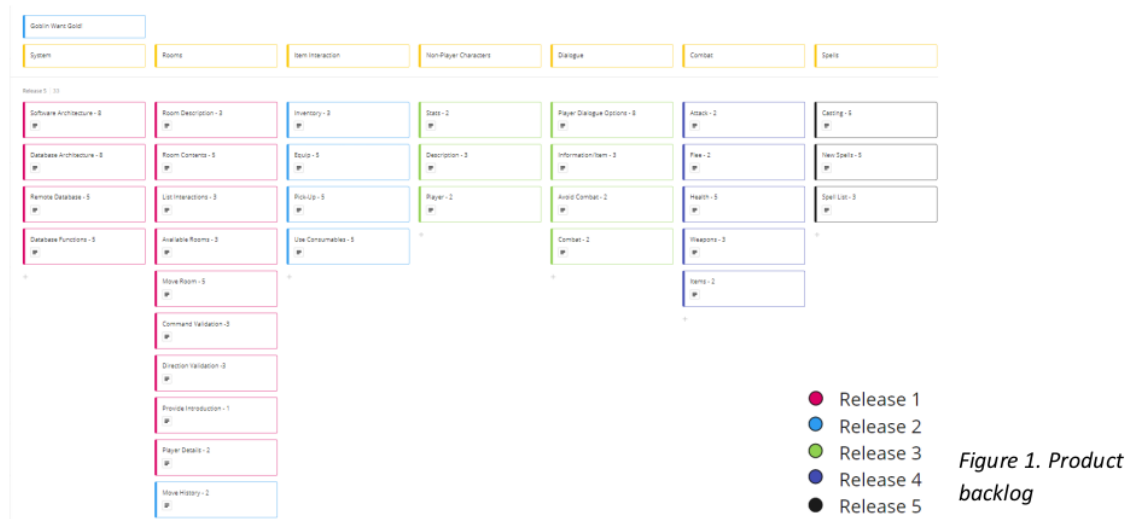
Student name:	Owen Skilton	Student ID number:	2013056
Programme:	Software Dev and Testing/Agile SEP/Database Systems		
Module:	ACCA7000/ACCA7001/ACCA7002		
Module code:		Contribution to Overall Module Assessment (%):	50% of all 3 modules
Lecturer:	Mike Dacey	Internal Verifier:	
Assignment Title:	Assignment 2: Block Test	Word count (or equivalent):	3000
Submission deadline:	11/12/20	Return date of provisional marks & written feedback:	
Submission method:	All written assessments, where practical and possible, must be submitted via Turnitin unless otherwise instructed by the Lecturer. (Please DO NOT put this assessment specification into Turnitin or it will match many similarities with other students' submissions.) Alternative submission method (if applicable): Late submission of the assessment will result in a late penalty mark. Penalties for late submission: Up to one week late, maximum mark of 50%. Over one week late, 0%. Only the Extenuating Circumstances Panel may approve a change to submission dates.		
Academic honesty / referencing:	Academic honesty is required. In the main body of your submission you must give credit to authors on whose research and ideas your work is based. Append to your submission a reference list that indicates the books, articles, etc. that you have used, cited or quoted in order to complete this assessment.		

Introduction:

During the last week between the 7th and the 11th of December, I along with 3 team members attempted to create a text based console application within java, based on a single player dungeon crawler. This report will present the methods we used during the 4 day sprint to develop the project architecture and database design as well as discussing in detail the code written and tests performed in order to complete the user stories. The team consisted of myself, Rhys Beckett, Nathan Sathanandan and Liam Roberts.

Software architecture and sprint planning:

The first day of the sprint was orientated towards the planning of the software architecture and the product backlog that we would attempt to perform over the coming week. The product backlog was formed by concentrating on the features specified within the design brief. The 4 main features requested included the ability to move rooms, talk and fight, pick up items and finally pick up and cast spells. From these we elected to use a feature driven development approach as we did not have the time within the sprint for BDD or TDD as the first 2 days of the sprint were used for project planning and database implementation. From the design brief we created 7 epics that would encompass 5 releases, each tailoring 1 -2 epics. These feature releases and epics can be seen within figure 1.



Unfortunately, due to time constraints, the team were unable to progress past the 2nd release. Release 1 was completed on the Wednesday afternoon and after discussion as a team we came to the decision to cease the designing for release 2 and instead focus on the implementation and in-depth testing of the features within release one. The project design architecture was done so in a way that creating the features in the future would simply build on top of the already created foundation of user input and database queries. We estimated that the features up to release 3 that include item interaction and dialogue could have been completed before the 5pm deadline, however, we also concluded that we did not have the time to thoroughly test these features and did not wish to release anything within the game that we did not feel comfortable had been tested thoroughly and could not guarantee that unnecessary bugs would arise from scrutiny.

The software architecture for the project was designed during this first day and went through a couple iterations during the week as more and more functions had to be added on for completeness. The focus was around the main program calling functions that were able to return large chunks of database info within a 3 dimensional array that was created by using SQL queries to return all the data relating to the room that the player currently occupied. This array could then be passed to other functions that extracts information depending on what was need such as room names or descriptions as well as IDs and valid inputs. From here these condensed arrays are passed to multiple functions including user input validation as well as text output and extracting only the necessary info.

[illegible]

For the sprint backlog, we elected to bring in the entirety of the first release which was orientated towards the foundations of the program such as the database connection and querying as well as the movement through the rooms and user input. The sprint back log can be seen below in figure 3. We also decided as a team to set the definition of done for any user story completion. We settled on a similar DoD as the DVLA which goes as follows:

- Code has been peer reviewed against the teams agreed standard.
- Story has agreed to have been completed from the perspective of the user, developer and a hypothetical product owner
- Functionality, integration and unit test (if applicable) have passed and been reviewed by fellow team members.
- The story has been developed using production quality code

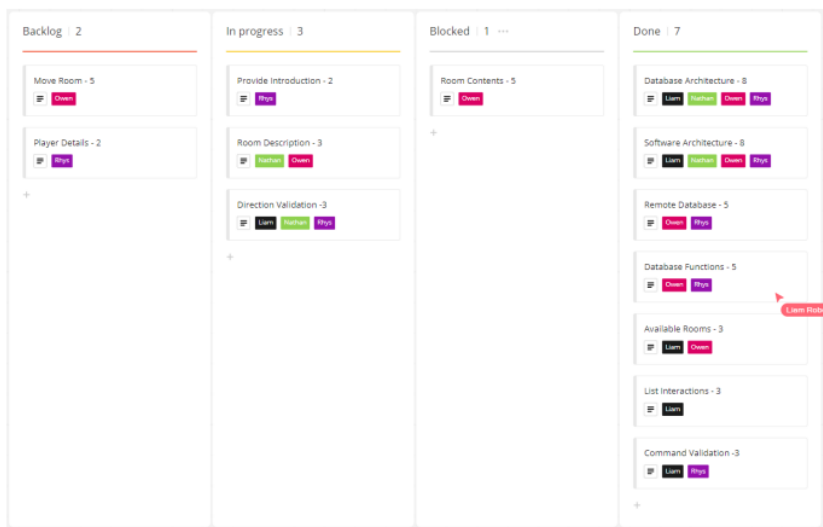


Figure 3, Sprint backlog

Database architecture

Upon completion of the product back log and sprint planning, the team then focused on the design of the database architecture. This was a joint effort and involved a lot of debate and discussion on how we wished the characters and game mechanics to interact with the database in the background. During the session we came up with the entities of the table and how they interact, plus identified the relationships between them. We settled on a design that was tailored around 3 main tables that when populated with data which would in turn interact with proxy tables to ensure that 3rd normal form across the database can be maintained. These 3 tables are Characters, room and items. The entire database design structure can be seen below in figure 4.

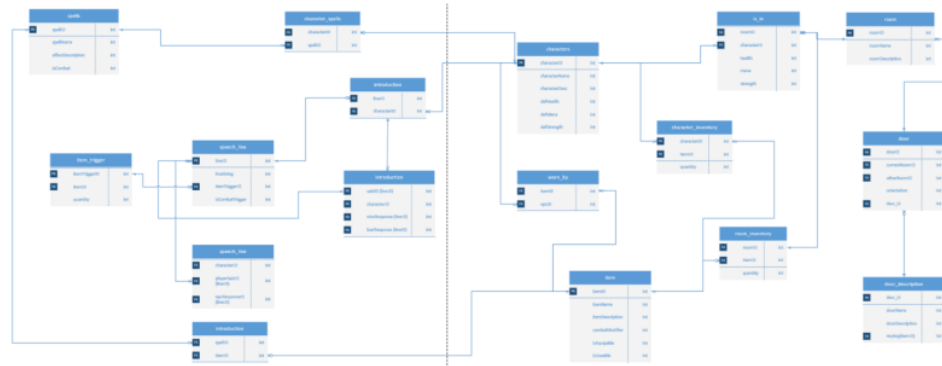


Figure 4,
Database
architecture

After the sprint planning session and database design, me and my fellow team member Rhys were tasked with discovering and implementing a way of hosting a database as well as java interaction for our game. I elected to take the hosting of the database portion as I had previous experience during our DVLA project with hosting applications as Rhys focused on the interaction between the database and the java application. The method I used for hosting was uploading a postgres database onto Heroku and then adding it as a server within PgAdmin4. Once I successfully set it up through my Heroku account I shared the database hosting platform and password with my team while Rhys created the connection string so all future function creation involving the database can simply be passed this connection string and would only require one connection request.

Once it was hosted successfully and everyone had successfully connected we begin creating the tables through Postgres and applying the constraints to them. I was tasked with creating the dialogue portion of the table and the interaction between them and the character. As with the item interaction within this table, these unfortunately are not utilised in the final product but have the infrastructure set up to be used in future. Once all the tables were created we then started populating the database with both test data for unit testing in the future and the actual data used within the finished product.

Coding:

The 3 user story task I'm going to focus on in this section is Room contents, Move room and room description.

RoomContents was designed as a user story for passing information to functions throughout the program from the database. It as a user story weight of 5 and is formed as a developer focused story for room information. It is designed as a multidimensional array that is formed using 4 separate SQL queries. This array consists of all the information of the doors within a room, the items contained within it, the characters and information about the room itself. All 4 of these are sub functions are essentially SQL queries to the database that populate this 3D array. 3 of them are inner join queries so that the data is formatted nicely once it's passed to other functions and can be read in a more developer friendly manner. This array created is passed to any function that requires information contained within the database for that room and eliminates the need to perform multiple calls to the database and instead takes the information from the array which is much faster and formatted much more friendly. Below in figure 5 shows an example of the getDoors() function that passes the 2 tables they wish to inner join as well as the column to apply a filter to. For every function this filter is currentRoomID, this means only one parameter needs to be

passed to roomContent other than the connection string for all the information that is needed to be known. The pattern for getDoors is repeated for the other 2 inner join functions and a select query for the room information.

```
public static String[][] getDoors(Connection conn, String currentRoomID) throws SQLException
{
    try {
        String[][] results = sqlFunctions.doorsOnDoorDescription(conn, door: "door", door_description: "door_description", (column: "current_room_id", currentRoomID));
        return results;
    } catch (SQLException e) {
        e.printStackTrace();
        throw e;
    }
}
```

Figure 5, getDoors function.

Room description is designed as a player orientated user story with a weight of 3 that is described as the player wanting to be told the description of a room they've entered so they know what is able to be interacted with. In planning this appears to be a simplistic function however in order for the text output to be as simplistic as possible, the array needs to be formatted very specifically so can be simply read through using for loops. This new array is 2D instead of the 3D array of RoomContents and has 10 aspects of information stored with in it. All the information is console output focused so doesn't contain things like IDs and properties. This user story was pair programmed with team member Nathan where he focused on the text output method and I focused on the array formation aspect. The 3D array from the RoomContent function is passed in to all the sub functions that generate the 2D array getCurrentRoomAllContent. The 3D array was designed so that all the information is parallel within the array and is always located in the same position no matter the contents of the room. For example, if a room didn't contain an item, the 2nd 2D array saved to the 3D array is left blank so that the character and room information remains in position 3 and 4 respectfully. This ensures that all functions that use this information don't have to be altered if some data is missing so they will always remain in the same spot regardless of the room contents. This applies to all the subfunctions that populate getCurrentRoomAllContent. All they require is to iterate through the 3D array within a specific pattern and all the information is saved to the 2D array and ready to be output in the text output function. Below in figure 6 shows an example of the sub function that extracts the character name.

```
public static String[] getCurrentRoomCharacterName(String [][] roomContent)
{
    String[] Temp = new String[roomContent[2].length];
    for (int i = 0; i < roomContent[2].length; i++)
    {
        Temp[i] = roomContent[2][i][5];
    }
    return Temp;
}
```

Figure 6, getCurrentRoomCharacterName function

The final story was the Move Room aspect of the game. It has a weight of 5 and is a player orientated story described as wanting to move between rooms so they can progress through the game. This function required discussion with the rest of the team as it had to be interactable with user input. It takes the currentRoomID and the direction specified by the player and performs a search within the doors table to find the adjacent roomID in the specified direction. CurrentRoomID is then overwritten with this ID that was just found and the process is repeated when a move input is flagged from the user input. This function was completed on my own but required constant communication with fellow team members working on the user input to ensure the format of the string passing through was correct. Below in figure 7 shows the current ID override and figure 8 the MoveRoom function.

```
currentRoomID = Move_room.MoveRoom(conn, command, currentRoomID);
```

Figure 7, currentRoomID being overridden by the return of MoveRoom.

```
public static String MoveRoom(Connection conn, String User_direction, String Current_room)
{
    String room_moving_to = "";
    try {
        String[][] results = sqlFunctions.selectStatement(conn, table: "door", column: "orientation", User_direction);
        for (int i = 0; i < results.length; i++) {
            if (Current_room.equals(results[i][1])) {
                room_moving_to = results[i][2];
                break;
            }
        }
        // System.out.printf("%s", room_moving_to);
        Current_room = room_moving_to;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return Current_room;
}
```

Figure 8, MoveRoom function

Testing:

Unit testing was performed on each of the 3 functions above upon their completion. For both `getCurrentRoom` and `RoomContent` I tested the sub functions that construct them individually against the test data that we inputted at the start of the week during the database architecture. For example, for the `getDoors()` function, I would set my expected array to consist of the test door ID, the 2 roomIDs that connect them together, the orientation for that door along with the descriptions and names. The actual results would be running a direct database query within the door table using the current room filter "RM0000". This pattern was repeated continuously for the 4 sub functions for `RoomContent` and the 10 for `getCurrentRoom`. Below in figure 9 and 10 shows the formation for a sub function test of `RoomContent` and `getCurrentRoom` respectfully:

```
@Test
void getDoorsTest() throws Exception {

    Connection conn = sqlFunctions.createConnection();

    String[][] room_expected = {{ "DR0000", "RM0000", "RM0000", "N", "DESC00", "test name", "Test Desc", null }};
    String[][] room_results = sqlFunctions.doorsOnDoorDescription(conn, door: "door", door_description: "door_description", column: "current_room_id", filter: "RM0000");

    room_results[0][0] = room_results[0][0].trim();
    room_results[0][1] = room_results[0][1].trim();
    room_results[0][2] = room_results[0][2].trim();
    room_results[0][3] = room_results[0][3].trim();
    room_results[0][4] = room_results[0][4].trim();
    room_results[0][5] = room_results[0][5].trim();
    room_results[0][6] = room_results[0][6].trim();

    assertEquals(room_expected, room_results);
}
```

Figure 9, sub function `getDoors` tested within test file for `RoomContent`

```
@Test
void getCurrentRoomDoorName() throws Exception {

    Connection conn = sqlFunctions.createConnection();

    String[] room_expected = {"test name"};
    String[][] room_results = RoomContent.roomContent(conn, currentRoomID: "RM0000");
    String[] Temp = new String[room_results[0].length];
    Temp[0] = room_results[0][0][5].trim();

    assertEquals(room_expected, Temp);
}
```

Figure 10, sub function `getCurrentRoomDoorName` tested within test file `getCurrentRoom`

A similar test was also performed on the move room function that ensured that only the correct roomIDs were returned from the SQL query performed. I didn't have to test for error handling or a wrong orientation input as this sort of validation is already performed during the user input phase which ensures only correct orientations can lead to a change in room state. This simple test can be seen below in figure 11 where I am within roomID "RM0005" and wish to go north or south to "RM0006" and "RM0004" respectfully.

```
//room 5
assertEquals(room_expected[5], Move_room.MoveRoom(conn, User_direction: "N", Current_room: "RM0005"));
assertEquals(room_expected[1], Move_room.MoveRoom(conn, User_direction: "S", Current_room: "RM0005"));
```

Figure 11, `MoveRoom` test for possible outcomes of currentRoomID "RM0005"

Reflection:

The positive aspects of the agile process utilised is that once the sprint back log and project architecture planning was completed it was very easy to get large amounts of code completed and submitted the repository very quickly with the definition of done and review process implemented. The problems however are that the agile process took

a day away from the sprint which could have been spent implementing and testing more features for the game itself. Unfortunately, a day was needed for product design and the 2 architectures to be created and half a day was needed to establish the database hosting, so the team only had 2 ½ days remaining worth of coding totalling 20hours in a 9-5 day. I am pleased with the work I completed and the future proofing of the functions I've designed however wish we had just an extra to implement more features that those functions could be used for. The focus on large foundation building was a time consuming effort however we believe is the correct method in terms of agile at tackling the project as a whole. If any team were to pick up this project, they would be able to build upon the code already created to implement the rest of the features into the game.

The team dynamic throughout the project was excellent. At all times individuals were offering assistance or completing user stories. That goes for the architecture as well, at all points during the design portion, team members were contributing their opinions and offering solutions to design problems. The only stories that I undertook individually were the move room function and 3D array querying from the database, everything else I work on I had a team member working on a different aspect at the same time and was always a point to turn to during the designing of the user stories at the start.

2013056 - Assignment 2 Block Test

GRADEMARK REPORT

FINAL GRADE

63/100

GENERAL COMMENTS

Instructor

Software design evaluation (30): 19

Database evaluation and implementation (30): 17

Coding and Testing (30): 21

Reflection (10): 6

The database design documentation is overly convoluted - as the diagram stands, it indicates a significant amount of data duplication (e.g. multiple tables called "introduction"). It seems you have over-complicated this problem.

The user story mapping diagram is not clear.

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
