|  |
| --- |
| DAT602 Project Report Milestone 2 |
|  |

|  |  |
| --- | --- |
| NMIT | Celeste Quinn |

Project Report for Milestone 2 of 3 for DAT602 Project

DAT602 Project Report  
Milestone 2

Celeste Quinn

Table of Contents

[Game Activities 1](#_Toc73890666)

[Activity Identification 1](#_Toc73890667)

[Register and Login 1](#_Toc73890668)

[Character Management 2](#_Toc73890669)

[Admin 2](#_Toc73890670)

[Movement 2](#_Toc73890671)

[Battle 2](#_Toc73890672)

[Interact 2](#_Toc73890673)

[Inventory Management 3](#_Toc73890674)

[Chat 3](#_Toc73890675)

[Logout 3](#_Toc73890676)

[Activity Procedures 3](#_Toc73890677)

[Multiplayer Gameplay 3](#_Toc73890678)

[Test Data 5](#_Toc73890679)

[Changes 5](#_Toc73890680)

[References 6](#_Toc73890681)

# Game Activities

## Activity Identification

### Register and Login

The first activity a player needs to do to access anything else in the game. Register includes signing up with a username and password. This creates an account that can be logged into. Logging in requires the user to enter a valid username and password.

Character Management

Once logged in the player is presented with the options as shown in Diagram 5 of M1 Report – A list of the character names and options to play and delete. This is how they select their player and confirm entering the game.  
Upon review I feel another option needs to be added here – edit. Which presents you with the edit player options that only the admin could previously access (D6 M1), so the player can update their own account details and add an email address, but obviously locking the account and isAdmin checkboxes would be disabled for regular players.

Admin

Admin functionality includes the ability to edit player’s information, lock accounts, make them admins, kill a session, create a new player, and delete a player.

Movement

When the player is in the game their player character must be able to move in either North, East, South, West. Movement is restricted to the squares directly around the tile they are currently on. Movement requires the use of one point of stamina, a character stat that can be increased. Depending on how difficult this is to implement, this may be removed by the final project.

Battle

If adjacent to a tile with another player or NPC on it, the player can initiate a fight where their stats are pitted against the opponent to find a winner. This is how they gain coins and sometimes items. I am as of yet undecided about a levelling system. My original plan was to use the items to adjust the players stats, therefore the better items the player had, the higher their stats were, but there’s also the classic experience system where a battle would reward experience and the player gains higher stats each level they gain. It will come down to implementation to fully decide on this.

Interact

A player can find items on the map randomly and possibly also interact with shop objects that appear randomly on the map and can use their coins to purchase items from. On paper this seems easy enough to implement, but also not sure yet. So the only solid interaction the game has would be collecting items off the ground as they move. So there needs to be a command for this, and a GUI prompt to show the player they are able to interact with something nearby.

Inventory Management

Clicking the inventory button from the main GUI should open a frame that shows what items the player has obtained, including their descriptions and value, and if applicable what actions they can take with the items.

Chat

Pressing the enter key (universal standard) or perhaps clicking on the input area on the chat box, would allow the player to talk to anyone else in the game (enter to send the message as well) If this were to be developed further there would have to be some options of reporting or ignoring a player you do not want to hear, but that is beyond the scope of this project.

Logout

Ends the session and saves the state of the game so the player can login next time and return to exact how they left off.

## Activity Procedures

Stored procedures are saved in the SQL file ‘database creation and procedures’ attached.

Player registration procedure: addCheckedUser (not used in visual studio currently), AddUser  
Player selection: pickClass  
Live gameplay: move  
Confirmation for a game/show online players: showPlayers  
Game admin functions: editAcc, delAcc

# Multiplayer Gameplay

Multiplayer gameplay will be supported because all of our interaction transactions are following the ACID protocol when they are executed. A transaction is a collection of procedures or sql statements that work together as one and is treated as one whole transaction. The acidity of transactions make them perfect for use with multiplayer games as multiple functions working as a single operation is ideal for how reliable it needs to be.

Lets take a couple of transactions as an example to use while relating them to being acidic, a transaction for a fight, and a transaction for moving.

Firstly, they have to be **Atomic.** This means that since everything inside the transaction works as one, everything has to either be all successful or fail completely. This means that if a transaction had 10 different statements inside it, and 9 succeeded but one failed, the whole thing would fail. The transaction is not allowed to be only partially working. In terms of our fight transaction here, it would hold all the statements necessary to work out the outcome of a battle. It would compare the players stats with the opponents stats, calculate damage based on that, report the outcome to any players involved and update their games accordingly. Not a single step in this is allowed failure, it must follow that flow or it will fail entirely – like we can’t just leave it in mid fight.

Secondly, the transactions must be **Consistent**. Which is to say that the data must obey the rules and stay in a consistent and valid state. One example for this is that in some games, a player might try to use hacks to change their stats or gold level (mostly only achievable in client-sided games) So consistency here would prevent someone from going in and changing their stats as that would not reflect on the records the database has, because if the data is not consistent, any changes will be rolled back and data integrity is maintained. Another is that gold value cannot be negative, only positive, so if a player was trying to buy an item from the store, they cannot buy something that is over the value of gold they have on hand, as it would bring their balance into the negatives. If this is attempted, the transaction is cancelled. The transactions the game have must account for situations like this.

Next, **Isolation**. This seems to have a few different meanings depending on which website is explaining it. One is that the transactions are not able to be interfered with by other transactions while they are busy doing their thing, or in an intermediate state. The transactions are not dependent on one another, so their isolation means that you wouldn’t need one transaction’s success to be reliant on another transaction that was happening at the same time, to also succeed beforehand, and also that the outcome of multiple transactions happening at the same time, will not affect their ability to succeed. The moving transaction would be a good use case to relate to isolation, as multiple players will be moving around the map at the same time, the transactions must ensure that no one is impacted by other’s movements, that everyone can successfully move around, asides from the rule that you can’t be on the same time as another player. The SQL must meet this rule.

Finally our transactions must have **Durability.** Which means that after a transaction has completed. The changes are permanent and will not be reverted if anything happens to the server/system/database. This is one of the most important things in gaming, especially online gaming, as server outages or disconnecting is one of the most annoying problems. Having a rollback of your progress because of a crash is every gamers worst nightmare, especially if something happens like, you’ve just killed a really hard boss, or you’ve received an ultra rare drop. The last thing you want is that being taken away from you. So for the transactions to have durability, they need to be not able to be undone.

At this point in the Assignment there are no transactions to ensure this, however there is some basic groundwork so far that could count as multiplayer support. This is in the form of the first iteration of the move() procedure. If the player attempts to move onto the coordinates that is occupied by a user already – there will be an error message displayed and they will be unable to complete the move.

# Test Data

C# work will be shown in the attached file “datconnection” solution.

In the visual studio project, the test data is inside the Program class, and dataAccess class. These will not be renamed as they were creating during a class session using the names we used then for testing, and I kept going on these files. I have learned the hard way that renaming things in visual studio breaks your whole solution because the process of renaming it doesn’t change everywhere that the file is already stored in other parts of the solution and I’ve already lost work attempting this for other activities.

Anyway the methods that call the procedures from mysql are in DataAccess, and then you can find where to input information in Program, where you can change the variables. As long as you follow the conventions for input as defined in the comments, there should be no errors. It comes with predefined data so you can just build and start straight away, and press enter in the console window that appears to continue to the next function after one has completed, it will give dialog queues. You can then check the output in mysql workbench to ensure the data has been written successfully to the database.

The procedures that have been written for this milestone, should cover the basic functions that the game needs to have in this early development stage. Registering a user account, being able to edit details, picking a class for the user, and then moving- as well as the extra procedures like showing players and deleting accounts. With no interface it can be tricky to visualise how these will work with the game in the end, and by this I mostly mean something like login validation. After taking web last year it seems to me this might have to be done with something like an outside authentication server like firebase? I don’t know. Maybe we have covered it in class and I’ll have to go back through and look to see if it can be done simply in mysql or not. It’s hard to remember everything if you’re not up to the point where you’re actually putting it in the development of your game actively.

Further procedures will have to be developed in future to include more of the activities as described in the above section in this report.

# Changes

A list of changes and reasoning from M1 to M2

* Added charID FK into table ‘account’  
  The account needs to know what characters have been created on it. Not sure if this also means that userID key should be removed. Currently userID is account’s PK and is an FK on character as well, so it shows what userID the character is associated with. This is somewhat confusing to explain – so – there is one account that belongs to the user that has the ability to have multiple characters on it. The account has a userID. And each character also has a characterID. Each character has a reference to the account they were created on via the userID, so each account has a list of charID’s that are on it. So unsure if both are really necessary.
* Added new column ‘status’ into character table  
  This is so we can complete the required query for confirmation of game, instead this will show the currently online players. So characters need to have a status showing them online or offline, and will be updated accordingly when a character joins the game.
* Deleted character table entirely and linked up the old rows that were still needed into account table. This was annoying because having an account and trying to link up the characters was complicated. It worked to an extent, I could link a new character to the last account that was created, but as there is currently no way to log in to an account and then be in it while creating a character to link it like that or something, like I just have no idea how that would have worked. This pretty much deprecates the above changes I’ve made but will still leave them in here for posterity. It also means I won’t have a procedure for the criteria of Player selection, as that was the old createChar() procedure which I’ve removed, so have to think of something else for that I guess.
* Added a template table that new characters must choose from when first starting, that determines their stats. This takes the form of 3 different builds a player can choose from. They are Berserker, which has an increased strength, Tank which has increased hitpoints, and Ninja which has increased stamina.
* Deleted map table as it was not being used

# References

Watts, S. (n.d.). *ACID Explained: Atomic, Consistent, Isolated & Durable*. BMC Blogs. Retrieved May 18, 2021, from <https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/>

ACID versus BASE Data Stores. (n.d.). *Dummies*. Retrieved May 18, 2021, from <https://www.dummies.com/programming/big-data/hadoop/acid-versus-base-data-stores/>

Roe, C. (2012, March 1). ACID vs. BASE: The Shifting pH of Database Transaction Processing. *DATAVERSITY*. <https://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/>

*IBM Docs*. (2021, March 29). <https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions>