**Detailed Design Document**

Software Engineering Methods - SW 201/400

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| Introduction………………………………………………………………………….. | [Page 2](#id.pzx4watz5c6d) |
| Architecture………………………………………………………………………….. | [Page 4](#id.fb82ggljpcor) |
| Communication Module Design……………………………………………………... | [Page 6](#id.35cz2fveb0eo) |
| Game Module Design………………………………………………………………... | [Page 20](#id.fv9g2qp4xdqp) |
| Database Design………………………………………………………………...…… | Page 38 |
| Abstract Data Types………………………………………………………………….. | Page 42 |
| Comprises Diagrams…………………………………………………………………. | [Page 4](#id.lqxluaowqty2)4 |
| USES Diagram………………………………………………………………………. | [Page 4](#id.hmny6prw1c5v)8 |
| Class Diagram………………………………………………………………………... | [Page 49](#ClassDiagram) |
| Integration Test Plan…………………………………………………………………. | [Page 5](#id.an5zswew4yp6)0 |
|  |  |

**INTRODUCTION**

This is the official design document for the application *Hello World: An Introduction to Computer Programming Concepts*. The purpose of this document is to expand upon the requirements defined in the Software Requirement Specification document and provide high-, mid- and low-level design models of the proposed system. The application will include the following main features:

* *Multi-Device Access:* users will have the ability to create a user account in order to save and track their progress in the game. Additionally, the user will be able to access and modify their personal information on the account, such as their username, password, and email address. This data will be stored in a centralized database that can be accessed via an Internet connection to synchronize user progress among multiple devices.
* *Captivating Graphical User Interfaces:* in order to minimize required training and maximize user interest, the application shall utilize visually-attractive user interfaces that, apart from the game itself, contain only the functionalities that are necessary to successfully interact with the application.
* *Classroom Usability:* other than for personal entertainment and an individual desire to learn about programming concepts, this application can be used in parallel to topics taught in a classroom environment. Teachers will have the ability to track their students’ progress and generate class reports.

In order to design a successful application within the time frame specified by the company, the team shall make effective use of third-party software and object-oriented programming techniques when designing the software. This will provide the team with the ability to take well-known game design techniques that have been proven to be successful and modify them so they can be incorporated into our game.

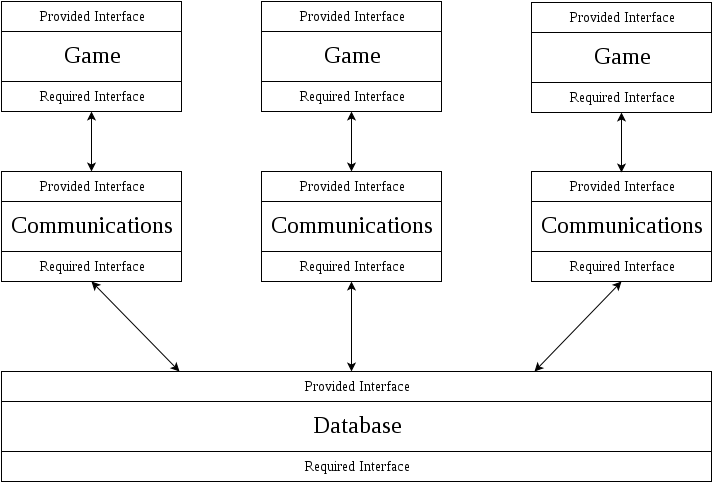
Our system will consist of three high-level components: a Database component, a Communications component, and a Game component. The Database component will contain all information relating to creating user accounts, updating user accounts, and generating reports. Users who create an account will have the ability to access information about their account in the Game. Data will be sent to and from the Game and the Database through a Communications module, which contains all intermediate processes that require data to successfully be transmitted between these two components.

The user will only be able to directly interact with the Game component of the system. This component will implicitly interact with the Database component in order to incorporate user data into the game. Within the application, the communications between the Game component and the Database component will be encapsulated into abstract objects that the user interacts with, such as menus and buttons. This ensures that the data transferred between the two components is original and no unexpected modifications have be made to it by the user.

The remainder of this document contains a more in-depth breakdown of how our system will interact with both the user and other components within the system and uses module designs to help explain how these components will work together.

**ARCHITECTURE**

Figure 1 shows a high-level architectural design of the system:



*Figure 1: High-Level Architectural Design of the Proposed System*

We will utilize both the Client-Server and Layered architectural styles in order to allow multiple instances of the Game and Communications components (the clients) to perform CRUDE (create, read, update, delete, and execute) operations on the Database (the server).

During the requirements gathering stage of the system lifecycle, the project sponsor informed the team that the system must include two main functionalities: an entertaining game and a system that allows users to create accounts and save their progress. In order to meet these requirements and exercise the many benefits of data encapsulation, the system will include three components, each of which is further defined in both natural language and diagrams in succeeding sections of this document.

The Game component will consist of the actual game itself; the user will directly interact with this component through a variety of inputs that depends on the device the user is using. The Communications component is responsible for handling all events relating to user accounts within the Game component. If the user performs any action involving their user account, the game will use objects and methods provided by the Communications system. The user will never have direct access to this component; they can only indirectly access it through the Game component, which will encapsulate the functions of the Communications component. The Database component will store all of the information relating to user accounts. It would be inefficient and unsafe to directly access the Database from within the Game, which is why we will design this process using the Layered architecture style.

We chose the Client-Server and Layered architectural styles over all other styles for several reasons. The team feels that allowing users direct access to the database is a security issue that can potentially become increasingly critical as more users use this system. We are not concerned with performance specifications pertaining to transferring data between the Game and the Database because we anticipate that the data being sent will be small (it will only consist of small amounts of changes at a time). Taking this into consideration, we believe that the Layered architectural structure will provide an extra layer of security when accessing the database. This system also needs the ability to work when a stable connection to the database is not available. To circumvent this issue, data related to the game will be locally stored on the device and only updated on the server when a stable connection is available. We anticipate many instances of the game having to update and query data on the database at the same time, so we chose the Client-Server architectural style for this aspect of the system to efficiently allow this to happen.

**COMMUNICATIONS MODULE DESIGN**

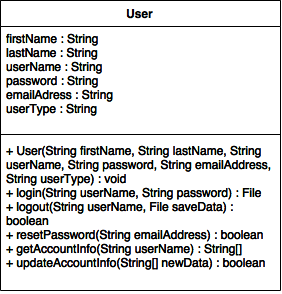
Figure 2 shows a mid-level modular view of the class hierarchy within the Communications component of the system. Following the diagram is a low-level breakdown of each module within the component.

Figure 2 -- Communications Design.png

*Figure 2: Mid-Level Module Architecture of the Database Component*

This modular hierarchy of the Communications component is crucial for providing users of different types with different sets of capabilities. If the Communications component were to contain only one User class, then every type of user would be assigned the same functionalities. This would hinder the system’s potential and create security risks (such as giving a general user the same capabilities as an administrator). By defining submodules that inherit the attributes and functionalities of *only* their ancestors, we are able to create a clear separation of concerns among users of different types. The hierarchy also allows us to create an Admin submodule that provides ultimate control of the Database to a user that inherits the same functionalities as a general user.

**User**



*Figure 3: Low-level breakdown of the User class*

Purpose:

The purpose of this module is to allow the application to create, modify, and retrieve data located in the Database component. This class is the superclass of all other classes within the Database and contains attributes and methods that are used in all types of user accounts. The User class addresses the requirements that allow the user to create and update a user account to track their progress in the game.

Provides Interface:

**public User(String firstName, String lastName, String userName, String password, String emailAddress, String userType)**

*Description:*

Retrieves data from the Game component that is required to create a new user account and adds it to the Database. Depending on the user type, which is part of the parameters, a subclass of the User class may be called.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (student, instructor, or general).

*Returns:*

N/A (constructor).

**public File logIn(String userName, String password)**

*Description:*

Checks if the credentials entered by the user are valid in order to access their account. If so, returns the save file associated with the specified account.

*Parameters:*

String userName: the username associated with the account.

String password: the password associated with the account.

*Returns:*

A File object containing the most recent save file. If the account does not exist, returns a null value.

**public boolean logOut(String userName)**

*Description:*

Initiates the logout sequence for the account.

*Parameters:*

String userName: the username associated with the account that is logged in.

*Returns:*

A boolean value - true if the user is successfully logged out, false otherwise.

**public boolean resetPassword(String emailAddress)**

*Description:*

Allows the user to reset the password associated with their account. The Game component passes the email address entered by the user to this method, which is queried in the Database component.

*Parameters:*

String emailAddress: the email address associated with the account.

*Returns:*

A boolean value - if the email address is found, creates a temporary password and sends it to the specified email address and returns true. Otherwise, returns false.

**public String[] getAccountInfo(String userName)**

*Description:*

Queries and returns the account information for a particular user account.

*Parameters:*

String userName: the username associated with the account.

*Returns:*

An array of Strings containing all of the information associated with the specified account.

**public boolean updateAccountInfo(String userName, String[] newData)**

*Description:*

Accepts information passed to this method by the Game component and updates the information in the database.

*Parameters:*

String userName: the username associated with the account.

String[] newData: an array containing the updated data.

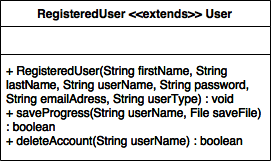
*Returns:*

A boolean value - true if the data was successfully updated, false otherwise.

Requires Interface:

N/A

**RegisteredUser**



*Figure 4: Low-level breakdown of the RegisteredUser class*

Purpose:

The purpose of this module is to further define the type of user that is using the system. To be classified as a registered user, one must have created an account. Upon doing so, the new RegisteredUser object will contain all of the attributes and methods of its superclass (User). This class is the superclass of Student and Instructor, and therefore contains the attributes and methods that are used in those specific user accounts. The RegisteredUser class addresses the requirements that allow the user to create and update a user account to track their progress in the game.

Provides Interface:

**public RegisteredUser(String firstName, String lastName, String userName, String password, String emailAddress, String userType)**

*Description:*

Retrieves data from the Game component that is required to create a new RegisteredUser account in the Database.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (student, instructor, or general).

*Returns:*

N/A (constructor).

**public boolean saveProgress(String userName, File saveFile)**

*Description:*

Saves the user’s latest progress in the game to the Database component.

*Parameters:*

String userName: the username associated with the account.

File saveFile: the file containing the user’s progress in a specific format that is unique to the Game component.

*Returns:*

A boolean. Returns true if the save file was successfully updated in the Database. Otherwise, returns false.

**public boolean deleteAccount(String userName)**

*Description:*

Deletes the account in the Database associated with the username passed to the method.

*Parameters:*

String userName: the username affiliated with the account to be removed from the Database.

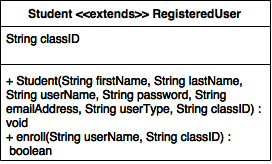
*Returns:*

A boolean value. If the account is successfully removed from the Database, returns true. Otherwise, returns false.

Requires Interface:

N/A

**Student**



*Figure 5: Low-level breakdown of Student class*

Purpose:

The purpose of this module is to further define the type of user that is using the system. To be classified as a Student, the Game component must pass a user type of “Student” and a valid class identifier to the Student constructor. This class is a subclass of RegisteredUser, and therefore utilizes the attributes and methods previously defined in the RegisteredUser class, which also makes it a subclass of the User class. The Student class addresses the requirements that allow the user to create and update a user account to track their progress in the game.

Provides Interface:

**public Student(String firstName, String lastName, String userName, String password, String emailAddress, String userType, String classID)**

*Description:*

Accepts data from the Game component that is required to create a new Student user account in the Database.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (student, instructor, or general).

String classID: the identifier provided to the student by their instructor.

*Returns:*

N/A (constructor).

**private boolean enroll(String userName, String classID)**

*Description:*

Checks the validity of the classID entered by the student and emails the instructor informing them that this student has enrolled in their class.

*Parameters:*

String userName: the username affiliated with the student’s account.

String classID: the class identifier code provided to the student by the instructor.

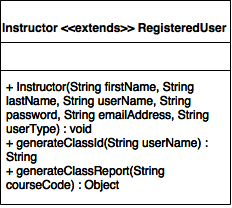
*Returns:*

A boolean. If the student is added to the class and the teacher is informed about the student enrolling in their class, returns true. Otherwise, returns false.

Requires Interface:

N/A (inherited from the RegisteredUser class).

**Instructor**



*Figure 6: Low-level breakdown of Instructor class*

Purpose:

The purpose of this module is to further define the type of user using the system. To be classified as an Instructor, the Game component must pass a user type of “Instructor” to this class’s constructor. If this is done, then a new Instructor account is created and the user has access to all of the capabilities of the User and RegisteredUser superclasses as well as the ability to generate reports on the progress of the students whose accounts contain a matching course code. The Instructor class addresses the requirements that allow the user to create and update a user account to track their progress in the game.

Provides Interface:

**public Instructor(String firstName, String lastName, string userName, String password, String emailAddress, String userType)**

*Description:*

Accepts data from the Game component that is required to create a new

Instructor user account in the Database.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password used to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (student, instructor, or general).

*Returns:*

N/A (constructor).

**public String generateClassID(String userName)**

*Description:*

Generates a new class identifier for the Instructor account.

*Parameters:*

String userName: the username associated with the instructor’s account.

*Returns:*

A string containing the class identifier.

**public Object generateClassReport(String courseCode)**

*Description:*

Retrieves the progress from the Database of all Student accounts with the courseCode attribute passed to this method.

*Parameters:*

String courseCode: the course code associated with the course that the Instructor wishes to generate a report on.

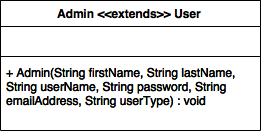
*Returns:*

A formatted Object containing the progress of all of the students associated with the courseCode attribute passed to the method.

Requires Interface:

N/A (inherited from RegisteredUser class).

**Admin**



*Figure 7: Low-level breakdown of Admin class*

Purpose:

The purpose of this module is to provide certain users with direct access to the Database containing information in regards to user accounts. Users are classified as an Admin by the development team and only have access to the Database component (not the Game component). If one is classified as an Admin, they are able to generate usage reports of the game and add, update, and remove user accounts. This class is a subclass of User, and therefore utilizes the attributes and methods previously defined in the User class. The Admin class addresses the requirements that allow a user with administrator privileges to create, update, or remove a user account to track their progress in the game.

Provides Interface:

**public Admin(String firstName, String lastName, String userName, String password, String emailAddress, String userType)**

*Description:*

Retrieves data from the system that is required to create a new Admin account in the Database.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (Admin).

*Returns:*

N/A (constructor).

Requires Interface:

**public boolean addUser(String adminUserName, String adminPassword, String newUserFirstName, String newUserLastName, String newUserUserName, String newUserPassword, String newUserEmailAddress, String newUserUserType)**

*Description:*

Allows the Admin user to override the automated system processes of adding a user to the Database and manually add a user to the Database.

*Parameters:*

String adminUserName: the username of the Admin user creating the new account.

String adminPassword: the password associated with the Admin user.

String newUserFirstName: the first name of the new user.

String newUserLastName: the last name of the new user.

String newUserUserName: the desired user name to be associated with the new account.

String newUserPassword: the password for the new user account.

String newUserEmailAddress: the email address to be affiliated with the new user account.

String newUserUserType: the type of the new user account.

**public boolean editUser(String adminUserName, String adminPassword, String firstName, String lastName, String userName, String password, String emailAddress)**

*Description:*

Allows the Admin user to manually update information regarding an account stored in the database.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

**public boolean resetUserPassword(String adminUserName, String adminPassword, String userUserName, userEmailAddress)**

*Description:*

Allows the Admin user to manually initiate the process that resets the password affiliated with an user account.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

String userUserName: the username affiliated with the account whose password shall be reset.

String userEmailAddress: the email address affiliated with the account whose password shall be reset.

**public boolean removeUser(String adminUserName, String adminPassword, String userUserName, String userEmailAddress)**

*Description:*

Allows an Admin to manually remove an account in the database.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

String userUserName: the username associated with the account to be removed.

String userEmailAddress: the email address associated with the account to be removed.

**public String generateClassroomID(String adminUserName, String adminPassword, String instructorUserName)**

*Description:*

Allows the Admin user to generate a new classroom identifier to be associated with an instructor’s account.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

String instructorUserName: the username of the instructor account that will be given a new classroom identifier.

**public Object generateReport(String adminUserName, String adminPassword, Object fields, Date startDate, Date endDate)**

*Description:*

Allows the Admin user to generate a report containing information from the Database.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

Object fields: the list of data fields to be included in the report. Depending on the implementation of the system, this data type will vary. Until that is decided, we will consider this information as a general object.

Date startDate: a Date object that specifies a subset of data in the Database that is to be included in the report.

Date endDate: a Date object that specifies a subset of data in the Database that is to be included in the report.

**public boolean saveReport(String adminUserName, String adminPassword, Object report)**

*Description:*

Saves report to a local location on the Admin’s computer.

*Parameters:*

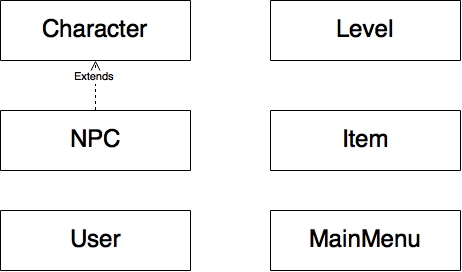
String adminUserName: the username of the Admin account attempting to make the changes.

String adminPassword: the password of the Admin account attempting to make the changes.

Object report: the report to be saved on the user’s computer.

**GAME MODULE DESIGN**

Below, Figure 8 provides a mid-level design of the Game component to be implemented into the system.

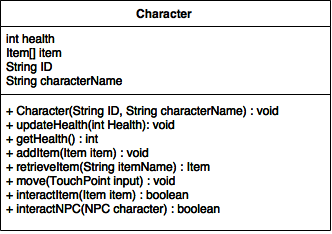


*Figure 8: High-Level Design of the Game Component*

The Game component will consist of all functionality relating to user interactions with the system. One of the main functional requirements of this system is that it includes a user-friendly graphical user interface in the form of a game that teaches the user fundamental concepts of computer programming. We feel that it makes the most sense to develop the game as a separate component that is only able to communicate with the Database component through the Communications component’s well-defined provided interfaces. The main motivation for this design tactic is to prevent the user from directly accessing the database (which contains information concerning *all* of the accounts affiliated with the system), thus adding an extra layer of security to the system.

On the following pages, you will find a low-level design of the Game component of this system.

**Character**



*Figure 9: a UML Class Diagram of the Character class*

Purpose:

The purpose of this module is to implement characters into the Game component. The Game will contain many types of characters, all of which will utilize the functionalities provided by this module. This class addresses the non-functional requirement specifying that the user shall be able to interact with the system in a fun, entertaining manner, which will be accomplished by playing the game from the perspective of a character that the user chooses when they first set up their user account.

Provides Interface:

**public Character(String ID, String characterName)**

*Description:*

This will create the character object. The character will then be used to navigate through the game.

*Parameters:*

String ID: a unique identifier for the character object.

String characterName: the name of the character.

*Returns:*

N/A (constructor).

**public void updateHealth(int health)**

*Description:*

Updates the character’s health when an event occurs that requires a change in the character’s health.

*Parameters:*

Int health: the amount of change in the character’s health.

*Returns:*

N/A.

**public int getHealth()**

*Description:*

Returns the current level of health of the character to the game.

*Parameters:*

N/A.

*Returns:*

An integer value representing the character’s health.

**public void addItem(Item item)**

*Description:*

Adds an Item object to the Item array attribute of the Character object.

*Parameters:*

Item item: the Item object to be added to the Character’s array of Items.

*Returns:*

N/A.

**public Item retrieveItem(String itemName)**

*Description:*

Finds the item itemName in the Character’s Item array attribute and returns it to the game for use.

*Parameters:*

String itemName: the item to be searched for in the Character’s array of Items.

*Returns:*

An Item object that can be used by the Character in the game.

**public void move(TouchPoint input)**

*Description:*

Determines the direction in which to move the character and performs the appropriate actions.

*Parameters:*

TouchPoint input: the input that, depending on the coordinates of the touch point on the screen, determines which direction to move the Character within the game.

*Returns:*

N/A.

**public boolean interactItem(Item item)**

*Description:*

Determines what happens when a Character interacts with an Item and returns the interaction type (good or bad) to the calling method.

*Parameters:*

Item item: the Item that the Character interacts with.

*Returns:*

A boolean value that determines the type of interaction with the Item. The method will return true if the interaction is “good” or false if the interaction is “bad.”

**public boolean interactNPC(NPC character)**

*Description:*

Determines what happens when the Character interacts with an NPC within the Game and returns the interaction type (good or bad) to the calling method.

*Parameters:*

NPC character: the NPC that the Character interacts with.

*Return:*

A boolean value that determines the type of interaction with the Item. The method will return true if the interaction is “good” or false if the interaction is “bad.”

Requires Interface:

**public Item selectItem()**

*Description:*

Allows the Character to select a particular Item.

*Parameters:*

N/A.

**public User(String id, Character userChar, String userName)**

*Description:*

Creates a User object to be used in the game.

*Parameters:*

String id: a unique identifier to identify the User object.

Character userChar: the Character object to be used within the game by the user.

String userName: the username of the specific user.

**public void moveCharacter()**

*Description:*

Performs the actions required to move a Character object in a particular direction.

*Parameters:*

N/A.

**public void readTouchPoint(TouchPoint input)**

*Description:*

Gets the coordinates of a touch point on the screen and calls the appropriate method to perform the action related to the touch point on the screen. Specific ranges of coordinates will contain buttons relating to separate actions (such as displaying a hint or selecting a level).

*Parameters:*

TouchPoint input: the positional coordinates on the screen that the user selects.

**public boolean isBad()**

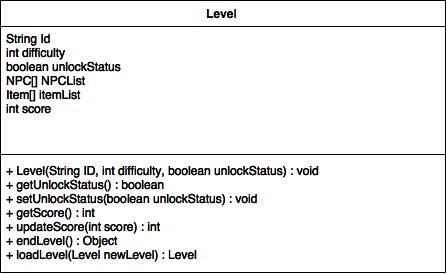
*Description:*

Returns the NPC’s isBad characteristic.

*Parameters:*

N/A.

**Level**



*Figure 10: a UML Class Diagram of the Level class*

Purpose:

The purpose of this module is to implement multiple levels into the Game component. The Game will contain many different levels, all of which will utilize the functionalities provided by this module. The user will interact with the levels using a Character that they chose when they first set up their account. This class addresses the non-functional requirement specifying that the user shall be able to interact with the system in a fun, entertaining manner, which will occur by guiding a Character through well-designed levels with objects that the character can interact with, such as Items and NPCs.

Provides Interface:

**public Level(String id, int difficulty, boolean unlockStatus)**

*Description:*

Creates a Level object. Each level will contain a specific level identifier, difficulty, unlocked status, and an array of NPC’s and Items that will be found in the specific level.

*Parameters:*

String ID: a specific identifier for the level.

int difficulty: the difficulty of the level.

boolean unlockStatus: value that represents whether or not the level is unlocked.

*Returns:*

N/A (constructor).

**public boolean getUnlockStatus()**

*Description:*

Returns the unlockStatus value of the Level to the game. This determines whether or not the level is unlocked.

*Parameters:*

N/A.

*Returns:*

A boolean value. Return true if the level is unlocked or false if the level is locked.

**public void setUnlockStatus(boolean unlockStatus)**

*Description:*

Sets whether or not the player can play this level (unlocked versus locked).

*Parameters:*

boolean unlockStatus: the value that represents whether or not the level is unlocked.

*Returns:*

N/A.

**public int getScore()**

*Description:*

Returns the user’s score to the calling method.

*Parameters:*

N/A.

*Returns:*

An int value that is the player’s score.

**public int updateScore(int score)**

*Description:*

Updates the player’s score for that particular level

*Parameters:*

int score: the new score that the user has earned.

*Returns:*

An int value that is the player’s updated score.

**public Object endLevel()**

*Description:*

Executes the code needed to end the level, return the new data to be saved to the user’s save file (if they are not playing as a guest), and returns to the main menu.

*Parameters:*

N/A.

*Returns:*

An Object containing the new data to be added to the user’s save file (if they are not playing as a guest).

**public Level loadLevel(Level newLevel)**

*Description:*

Loads the level that is passed to the method.

*Parameters:*

Level newLevel: a Level object that is the level to be loaded.

*Returns:*

A Level object that the User wishes to load.

Requires Interface:

**public User(String id, Character userChar, String userName)**

*Description:*

Creates a User object to be used in the game.

*Parameters:*

String id: a unique identifier of the User object.

Character userChar: the Character object to be used by the User within the game.

String userName: the username of the specific user.

**public void pause()**

*Description:*

Executes the code required to pause the game.

*Parameters:*

N/A.

**public void displayHint()**

*Description:*

Displays the hint for the active Level.

*Parameters:*

N/A.

**public void readTouchPoint(TouchPoint input)**

*Description:*

Gets the coordinates of the touch point on the screen and calls the appropriate method depending on the coordinates. Specific ranges of coordinates will contain buttons relating to separate actions (such as displaying a hint).

*Parameters:*

TouchPoint input: the positional coordinates that the user selects.

**public Character(String ID, String characterName)**

*Description:*

This will create the character object. The character will then be used to navigate through the game.

*Parameters:*

String ID: a unique identifier for the character object.

String characterName: the name of the character.

**public int getHealth()**

*Description:*

Returns the integer value that represents the health of the Character to the Game.

*Parameters:*

N/A.

**public void addItem(Item item)**

*Description:*

Adds an Item object to the Item array attribute of a Character object.

*Parameters:*

Item item: the Item object to be added to the Character’s Items array attribute.

**public void move(TouchPoint input)**

*Description:*

Determines the direction in which to move the character and performs the appropriate actions.

*Parameters:*

TouchPoint input: the input that, depending on the coordinates of the touch point on the screen, determines which direction to move the Character within the game.

**public Item(String itemID)**

*Description:*

Creates an Item for the specific level. Each item is given a unique identifier.

*Parameters:*

String itemID: a unique identifier that is assigned to each item.

**public void setPosition(String position)**

*Description:*

Sets the position of the item within the level

*Parameters:*

String position: the positional coordinates of the Item.

**public String getPosition()**

*Description:*

Gets the position of the item within the level.

*Parameters:*

N/A.

**public NPC(String NPCname)**

*Description:*

Creates a NPC character. Each character is assigned a unique name.

*Parameters:*

String NPCname: a unique name for the NPC.

**public String displayText()**

*Description:*

Displays text from the NPC to the User’s Character.

*Parameters:*

N/A.

**public File save(String[] updatedData)**

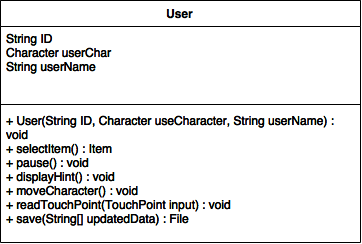
*Description:*

If a RegisteredUser is logged in, updates the save file associated with the account.

*Parameters:*

String[] updateData: an array of Strings containing the updated data for the new save file.

**User**



*Figure 11: a UML Class Diagram of the User class*

Purpose:

The purpose of this module is to implement the User into the Game component. The Game will contain many different functionalities for the User, all of which will utilize the functionalities provided by this module. The user will interact with the user interface with a character that they chose when they first set up their account. This class addresses the non-functional requirement that the game should provide a fun, entertaining way of teaching programming concepts to the user, which will be achieved through eye-catching and easy-to-navigate level designs.

Provides Interface:

**public User(String id, Character userChar, String userName)**

*Description:*

Creates a user object to be used in the game.

*Parameters:*

String id: the unique user identification value.

Character userChar: the Character object to be used by the User in the Level.

String userName: the username of the specific user.

*Returns:*

N/A (constructor).

**public Item selectItem()**

*Description:*

Allows the User to select a particular Item.

*Parameters:*

N/A.

*Returns:*

The Item that the User selects.

**public void pause()**

*Description:*

Performs the necessary actions to pause the game.

*Parameters:*

N/A.

*Returns:*

N/A.

**public void displayHint()**

*Description:*

Displays the hint for the active level.

*Parameters:*

N/A.

*Returns:*

N/A.

**public void moveCharacter(TouchPoint input)**

*Description:*

Determines the direction in which the user wishes to move the Character.

*Parameters:*

TouchPoint input: the coordinates of a touch input on the screen which, depending on the coordinates, initiates code for moving the character in a particular direction.

*Returns:*

N/A.

**public void readTouchPoint(TouchPoint input)**

*Description:*

Gets the coordinates of the touch point on the screen and calls the appropriate method depending on the coordinates. Specific ranges of coordinates will contain buttons relating to separate actions (such as displaying a hint).

*Parameters:*

TouchPoint input: the positional coordinates that the user selects.

*Returns:*

N/A.

**public File save(String[] updatedData)**

*Description*:

If a RegisteredUser is logged in, updates the save file.

*Parameters:*

String[] updatedData: an array of Strings containing the data for the new save file.

Requires Interface:

**public mainMenu()**

*Description:*

Creates a mainMenu object.

*Parameters:*

N/A.

**public Level selectLevel (Level selectedLevel)**

*Description:*

Allows the user to select the level they want to play.

*Parameters:*

Level selectedLevel: the level that the user wishes to load.

**public boolean logOut()**

*Description:*

Initiates the code necessary to log the user out of the game.

*Parameters:*

N/A.

**public void readTouchPoint(TouchPoint input)**

*Description:*

Gets the coordinates of the touch point on the screen and calls the appropriate method depending on the coordinates. Specific ranges of coordinates will contain buttons relating to separate actions (such as selecting a level).

*Parameters:*

TouchPoint input: the positional coordinates that the user selects.

**public Level loadLevel(Level level)**

*Description:*

Loads the level that is passed to the method.

*Parameters:*

Level level: the level to be loaded.

**public int getScore()**

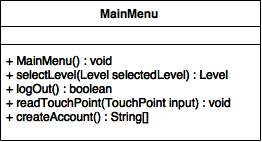
*Description:*

Returns the user’s score to the calling method.

*Parameters:*

int score: the user’s score.

**MainMenu**



*Figure 12: a UML class diagram of the MainMenu Class*

Purpose:

The purpose of this module is to provide the user with the option to select their desired difficulty and, within that difficulty, their desired level. This class addresses the functional requirement stating that the user shall be able to select a difficulty level and a specific topic within that difficulty level.

Provides Interface:

**public mainMenu()**

*Description:*

Creates a MainMenu object.

*Parameters:*

N/A.

*Return:*

N/A (constructor).

**public Level selectLevel (Level selectedLevel)**

*Description:*

Allows the user to select the level that they wish to play.

*Parameters:*

Level selectedLevel: the level that the User selects.

*Returns:*

A Level object that the User wishes to load.

**public boolean logOut()**

*Description:*

Executes the necessary code to log the User out of the Game.

*Parameters:*

N/A.

*Returns:*

Returns true if the user is successfully logged out and false otherwise.

**public void readTouchPoint(TouchPoint input)**

*Description:*

Gets the coordinates of the touch point on the screen and calls the appropriate method depending on the coordinates. Specific ranges of coordinates will contain buttons relating to separate actions (such as selecting a level).

*Parameters:*

TouchPoint input: the positional coordinates that the user selects.

*Returns:*

N/A.

**public String[] createAccount()**

*Description:*

Allows the player to create an account.

*Parameters:*

N/A.

*Returns:*

An array of Strings that is needed to create a new user account.

Requires Interface:

**public User(String firstName, String lastName, String userName, String password, String emailAddress, String userType)**

*Description:*

Retrieves data from the Game component that is required to create a new User and add it to the Database. Depending on the user type, which is part of the parameters, a subclass of the User class may be called.

*Parameters:*

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

String userType: the type of user (student, instructor, or general).

**public File logIn(String userName, String password)**

*Description:*

Checks if the credentials entered by the user are valid in order to access their account. If so, loads the save file associated with the specified account.

*Parameters:*

String userName: the username of the account.

String password: the password for the account.

**public boolean logOut(String userName)**

*Description:*

Initiates the logout sequence for the account.

*Parameters:*

String userName: the username associated with the account that is logged in.

**public Object generateClassReport(String courseCode)**

*Description:*

Retrieves the progress from the Database of all Student accounts with the courseCode attribute passed to this method.

*Parameters:*

String courseCode: the course code associated with the course that the Instructor wishes to generate a report on.

**public boolean getUnlockStatus()**

*Description:*

Checks to see if a level is unlocked.

*Parameters:*

N/A.

**public Level loadLevel(Level level)**

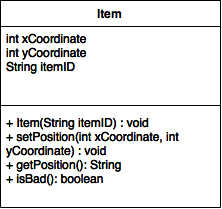
*Description:*

Loads the level that is passed to the method.

*Parameters:*

Level level: the level to be loaded.

**Item**



*Figure 13: a UML class diagram of the Item Class*

Purpose:

The purpose of this module is to create items that the character can use in the game. These items will be different from level to level but items will be retained in a character’s *backpack* (the array of Items included in each Character object)*.* These items will be used to assist the character in completing the various levels. The use of Items addresses the non-functional requirement that the system must keep the user’s interest for long periods of time. Through the use of Items, the Character will be able to perform additional tasks that make the game more fun to play.

Provides Interface:

**public Item(String itemID)**

*Description:*

Creates a new Item. Each item is given a unique identifier.

*Parameters:*

String itemID: a unique identifier assigned to the Item object.

*Returns:*

N/A (constructor).

**public void setPosition(int xCoordinate, int yCoordinate)**

*Description:*

Sets the position of the Item within the level.

*Parameters:*

int xCoordinate: the x-coordinate of the Item within the Level.

int yCoordinate: the y-coordinate of the Item within the Level.

*Returns:*

N/A.

**public String getPosition()**

*Description:*

Returns the position of the Item (as a String) within the level.

*Parameters:*

N/A.

*Returns:*

A String representing the coordinates of the Item.

**public boolean isBad()**

*Description:*

Returns the value of the isBad attribute the Item.

*Parameters:*

N/A.

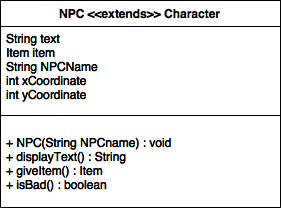
*Returns:*

A boolean value of true if the Item is “good” and a value of false if the Item is “bad.”

Requires Interface:

N/A.

**NPC**



*Figure 14: a UML class diagram of the NPC Class*

Purpose:

The purpose of this module is to create NPCs (non-playable characters) that can interact with the character. These NPCs will be able to interact with the player. This is a subclass of the Character class. This class addresses the non-functional requirement that the the Game keeps the user interested for long periods of time. Through interactions with NPCs, the User can choose to complete various tasks which will teach them the programming concept that is the focus of the active Level.

Provides Interface:

**public NPC(String NPCname)**

*Description:*

Creates an NPC object to be inserted into a Level.

*Parameters:*

String NPCname: a unique name for the NPC object.

*Returns:*

N/A (constructor).

**public String displayText()**

*Description:*

Displays text associated with the NPC object to be displayed to the Character associated with the User.

*Parameters:*

N/A.

*Returns:*

A String to be displayed on the screen.

**public Item giveItem()**

*Description:*

Initiates the code needed to give the Character the Item associated with the NPC object.

*Parameters:*

N/A

*Returns:*

An Item object that is given to the user when the user interacts with the NPC.

**public boolean isBad()**

*Description:*

Determines whether or not NPC’s isBad characteristic is true or false.

*Parameters:*

N/A.

*Returns:*

A boolean value of true if the NPC is “good” and a value of false if the NPC is “bad.”

Requires Interface:

N/A

**DATABASE DESIGN**

Description:

The purpose of the database is to simultaneously allow multiple instances of the Game and Communications components to create, read, update, delete, and execute data.

Provides Interface:

**public boolean addUser(String adminUserName, String adminPassword, String newUserFirstName, String newUserLastName, String newUserUserName, String newUserPassword, String newUserEmailAddress, String newUserUserType)**

*Description:*

Allows the Admin user to override the automated system processes of adding a user to the Database and manually add a user to the Database.

*Parameters:*

String adminUserName: the username of the Admin user creating the new account. (This will be validated before creating the new account.)

String adminPassword: the password associated with the Admin user. (This will be validated before creating the new account.)

String newUserFirstName: the first name of the new user.

String newUserLastName: the last name of the new user.

String newUserUserName: the desired user name to be associated with the new account.

String newUserPassword: the password for the new user account.

String newUserEmailAddress: the email address to be affiliated with the new user account.

String newUserUserType: the type of the new user account.

*Returns:*

A boolean value. If the new user account was successfully created, returns true. Otherwise, returns false.

**public boolean editUser(String adminUserName, String adminPassword, String firstName, String lastName, String userName, String password, String emailAddress)**

*Description:*

Allows the Admin user to manually update information regarding an account stored in the database. The Admin does *not* have the ability to edit data relating to the game itself, such as saved configurations and saved game progress.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String firstName: the first name of the user.

String lastName: the last name of the user.

String userName: the desired user name to be associated with the account.

String password: the password to access the account.

String emailAddress: the email address associated with the account.

*Returns:*

A boolean value. If the user account was successfully updated, returns true. Otherwise, returns false.

**public boolean resetUserPassword(String adminUserName, String adminPassword, String userUserName, userEmailAddress)**

*Description:*

Allows the Admin user to manually initiate the process that resets the password affiliated with an user account.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String userUserName: the username affiliated with the account whose password shall be reset.

String userEmailAddress: the email address affiliated with the account whose password shall be reset.

*Returns:*

A boolean value. If the password reset process completes successfully, returns true. Otherwise, returns false.

**public boolean removeUser(String adminUserName, String adminPassword, String userUserName, String userEmailAddress)**

*Description:*

Allows an Admin to manually remove an account in the database.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String userUserName: the username associated with the account to be removed.

String userEmailAddress: the email address associated with the account to be removed.

*Returns:*

A boolean value. If the account is successfully removed from the database and an email was sent to the user informing them of this change, return true. Otherwise, returns false.

**public String generateClassroomID(String adminUserName, String adminPassword, String instructorUserName)**

*Description:*

Allows the Admin user to generate a new classroom identifier to be associated with an instructor’s account.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String instructorUserName: the username of the instructor account that will be given a new classroom identifier.

*Returns:*

A string containing the generated classroom ID.

**public Object generateReport(String adminUserName, String adminPassword, Object fields, Date startDate, Date endDate)**

*Description:*

Allows the Admin user to generate a report of information from the Database.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

Object fields: the list of data fields to be included in the report. Depending on the implementation of the system, this data type will vary. Until that is decided, we will consider this information as a general object.

Date startDate: a Date object that specifies a subset of data in the Database that is to be included in the report.

Date endDate: a Date object that specifies a subset of data in the Database that is to be included in the report.

*Returns:*

An object containing the generated report.

**public boolean saveReport(String adminUserName, String adminPassword, Object report)**

*Description:*

Saves report to a local location on the admins computer.

*Parameters:*

String adminUserName: the username of the Admin account attempting to make the changes (used to ensure database access is not malicious).

String adminPassword: the password of the Admin account attempting to make the changes (used to ensure database access is not malicious).

Object report: the report to be saved on the user’s computer.

*Returns:*

A boolean value. If the file is successfully saved, returns true. Otherwise, returns false.

**public boolean checkUsernameAvailability(String userName)**

*Description:*

Scans the database to check if the username that the user wishes to associate with their account has already been taken.

*Parameters:*

String userName: the desired username.

*Returns:*

A boolean value - true if the username is available, false otherwise.

Requires Interface:

N/A (only provides to the rest of the system).

**ABSTRACT DATA TYPES**

Character {

int health

Item[] item

String ID

String characterName

}

Level {

String id

int difficulty

boolean unlockStatus

NPC[] NPCList

Item[] itemList

int score

}

User {

String id

Character userChar

String userName

}

Main Menu {

}

Item {

int xCoordinate

int yCoordinate

String itemID

}

NPC {

int xCoordinate

int yCoordinate

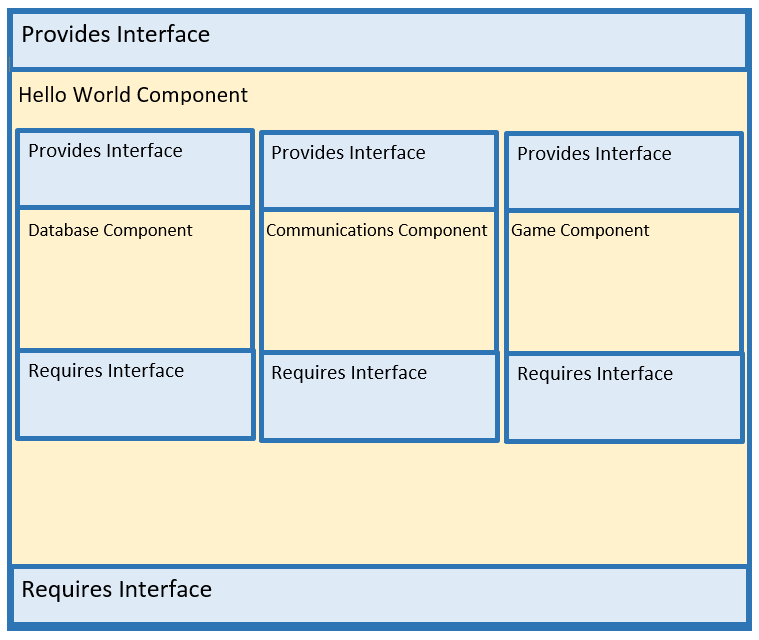
String text

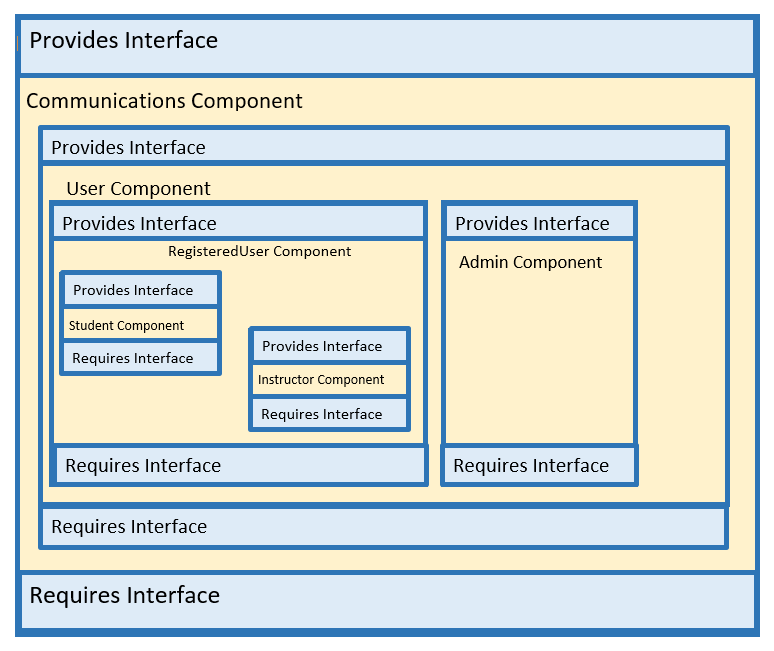
Item item

String NPCName

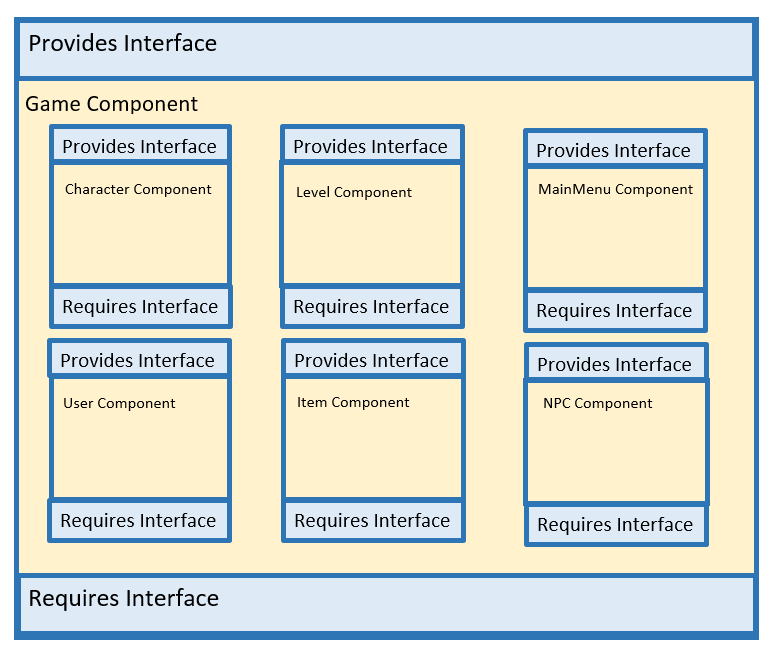
}

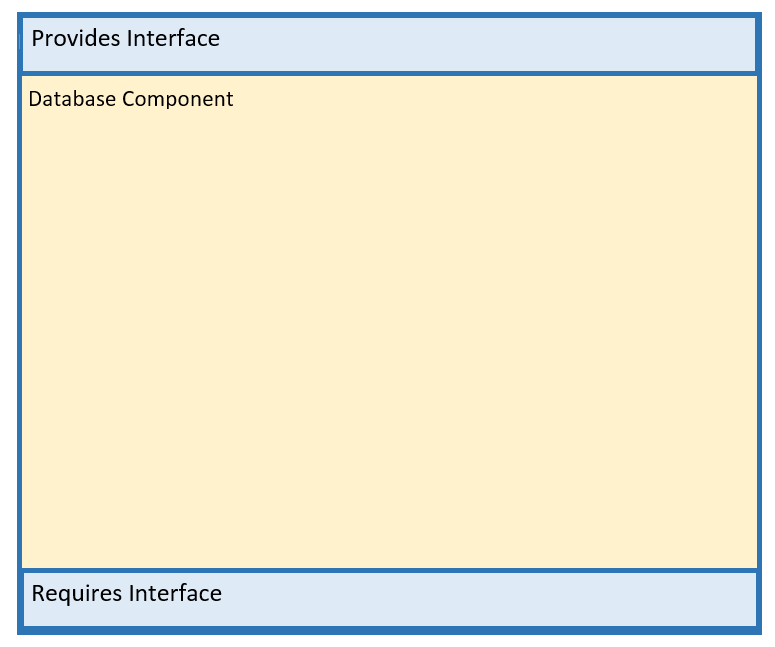
**COMPRISES DIAGRAMS**





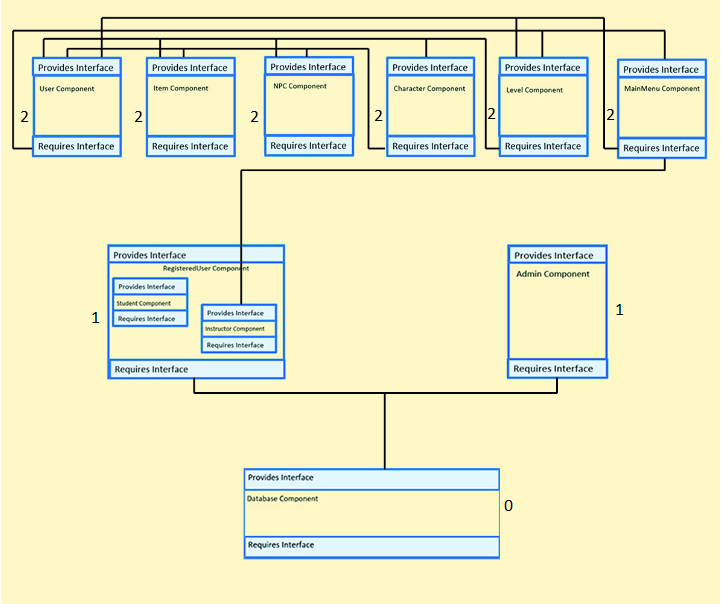
*Figure 15: Comprises Diagram of the Communications Component of the System*

*Figure 16: Comprises Diagram of the Game Component of the System*



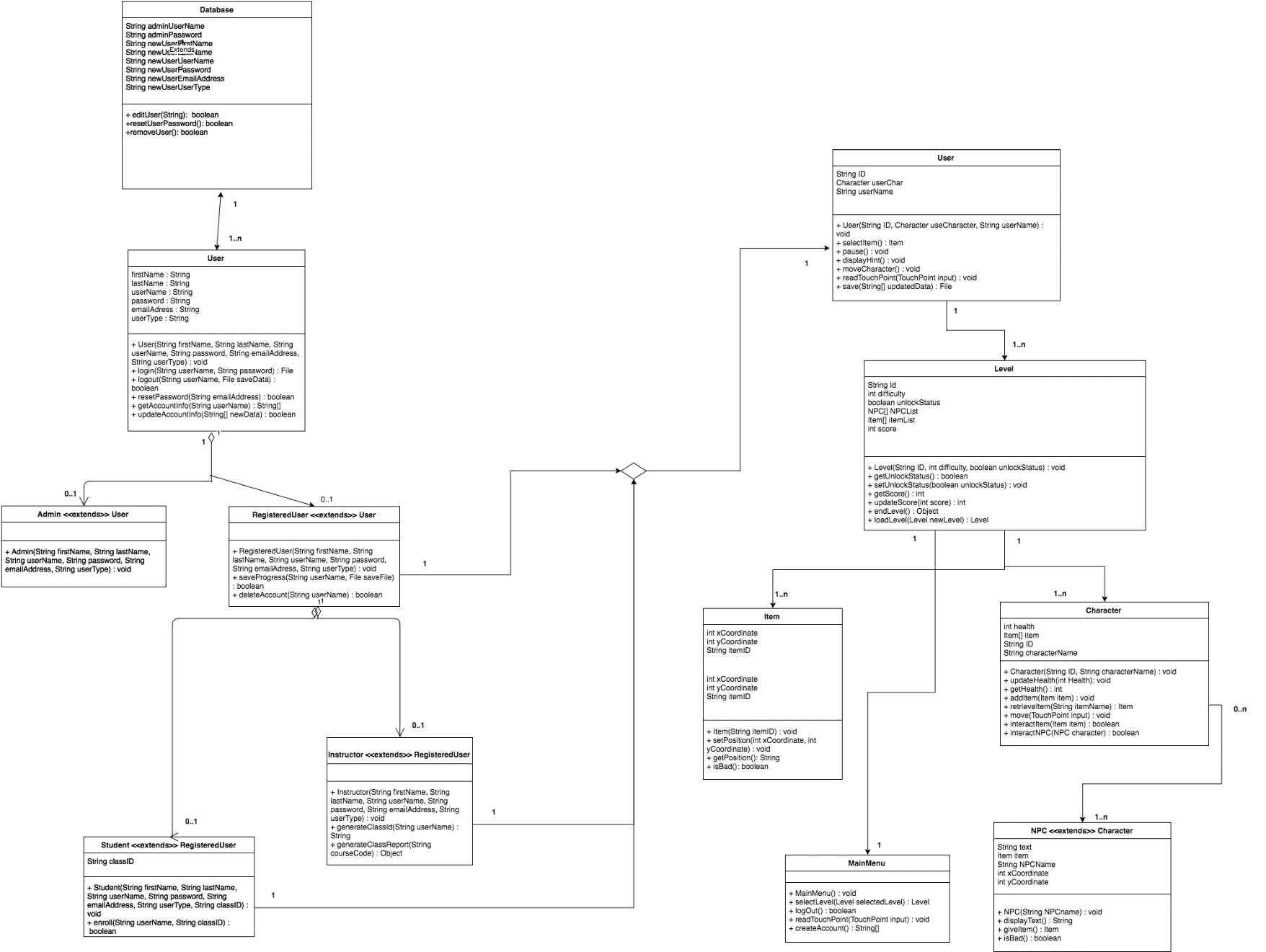
*Figure 17: Comprises Diagram of the Database Component of the System*

**USES DIAGRAM**



*Figure 18:USES Diagram of the System*

**CLASS DIAGRAM**



**INTEGRATION TEST PLAN**

**Approach**

For the integration and testing of *Hello World: An Introduction to Computer Programming Concepts* we are a bottom-up approach. The bottom-up approach allows us to use drivers to test the low level components. When we do the final test, we know that every other sub component will successfully run.

**Order of Integration**

In the table below are listed the various phases that will be tested to ensure proper integration. We will begin with the database module, move to the communication module, and finish with the game module.

|  |  |  |
| --- | --- | --- |
| **Integration Group** | **Driver(s) Needed** | **Stub(s) Needed** |
|  |  |  |
| **Phase 1: Database Module** | | |
| N/A | N/A | N/A |
| **Phase 2: Communication Module** | | |
| Admin Module | User Driver | None |
| Registered User Module | User Driver | None |
| Student Module | User Driver | None |
| Instructor Module | User Driver | None |
|  |  |  |
| **Phase 3: Game Module** | | |
| Character Module | Level Driver | None |
| Level Module | Level Driver | None |
| User Module | User Driver | None |
| MainMenu Module | MainMenu Driver | None |
| Item Module | Level Driver | None |
| NPC Module | Level Driver | None |

**Driver Definitions**

User Driver: simulates the creation of the user accounts

Level Driver: simulates the level environment

MainMenu Driver: simulates a menu driver

**Tests Cases**

**Phase 1: Database Module**

**Phase 2: Communications Module**

**ID:** Admin

**What is being tested:** The admins ability to access and modify the database.

**Input Data:** Various attributes of a user account.

**Expected Output:** Changes are successfully made to the database.

**Environmental Constraints:** There is a connection to the database.

**Testing Constraints:** If adding, the user with that username must not exist already. If editing, the user must already exist.

**ID:** RegisteredUser

**What is being tested:** Create or edit a registered user account.

**Input Data:** Various profile information.

**Expected Output:** A created or update user account.

**Environmental Constraints:** There is a connection to the database.

**Testing Constraints:** If adding, the user with that username must not exist already. If editing, the user must already exist.

**ID:** Student

**What is being tested:** Create or edit a student user account and the ability to enroll in the class.

**Input Data:** Various profile information.

**Expected Output:** A created or updated student account.

**Environmental Constraints:** There is a connection to the database.

**Testing Constraints:** If adding, the user with that username must not exist already. If editing, the user must already exist.

**ID:** Instructor

**What is being tested:** The ability to create or edit an instructor account and to create a class. Also the ability to generate a report of a class.

**Input Data:** Various profile information.

**Expected Output:** A created or updated instructor account.

**Environmental Constraints:** There is a connection to the database.

**Testing Constraints:** If adding, the user with that username must exist already. If editing, the user must already exist. If generating a report, the class must exist already.

**Phase 3: Game Module**

**ID:** Character

**What is being tested:** The ability to create and modify the state of character.

**Input Data:** The name and ID of a new character.

**Expected Output:** A new character is made and can be modified.

**Environmental Constraints:** N/A.

**Testing Constraints:** N/A.

**ID:** Level

**What is being tested:** The creation for a level and that it can be modified.

**Input Data:** Various information regarding level characteristics.

**Expected Output:** The instantiation of a level.

**Environmental Constraints:** N/A.

**Testing Constraints:** N/A.

**ID:** User

**What is being tested:** The user is able to interact with the game.

**Input Data:** Input from the device.

**Expected Output:** Provided input is successfully executed.

**Environmental Constraints:** N/A.

**Testing Constraints:** N/A.

**ID:** MainMenu

**What is being tested:** The creation and successful interaction with a menu

**Input Data:** Input from the device.

**Expected Output:** Correct actions depending on which features for the menu are used.

**Environmental Constraints:** N/A.

**Testing Constraints:** N/A.

**ID:** Item

**What is being tested:** The item is created, can be used, and modified within a level.

**Input Data:** Information that identifies and modifies the item.

**Expected Output:** A created, usable, and modifiable item.

**Environmental Constraints:** You are within a level.

**Testing Constraints:** N/A.

**ID:** NPC

**What is being tested:** The creation of an intractable NPC.

**Input Data:** Information needed to create and identify a NPC.

**Expected Output:** A NPC that the user can interact with.

**Environmental Constraints:** You are within a level.

**Testing Constraints:** N/A.