7-2006

Advancements in frameworks for educational games through sound software engineering principles.

Christy M. Bogard
University of Louisville

Recommended Citation
Bogard, Christy M., "Advancements in frameworks for educational games through sound software engineering principles." (2006). *Electronic Theses and Dissertations.* Paper 121.
https://doi.org/10.18297/etd/121

ADVANCEMENTS IN FRAMEWORKS FOR EDUCATIONAL GAMES THROUGH SOUND SOFTWARE ENGINEERING PRINCIPLES

By

Christy M. Bogard
B.S., University of Louisville, 2004

A Thesis
Submitted to the Faculty of the
University of Louisville
J. B. Speed School of Engineering
in Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Computer Engineering and Computer Science

July 2006

ADVANCEMENTS IN EDUCATIONAL GAMES
THROUGH SOUND SOFTWARE ENGINEERING PRINCIPLES

Submitted by: ______________________________
Christy M. Bogard

A Thesis Approved on

____
(Date)

by the Following Reading and Examination Committee:

____
Dr. Rammohan K. Ragade, Thesis Director

____
Dr. Ibrahim N. Imam

____
Dr. Julius P. Wong

DEDICATION

For Mom and Jim,
You told me I could accomplish anything.
And I believed you.

For Stephanie and Leah,
You've shown me that life is a series of humorous moments,
and the only way to survive is to laugh at it all.

For Jason,
Your inquisitive mind and constant encouragement
have helped me accomplish the impossible.

For My Teachers,
You pushed me to greatness,
then showed me how to go even further.

ABSTRACT

Educational games have steadily entered classrooms as a means of challenging advanced students and tutoring those lacking comprehension. However, without adequate educational benefits, instructors are struggling to continually justify the marginal value added of using these programs. It is the intent of this thesis to demonstrate that sound software engineering principles can improve the framework of educational games. First, the core framework requirements of computer-based educational games are outlined. Current educational games are then evaluated based on their ability to meet these requirements. From this analysis, necessary architectural changes are recommended to best facilitate future game advancements. Finally, to demonstrate the viability of the changes, a functional, elementary level educational game is developed based on the recommended modular architecture with low coupling and high cohesion.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Students have a higher retention rate of subject material when it is reinforced by additional sources outside of classroom instruction. Instructional simulations and educational games have the potential to provide this additional reinforcement. However, educational software, as it exists today, has fragmented content. When designed, a small to mid-size cluster of programmers attempt to address a single concept or problem. Often, it will take several months to a year to adequately address the concept or problem within a single focalized application. The specialized nature of the developed application then makes it very difficult to be of value to a broad audience of students. Without an effective interface to combine a multitude of these specialized applications, instructors are overwhelmed attempting to justify the myriad of necessary applications required to reinforce course content. This limitation, combined with limited high quality educational products available, outweighs the benefits of using educational software within the classroom [3, 5, 8, 11, 15].

Given the myriad of specialized software developed to address single concepts or problems, it becomes essential for educational components to be developed modularly for educational games and simulations to continually advance. If a new component is developed based on the current architecture of a specialized application, it becomes virtually impossible to integrate the new component with any other currently available application. Currently, high coupling within these specialized applications prevents component modularity among different educational applications. As such, advancements in development, while valuable to the application in which it is focused, are virtually obsolete elsewhere.

While there is currently limited potential of interfacing these specialized applications due to their high coupling, components that are implemented as modular shells independent of current applications have increased reusability. This project develops an independent external interface to enhance needed educational components, such as incremental advancement of problem difficulty to continually challenge students at the appropriate level, record of student progress to enable instructors to analyze a student's strengths and weaknesses, and capability to tailor the computer game to an instructor's individual classroom specifications.

To incorporate these necessary components within an educational game, the application's interface controls the interaction between the student and the game by making the necessary calls to the game's modular game components. These components, in turn, make relevant calls to an underlying database, maintaining separation of component implementation and game content. Using an associated database to contain game questions eliminates the specialized nature of many current applications because the database can be easily modified to address more than one concept or problem.

II. LITERATURE REVIEW

2.1 Educational Game Growth and Literature

While the gaming industry has been growing at an unprecedented rate, expecting to grow by 71 percent to $85.7 billion by 2006, the educational software sector has dramatically lagged, representing only 6.5% of the computer and video game dollar sales. As such, published literature on educational computer games has only begun gaining substantial volume since 2000 [7, 13, 15, 16, 17, 18, 25].

However, even given the substantial growth in literature, leading researchers are divided on how useful computer and video games are. Those in favor of these games claim they can further develop social and cognitive skills, increase in the retention of information, and keep students engaged and motivated in learning. Those against these games claim they can increase youthful aggression, result in social isolation, and because of their addictive nature, cause weight and health complications [2, 4, 9, 10].

Given the lack of consensus on the usefulness of computer and video games within the classroom, the majority of published literature on educational software has focused on the following four categories:

The first category of articles contains general overviews of computer and video games coming to the market. These articles focus on what new games have been developed and how they meet a specific need. In most cases, because the primary intention of the authors is to sell the given product, only a biased evaluation is presented, giving a skewed representation of educational value contained within.

The second category of articles is focused on pre- and post- testing of specific educational software within a controlled environment. Because they resemble short-term, limited case studies often on a single narrow topic, conclusions reached are often difficult to reproduce and even more difficult to generalize in order to make the results beneficial outside their narrow scope.

The third category of research is focused on the effects of gaming on individuals. This is the largest and most controversial area of research, examining the physiological, cognitive and social effects of playing games on users. These articles are often co-authored by psychologists, focusing more on the benefits or consequences of educational games, not on the educational games themselves.

The fourth and final category is focused on research reviews and meta-analyses. This sector of articles is often authored by educators and aimed at critiquing the components of educational software. Game developers are most interested in this area of research because it provides a glimpse into user specifications for educational software. However, deciphering key specifications is often more difficult, as educators are primarily focused on what hinders usefulness in the classroom and not what is necessary to make the games beneficial [15, 16, 18, 21].

2.2 Current Educational Game Types Available

Before one can determine clear client requirements, it is critical to have an understanding of the games currently available. Educational games can be divided into five general categories: Drill and Practice Games, Half and Half Games, Discovery Games, Content Games, and Non-Traditional Games. These five games range from a primary focus on educational content to a primary focus on entertainment, respectively [12, 20, 23, 24].

Drill and Practice Games, the first type of game, place focus on continually presenting similar problems centered on a single concept. The student practices over and over until he or she can successfully complete a predetermined number of problems. At such time, the student is rewarded, usually with a miniature activity game or animation. Because students are often relentlessly drilled on the concepts within the classroom, this type of game is typically only entertaining to, and thus effective for, elementary-age youth. Often, these games are presented in "Jeopardy"-like atmospheres, where players create a simple virtual character that gains points or money when he or she correctly answers the posed question and loses points or fined a set amount of money when he or she incorrectly answers the posed question.

Half and Half Games are the second type of educational games. These games, considered the foundation of edutainment, present educational content within an entertaining game environment. Players are rewarded by increasingly more difficult scenarios as they successfully complete the previously presented challenge. Because the game environment is highly interwoven with the educational content presented, the scope of the educational game is often very narrow, making the game too specific to be valuable to a broad audience. One such example is Oregon Trail. Oregon Trail defines survival problems for the game player as they move across the western plains. While players are presented with some differing scenarios as they progress, these scenarios are limited in variety to ensure completion of key educational modules.

Discovery Games expand the Half and Half games by shifting the focus even further to the entertaining aspect of the game. This is achieved by introducing an exploration aspect to the game. Students are encouraged to seek out the solution to the challenge presented through a less structured game environment. Given the increased time required to complete a challenge or reach a suitable stopping point, these types of games are often unsuited to the classroom because most students are forced to leave the task unfinished, an undesirable state. “Where in the World is Carmen Sandiego?” is one such Discovery Game. Students must move throughout the world in search of clues as to where Carmen has fled to with a precious artifact. While the student has vastly more control over his or her interactions, the game provides a myriad of clues to assist a lost player.

Content games expand upon the Discovery Games by shifting the focus primarily to the entertaining environment aspect, making the educational content presented the secondary focus. These games introduce an increased risk aspect and further exploration aspects by reducing the structured rules of the Discovery Games. However, the reduced structure combined with the shift in focus away from the educational content make these games extremely difficult to use within the classroom setting because success is often based on lucky or random discovery of key pieces of knowledge. An example of a Content Game is the “Riddle of the Sphinx.” A player is released into the desert to discover the ancient Egyptian world with only limited instructions. While the student can ultimately complete the objective at hand, it is often difficult to accurately measure one’s accomplishments due to lack of guidance.

Finally, Non-Traditional Games are the fifth type of educational game. These games have some clear educational value presented to the student, but weren't originally developed for educational purposes. As such, these games do not easily classify into the four traditional game types aforementioned [20, 23, 24].

2.3 Designing an Entertaining Game

Understanding what the five types of games are lends itself to a discussion of what elements make a game entertaining. First, and most importantly, games must have an interactive environment. The player's decisions should drive the game's responses, making the interactive element the distinctive thread separating games from other artistic ventures such as movies, music, or paintings. Thus, entertaining games must present scenarios in which users choose between different options.

Decision-making brings the next element into focus. The game must appropriately respond to different selections made by the user. If the game doesn't produce different responses to alternative selections, then the game lacks true interactivity.

The game also needs an element of achievement, the third critical element. While achievement can take on different meanings with different game contexts, successfully completing progressive challenges indicates a natural advancement through the game in actively seeking the end challenge. With elements of achievement, there also needs to be varying degrees of failure. Not successfully completing a challenge should result in a setback in the journey to the conclusion. However, failures should not result in unconquerable game scenarios.

The next element needed to make a game entertaining is a clearly defined challenge. Additionally, the problem presented should be interesting and having a logical solution that can be reached by interacting with the game. The key is finding the appropriate problem level that is not too simplistic as to bore the user to move on to other games and not too complex as to frustrate the user to quit entirely.

An entertaining game must also be fully self-encapsulated, creating an environment in which the user becomes self-absorbed within the game world. This is often called suspension of disbelief, because the user is so engaged in the game that he or she is unaware of one's surroundings.

Finally, an entertaining game should have a personal experience for the user, meaning that while users will have similar experiences, there are specific aspects that appeal to each individual user. This is often subdivided into what the user perceives as fun, what the user learns from the experience, and what alternative reality the user supplements with the actual game environment.

These are only a few of the components important to creating an entertaining game, but they represent the basic building blocks of the game. It is also critical to recognize that games are created uniquely in their selected trade-offs in each of these basic elements [4, 5, 8, 10, 11, 15].

2.4 Determining Client Requirements

Understanding the types of educational games available and the critical components that make a game entertaining leads to the determination of what additional requirements the clients, or school educators in this context, are seeking. To aid game developers in determining the specific needs of educators, leading officials have begun evaluating aspects that are critical for a game to have educational value within the classroom. For example, Bringing Educational Creativity To All (BECTa) and Teachers Evaluating Educational Multimedia (TEEM), two leading research organizations in educational computer games, have both developed comprehensive lists of components that are required for games to contribute value within the classroom, but are currently not present.

First, it is critical for the educational game to record what the student completed during the gaming session. Educational games are valuable in the classroom if it can both increase a student's understanding of a concept and provide an analysis of the student's learning to the instructor. Current educational games on the market only record a student's level of mastery, often given as a percent success rate or subjective description of mastery such as "excellent" or "good." Because of the limited artificial intelligence within the educational games, instructors cannot determine the underlying concepts that a student does or does not understand based on a level of mastery.

In order to provide instructors with the needed information, the game play interface must record what the student was able to successfully accomplish and what the student failed to master. Because learning is a complex process that does not easily fit into precise categories, games should not attempt to determine the underlying misconception, but instead provide the most amount of information possible to the instructor through a record of the student's interactions with the game.

Secondly, educational games should be able to adapt to students with different skill levels. In order to continually challenge a student requires a custom-tailored program that reacts appropriately to his or her demonstrated skills. Advancing question difficulty only after a student has fully demonstrated mastery of the previous challenge level results in boredom when a student has gained mastery but has not yet completed the current level requirements and frustration when a student cannot achieve success at the next challenge level.

In addition to being able to adapt to students with different skills levels, educational games should provide similar, but not identical, repeated experiences. This is especially beneficial when all students do not interact with the game simultaneously, but instead play sequentially. This ensures the latter students are not simply reproducing memorized experiences relayed from the former students. Furthermore, similar but unique gaming experiences promote classroom discussion and enable students to comprehend the experiences of their peers without having exactly the same experience.

The fourth component required to make educational games beneficial within the classroom is providing suitable breaking points during the game play. Using an educational game within the classroom is often inhibited by time constraints and possible interruptions. Providing stopping points allows the student to complete a task while not feeling unsatisfied for having an uncompleted task. Additionally, providing completion points can often reduce unnecessary time repeating previous accomplishments to resume game play.

Another critical component currently lacking is appropriate management tools provided to instructors. Educational games currently on the market lack developed Instructor's manuals that include pertinent information on structure content and underlying game models. For example, game scenarios should mimic realistic expectations and physical properties of the real world, furthering psychological, social, and intellectual development in students. Providing this information allows instructors to analyze games for their educational value as well as their appropriateness to the instructor's classroom.

While most educational games provide limited instructions for the instructors, the educational game play may require elaborate written instructions to be understood by the user. When such instructions are required, the reading comprehension level should match the target audience age. It is essential game designers recognize that an educational game played within the classroom setting must be capable of functioning independently of instructor's involvement, as instructors are often engaged with students not currently engaged with the educational game.

Finally, educational games should foster an encouraging environment that motivates students to continue involvement with the game, such as through satisfaction, desire, anger, absorption, interest, excitement, enjoyment, and pride in achievement. Educational games that do not continually engage the student's interest are often dismissed as futile, quickly rendering any educational value added ineffective.

While BECTa and TEEM have differing opinions as to the priority of these components, both agree that without these components, the costs of using educational software within the classroom will continue to outweigh the benefits. Unfortunately, these components are often expensive to implement. Commercially, these investments are justified by the substantial return on investment through the mass sale of the produced game. For example, Electronic Arts, the leading producer of computer games, reported 2004 revenues at nearly three billion dollars. But educational software cannot produce these high revenues. As such, producers of educational games lack the necessary resources to produce a high quality product comparable to the currently available entertainment computer games. This is further illustrated by examining educational games currently on the market and what components they successfully incorporate. The table presented below outlines twelve of the most popular educational games currently available on the market. These games are compared to the critical components outlined by both BECTa and TEEM. As one can see, no game currently available meets even half of the listed requirements outlined [3, 5, 12, 13, 15, 19].

Key Components \ Educational Game	Finding Nemo: Learning with Nemo	JumpStart Reading	Nancy Drew: Curse of Blackmoor Manor	Charlie and the Chocolate Factory	Pre-Algebra Solved!	The Charles W. Morgan
Record of student progress						
Adaptable level of challenge						
Non-identically repeated experiences			√			
Ability to save and restart games		√	√	√		
Suitable stopping points throughout game play	√	√			√	
Instructor's manual including information on structure content and underlying game models						√
Game scenarios mimic realistic expectations and physical properties of the real world		√	√		√	√
User interface and instructions that do not require elaborate written instructions	√				√	
Limited noise and distractions for non-users				√	√	
Player interaction that enables users to choose what to do within limits, while still following rules	√	√		√		
Encouraging environment that motivates students	√		√	√		√
Play environment that offers complements to 'real' play			√			√
Sophisticated user interface and content to match game players' expectations.	√			√		√

Figure 2-1: Evaluation of Current Educational Games

Educational Game / Key Components	The Book of Lulu	The Number Devil	Stationary Studio	Clifford the Big Red Dog: Phonics	I Spy Fantasy	Brother Bear
Record of student progress						
Adaptable level of challenge						
Non-identically repeated experiences						
Ability to save and restart games		√		√		√
Suitable stopping points throughout game play		√		√	√	
Instructor's manual including information on structure content and underlying game models		√	√			
Game scenarios mimic realistic expectations and physical properties of the real world				√	√	
User interface and instructions that do not require elaborate written instructions				√	√	√
Limited noise and distractions for non-users			√		√	
Player interaction that enables users to choose what to do within limits, while still following rules		√	√	√	√	√
Encouraging environment that motivates students	√	√	√	√		√
Play environment that offers complements to 'real' play						
Sophisticated user interface and content to match game players' expectations.	√		√			√

Figure 2-1 (cont): Evaluation of Current Educational Games

Reviewing the currently available educational games reveals that the majority of the critical components listed are implemented in at least a few of the games evaluated. However, three of the critical components listed above have either been implemented in only one currently available game or are not implemented at all. As such, to demonstrate the benefits of the design concepts presented in this thesis, these three components were selected for implementation in a functional, elementary level educational game developed as a proof of concept model.

The first component selected is the incremental advancement of problem difficulty to continually challenge students at the appropriate level. If the educational game only increases difficulty once the student has thoroughly demonstrated comprehension, then the student only progresses once he or she has become bored with the material. Additionally, if the educational game increases in difficulty as a concretely defined transition point, then the student may quickly feel overwhelmed, frustrated, or inadequate at the sudden inability to comprehend the new material.

The second component selected is to record student progress to enable instructors to analyze a student's strengths and weaknesses. Games that offer only an overall success rate offer no insight into actual student accomplishments and areas lacking comprehension, both of which are required to appropriately address the student's education.

Finally, the third component selected is the capability to tailor the game to an instructor's individual classroom specifications. Specialized software may adequately address a given subject matter, but may not be suited to the individual instructor's needs, making it difficult to justify the use of the game within the classroom setting.

III. DESIGNING EDUCATIONAL SOFTWARE

3.1 Examining the Underlying Problem

Game developers have long believed that the limited revenues in educational games have made it virtually impossible to develop a game capable of meeting all of the specifications needed to make it beneficial to the classroom. However, using today's software engineering concepts, the current educational game architecture can be modified to make implementation of every key component possible within reasonable budgetary constraints.

Perhaps the most inhibiting factor in educational games today is the lack of sound software architecture. Once the software manufacturer has formulated an idea for an educational game within an entertaining environment, focus is directed to quick implementation in order to have minimal time to market. Such hastily implemented programs do not give due consideration to software design issues. The result is often a highly coupled, minimally cohesive software application. Highly coupled applications are characterized by high dependency between the application subsystems. Thus, modifications to one subsystem will affect all other application subsystems that interact with the modified version [6, 11, 14].

Minimally cohesive applications are characterized by the lack of similarity between objects and activities within a given subsystem. In other words, it appears as if the subsystem was created by combining objects and activities based on an obscure, unknown, or non-existent set of criteria. Thus, when modifications are made to one object or activity within the subsystem, it is often difficult to distinguish if there are any additional modifications required as a result.

A sound software application will have architecture based on low coupling and high cohesion. In other words, the application should be divided into primarily independent subsystems based on a defined set of criteria that clearly indicates how the objects and actions of the subsystem are related. While this statement seems rather intuitive, it can have vast implications for game development.

First, low coupling and high cohesion dramatically increase the maintainability of the game source code. Since each separate component is contained within an independent subsystem, modifications to one application component can be easily isolated and completed within a minimal time frame without affecting the remaining application components. Additionally, component functionality can be verified for its accuracy independent of the application being developed.

Because consumer needs are continuously changing, maintainability enables to the code to be modified with relative ease to meet these ever changing needs. Thus, software applications with increased maintainability also have a higher tendency of survivability. Survivability implies that the application is flexible enough that it can continually meet the needs of the consumer over an extended period of time.

Low coupling and high cohesion also dramatically increase the reusability of application components. Because the application components are contained within an independent subsystem, multiple applications can effortlessly incorporate established components by including the subsystem within the project. Such reusability enables a reduction in the amount of implementation required when developing new applications.

Similar to reusability, portability enables the application to be used on several different platforms. Because the application's functionality is loosely coupled with the application's interface, developers can implement one functional set of components with multiple platform-dependent interfaces. Therefore, a single application can now meet a broader audience.

High coupling and low cohesion, as pertaining to educational game design architecture, is most evident in the application's functionality extensively interwoven within the application's interface. Such poorly designed architecture restricts the application's functionality to the single game being developed, as components cannot be easily isolated for reusability. Additionally, such restriction prevents the educational game from being updated, expanded, or easily maintained, making the game virtually obsolete from its introduction [6, 22].

3.2 Developing a Sound Educational Game Architecture

Educational games can avoid such obsolescence by reevaluating the game's architecture. At the highest level, the educational game's functionality needs to be implemented independent of the game's interface. This enables a single game functionality to be contained with various types of educational game environments. For example, a mathematics – based game can be presented as both a Drill and Practice Game as well as a Content Game by modifying only the game's interface. Conversely, a single game interface, such as that of a Half and Half Game, can be used to present a mathematics game, a science game, and a reading comprehension game by modifying only the focus of the educational content outside of the game's interface.

Once the educational game is subdivided into functionality and interface subsystems, these subsystems need to be modularized into subclasses based on the component's cohesion. For example, the interface subsystem should be partitioned into each of the modules presented to the user. Thus, the interface subsystem should have a separate subclass containing the implementation of the welcome screen all users interact with when initializing the game. A second, separate subclass should be used to implement the module for establishing a new user account. Likewise, any additional, independent module presented to the user should be implemented within its own subclass of the interface subsystem.

Analogous to the interface subsystem, the functionality subsystem also must be partitioned based on each of the components implemented for the game. For example, the functionality subsystem should be subdivided into separate subclasses for the educational content presented, the game play semantics, the scoring mechanisms, and user movement between the different challenge levels. Because each of these subclasses is still rather large, implementing a variety of independent behaviors, these subclasses should be further modularized until each module contains only one distinct, independent object and its behaviors.

Designing an architecture that is modularized in this manner induces a low coupling and high cohesion application capable of meeting the specifications outlined for not only the current game being developed, but expansions and future games that can benefit from implementation already completed.

IV. IMPLEMENTATION OF AN EDUCATIONAL GAME

4.1 An Overview of the System Design

The educational game is divided into three key subsections, each addressing a separate function of the game. First, the user interfaces control all of the interactions between the users and the program. Next, the game play contains the presentation of the game content and the scoring of student progress within the presented game environment. Finally, the database subsection contains the specific, interchangeable information related to the game. This type of design enables each component to be developed virtually independent of the remaining subsections of the game, then be pulled together seamlessly with minimal interdependencies.

In addition to the three key subsections of the game listed above, the educational game also includes a separate, fourth package that implements the ability to write to an external file. Because such functionality is a separate, additional ability of the game, it is developed in its own independent package within the game project. This maintains modular code design with low coupling and high cohesion. A discussion of each of the three subsections follows in the proceeding chapters.

4.2 Game Play Development

The game play portion of the educational game is focused on the presentation of the game content and the scoring of student progress within the presented game environment. Recognizing that this needs to be developed independent of the material being presented, the package focuses on retrieving the appropriate information from an outside subsection – the database – and loading the information into the game. Additionally, the game play portion retrieves from the game the appropriate information regarding the student's interaction and then sends the information to the database section to appropriately record the information. Thus, as simplistic as these may seem, it serves to meet to of the outlined specifications of the educational game.

4.3 Game Content and Scoring

First, the game play portion controls the retrieving of the appropriate content for the student's level. Thus, as a student continues to interact with the game, the game play package must continually adapt the level of challenge to meet the student's demonstrated skills. While there is conflicting educational documentation as to how best to set up instructional design, most educational references believe that as a student consistently shows understanding of a given subject's difficulty level, questions of higher difficulty should be gradually introduced into the game play. Continued subject mastery through gradually increasing levels of difficulty ensures a thorough and complete comprehension of subject material. Conversely, the game must also be able to adapt if a student cannot demonstrate skills compatible with the questions being presented. If a student continually struggles with a subject's difficulty, questions of lower difficulty should be reintroduced to the student.

In order to continually challenge a student at his or her level requires a custom-tailored program that reacts appropriately to a student's demonstrated skills. Advancing question difficulty only after a student has fully demonstrated mastery of the previous challenge level results in boredom when a student has gained mastery, but has not yet completed the current level requirements, and frustration when a student cannot achieve success at the next challenge level. To dissolve these defined challenge levels, the developed game works to seamlessly blend challenge levels to meet the student's skills by increasingly challenging the student with higher difficulty questions while maintaining a level of success by integrating questions from one challenge level less than the student's current level. Determining the appropriate blend of these questions requires a more complex approach than the traditional method to ascertain the student's skill level.

The student's current challenge level is subdivided into a three-tier hierarchy. First, the student has a given subject in which he or she is attempting. For demonstration purposes, the developed game tests basic addition, subtraction, multiplication, and division at the elementary level.

Within each subject, there are different difficulty levels representing the complexity of questions within the category. To accommodate for expansion, the difficulty level begins at level 1 and increases, providing an unlimited number of levels that can be contained within a given subject category.

Finally, within each difficulty level are a series of point levels that indicate the student's current mastery of the subject at the given difficulty level. The student's score indicates in which of the four points levels a student resides. These levels determine the blend of question difficulty levels presented to the student. For example, if the student has a score of 3.25 points, 25% of the questions presented will be from the previous difficulty level and 75% of the questions presented will be from the current difficulty level. When a student has earned 15 points, the questions from the student's current level are no longer supplemented with questions from the previous difficulty level, but instead with questions from the next difficulty level. Finally, when a student has earned 25 points, his or her difficulty level is increased to the next level and his or her points are reset to -5 points. Conversely, if a student does not demonstrate an understanding of the material and gradually decreases his or her score to -5 points, then the difficulty level is reduced to the previous level and the student's points are reset to 24.5. A summary of the four points level is given in the Figure 4-1. While the points distributions are prepackaged within the educational game developed, they can be modified to accommodate the differing needs of instructors.

Points Levels	Percentage of Questions from Previous Level	Percentage of Questions from Current Level	Percentage of Questions from Next Level
-5 to 0	40%	60%	0%
0 to 7	25%	75%	0%
7 to 15	15%	85%	0%
15 to 25	0%	80%	20%

Figure 4-1: Question Bank Breakdown Based on the Student's Points Earned

In order to verify that the appropriate percentage of questions from each difficulty level is being presented based on the student's current points level, questions are selected from a separate question bank subset representing the student's current level. The subset is created in two stages. First, all questions from the student's current subject and difficulty level are added to the question bank subset. Then, a count of the total number of questions pulled paired with the student's points level indicates how many questions should be pulled from either the previous challenge level or the next challenge level, as indicated by the student's points. For example, if 100 total questions were at the student's current subject and difficulty level, and the student currently has a score of 8 points, then 15% of the questions would be from the previous challenge level. Multiplying the total number of questions pulled form the student's current level by the percentage needed indicates the number of questions needed from the previous difficulty level. A random sampling based on the current time stamp is used to pull the required number of questions from the previous difficulty level. These randomly selected questions are then added to the question bank subset.

Once the question bank subset has been created, the game play content class will then randomly select questions from the question bank subset. As the student gains or losses points based on game play, moving between the different points levels, the question bank subset is reconfigured for the new challenge level.

While this method does not guarantee that the student will always receive the exact percentage of questions based on his or her current points level, it does seamlessly integrate questions from different levels while maintaining a non-identically repeated experience for each student. In fact, given the randomization used to select the questions included in the question bank subset and the randomization used to select the questions presented during game play, students who repeat a challenge level will not have the same experience as the last game play.

As implied in the game content section, the points earned by the student are an integral part of the game content. The points range from -5 to 24.5 points, subdivided into four distinct levels. When a student enters a new challenge level, he or she begins with -5 points. If the student answers the question presented correctly on the first try, 0.5 is added to his or her score. If the student incorrectly answers the question presented on the first try, but answers the question presented correctly on the second try, 0.125 is added to his or her score. Finally, if the student incorrectly answers the questions presented on both attempts, 0.25 is subtracted from his or her score.

V. USER INTEFACES

When designed effectively, user interfaces provide a visual representation of a software program's processes and capabilities in an intuitive, easy to follow layout, without revealing the complex implementation required to complete such tasks. Thus, a usability engineer must carefully consider what visual aspects enhance comprehension and productivity so that a user can move seamlessly through the application to reach a desirable end state without being inhibited by the application's complex implementation. The focus of this chapter is to examine and understand the user interfaces designed for the functional educational game developed.

5.1 User Interfaces Overview

Within the functional educational game developed, there are two different perspectives that can be invoked, each represented by its own set of user interfaces. The game initially opens to the Welcome Interface, which enables the respective interfaces based on the interactions with the user. If the user is determined to be a student, they can enter the game play portion of the game by entering his or her unique Student ID and clicking the Start Program button. Alternatively, the student can establish a new student account by clicking the New Account Setup button. This opens a new interface to get the necessary information from the user, then moves to the game play portion of the game.

The Welcome Interface also holds the ability to enable the instructor's perspective. By clicking on the small i button located in the lower left hand corner, the user is prompted with the Instructor's Sign On Interface. After entering the correct credentials, the instructor is moved to the Instructor's Interface. The Instructor's Interface contains all of the administrative abilities available, which each open either a corresponding interface or application. These interfaces are discussed individually in the following sections. Figure 5-1 and Figure 5-2 show the user interface flow of the educational game for the student perspective and the instructor perspective, respectively.

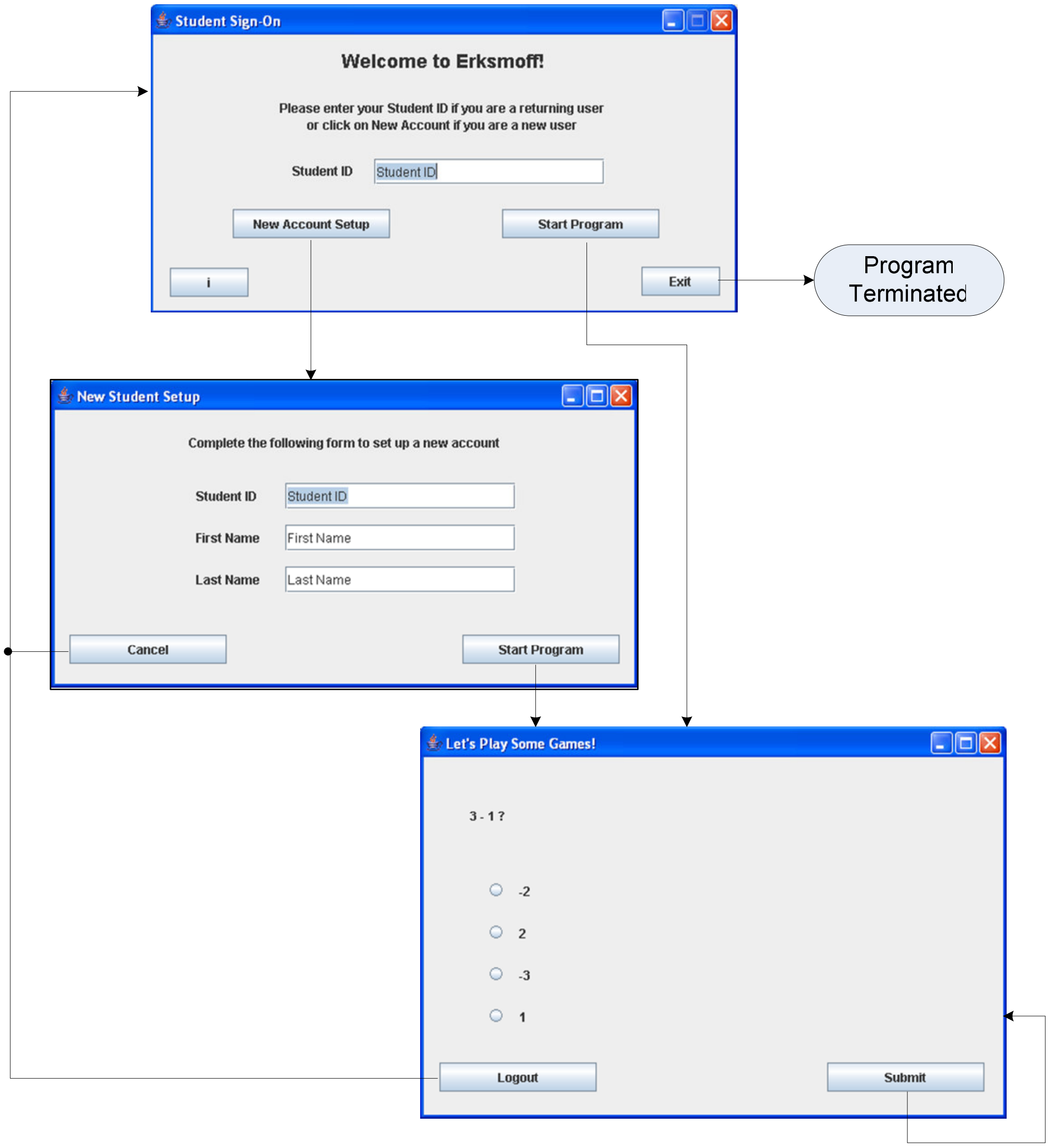
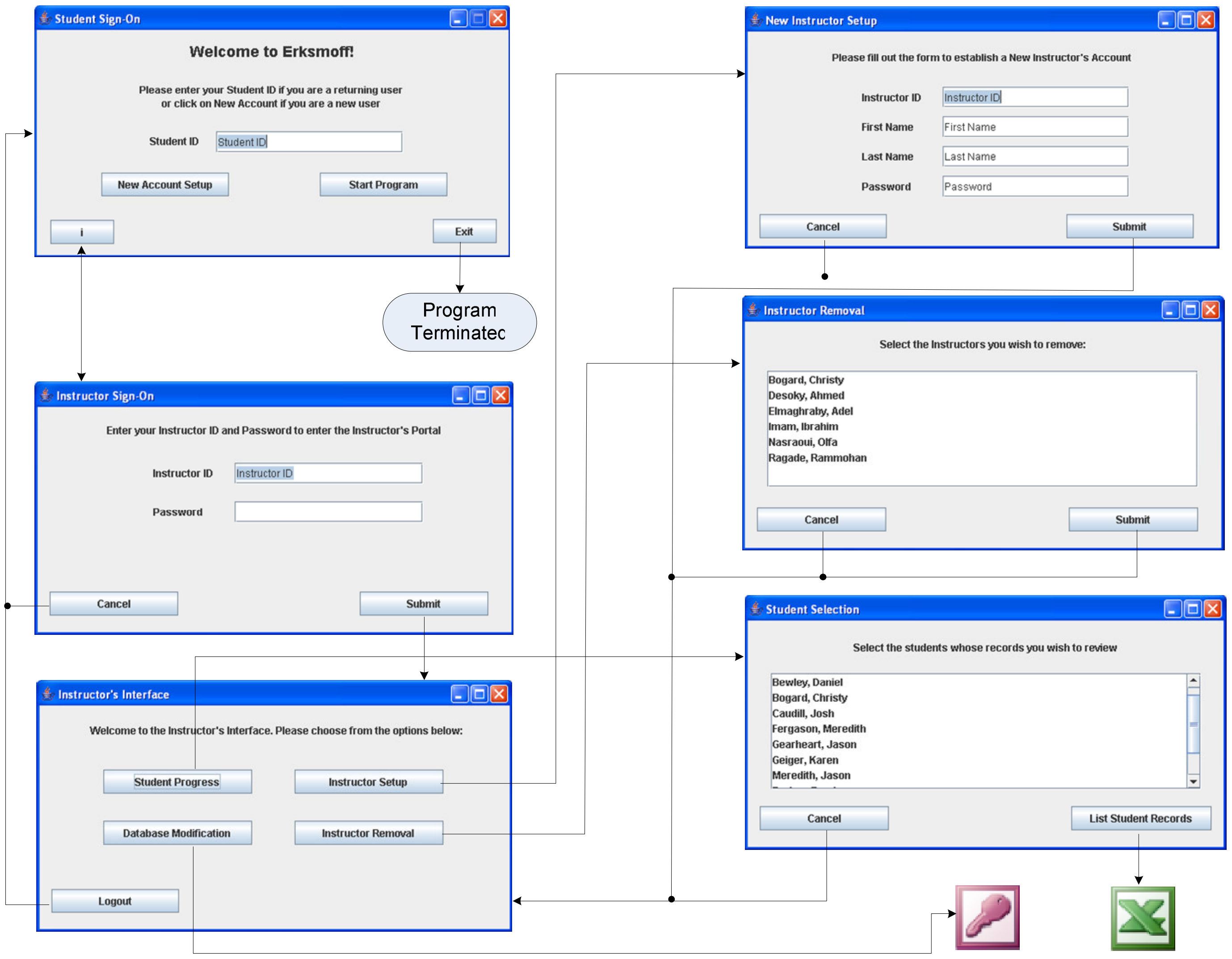


Figure 5-1: User Interface Interactions for the Student Perspective



Welcome to Erksmoff!

Please enter your Student ID if you are a returning user
or click on New Account if you are a new user

Enter your Instructor ID and Password to enter the Instructor's Portal

Welcome to the Instructor's Interface. Please choose from the options below:

Please fill out the form to establish a New Instructor's Account

Select the Instructors you wish to remove:

Bogard, Christy
Desoky, Ahmed
Elmaghraby, Adel
Imam, Ibrahim
Nasraoui, Olfa
Ragade, Rammohan

Select the students whose records you wish to review

Bewley, Daniel
Bogard, Christy
Caudill, Josh
Fergason, Meredith
Gearheart, Jason
Geiger, Karen
Meredith, Jason

Figure 5-2: User Interfaces Interactions for the Instructor Perspective

5.2 Welcome Interface

When the educational game is instantiated, the user is presented with the Welcome Interface. The primary purpose of the Welcome Interface is to establish the user perspective and enable the corresponding game features associated with the given perspective. Since the most common perspective is the returning student, the user is presented with the student sign-on. To sign into the game, the returning student enters his or her unique student id, then selects to the start the program by clicking the Start Program button.

Recognizing that a student may not be a returning student, the welcome interface also includes the option to set up a new account. Because establishing a new account is a one-time process for a given user, the process is segregated into an external form. New students can access the form by clicking the New Account Setup button that will prompt the student to create an account.

In addition to the student perspective, instructors also utilize the game to perform a variety of administrative tasks. In order to avoid interfering with the student game play, the Instructor's Interface is accessible by clicking the small i button in the lower left-hand corner. While not the most intuitive option available, current software applications often minimize the intrusiveness of administrative functions by hiding the functionality behind a small, dismissible button. Instructions for gaining administrative access are discussed in the User Manual associated with a given game. Upon clicking the i button to enable administrative aspect, a new visual interface is provided in order to maintain ease of use for the instructors as well.

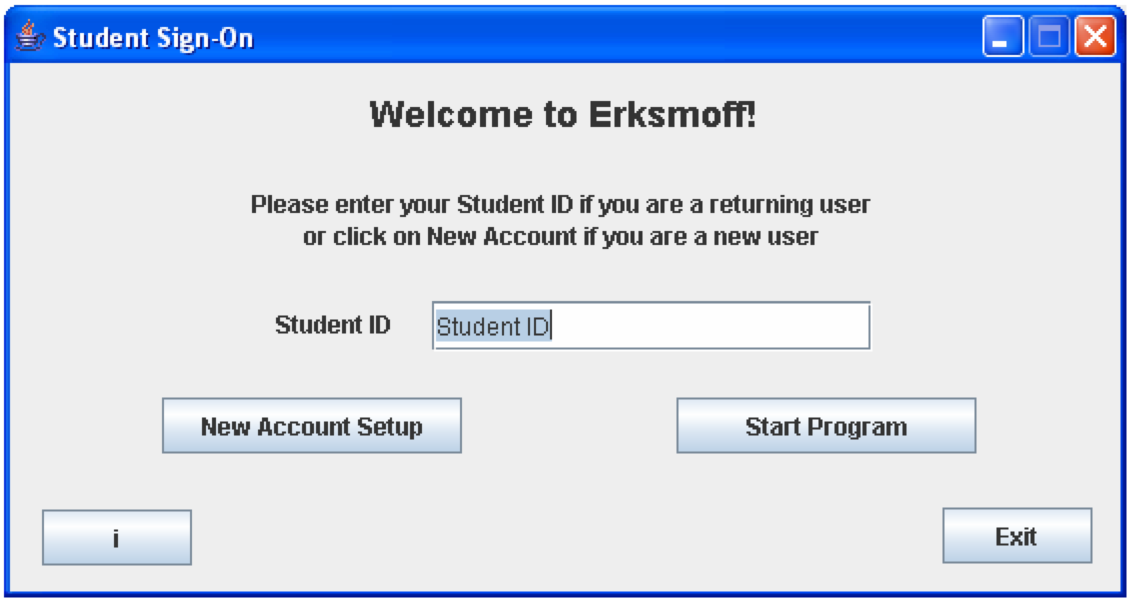

Student Sign-On
Welcome to Erksmoff!
Please enter your Student ID if you are a returning user
or click on New Account if you are a new user
Student ID
Student ID
New Account Setup
Start Program
i
Exit

Figure 5-3: Welcome Interface

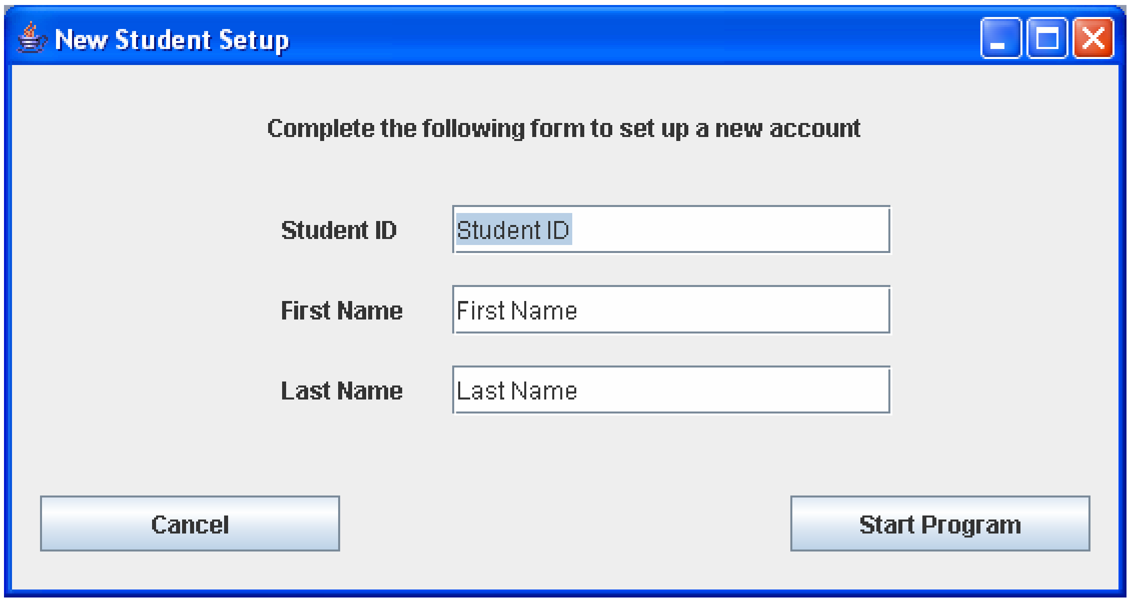
In addition to establishing the user's perspective, the Welcome Interface establishes a clean starting and ending points for the game environment. Once a user has completed their desired objectives, regardless of his or her perspective, the user can click on the Exit button to terminate the game. By ensuring that all users must terminate the game in the same manner ensures all remaining aspects can be terminated appropriately. For example, the educational game developed maintains a continual connection with an external database. Forcing the user to terminate the program with the Exit button verifies that the database connection will be closed appropriately.

Finally, it is important to note that the Welcome Interface also includes visual aesthetics that provide clarity for use. For example, instructions are given to inform the student, regardless if he or she is returning or new, as to how to sign on to the game. While this seems to be only a minor aspect, visual aesthetics can be the deciding factor to the ease of use of a software application.

5.3 New Student Account Setup

From the Welcome Interface, a new student can select the New Account Setup button to create a new student account. Selecting to open a new student account will instantiate the New Student Setup form. The New Student Setup form contains three input fields and two buttons. Each of the fields prompts the student to enter in a required piece of information. The first field is the Student ID, a unique identifier in which the student will use to sign on to the game. The second and third fields are the first and last name of the student, respectively. These enable the Student ID, which can be any unique combination of letters, numbers, and symbols, to be identified with a particular student by an instructor.

The two buttons included on the New Student Account Setup represent the two distinct actions in which the student can take. The first option is the ability to cancel the new account setup. Selecting this option clears the New Student Account Setup form and returns the user to the Welcome Interface. The second option establishes the new student account. In establishing a new student account, the program verifies all fields are completed, the Student ID is a unique identifier, and then starts the program game play.



Complete the following form to set up a new account

Student ID	Student ID
First Name	First Name
Last Name	Last Name

Figure 5-4: New Student Account Setup

5.4 Gaming Interface

Regardless as to whether the student instantiates the game play through the Start Program button from the Welcome Interface or the New Student Account Setup form, the Game Play Interface is opened. Because the game developed is intended to demonstrate functionality, it is designed as a Drill and Practice Game, meaning students are continually presented with a series of questions until they can demonstrate mastery. As such, the game play environment consists of only four components: the question presented, the list of possible solutions, the submit button, and the logout button.

When the game is in the play, the question label is replaced with the question content presented to the user. The student is also provided with four possible solutions in which he or she can choose by selecting the corresponding radio button. Once the student has made his or her selection, the student can finalize the answer by clicking the submit button. If the student has selected the correct answer, then he or she is presented with a new question. If the student has selected an incorrect answer, then he or she is given a second opportunity to select the correct response. After two attempts, a new question is presented to the student.

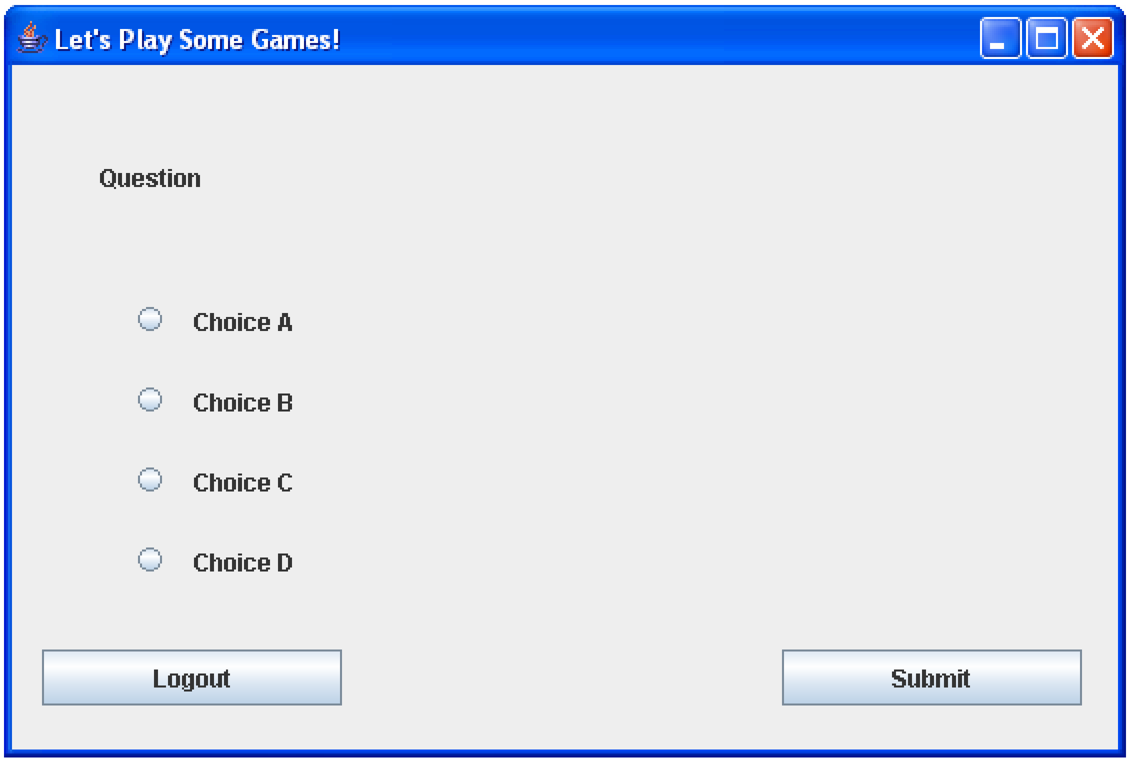


Figure 5-5: Gaming Interface

The fourth component of the game play interface is the ability to logout of the game. Clicking the logout button will return the student to the Welcome Interface, where he or she can exit the game entirely or a new user can being game play.

5.5 Instructor Sign On

The remaining user interfaces associated with the educational game are associated with the instructor's perspective. As discussed previously, the Instructor's Interface is accessible by clicking the small i button in the lower left-hand corner of the Welcome

Interface. Upon clicking the i button, the Instructor Sign On form is launched in order to verify the user's accessibility. An instructor will enter his or her Instructor ID and Password into their respective fields. Once completed correctly, the instructor can click the Submit button to open the Instructor's Interface. In the event the user is not a valid instructor, the user can click the Cancel button to return back to the game's Welcome Interface.

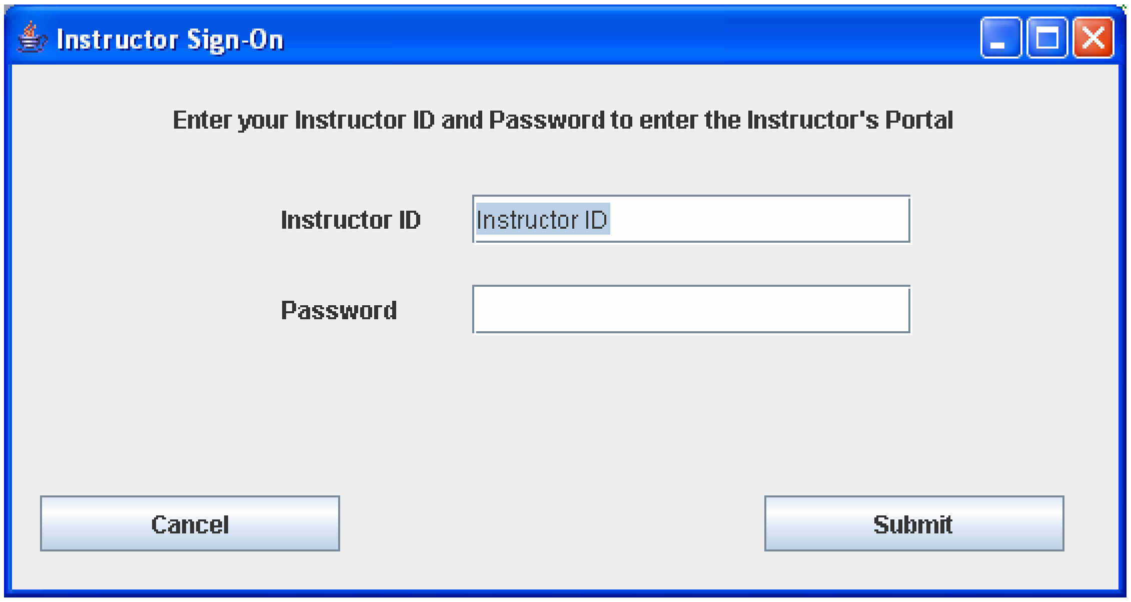

Instructor Sign-On
Enter your Instructor ID and Password to enter the Instructor's Portal
Instructor ID
Instructor ID
Password
Cancel
Submit

Figure 5-6: Instructor Sign On

5.6 Instructor's Interface

If the instructor has entered the correct credentials in the Instructor Sign On form, then the Instructor's Interface is opened. The Instructor's Interface contains the four administrative tasks in which an instructor can perform: review student progress, setup a new instructor, remove a current instructor, and modify the database associated with the game. Additionally, the Instructor's Interface contains a Logout button to exit out of administrative capabilities and return the game back to the Welcome Interface.

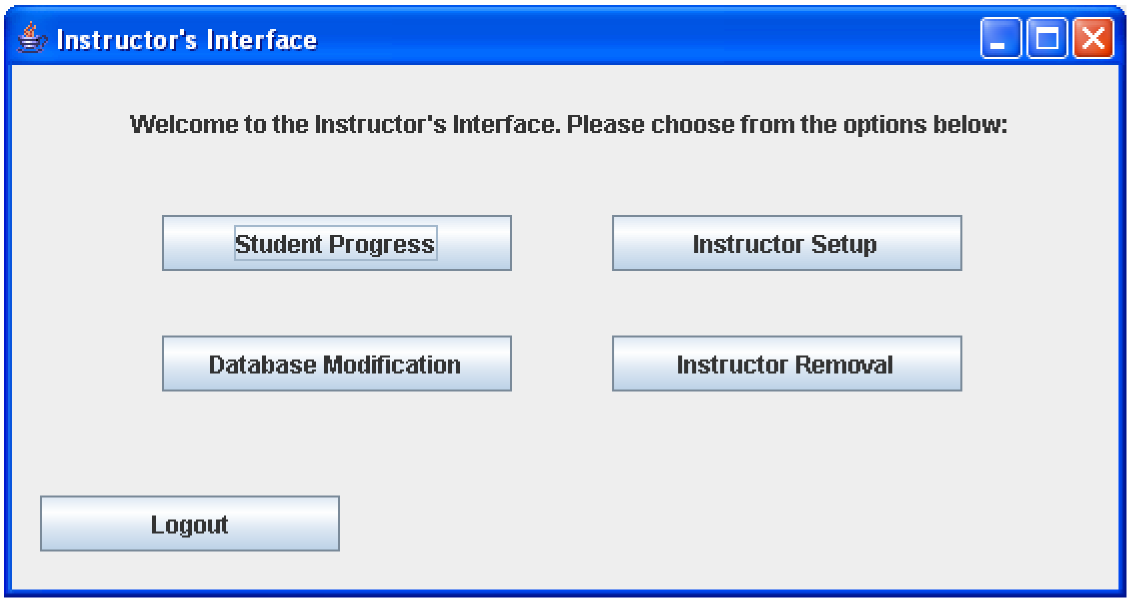


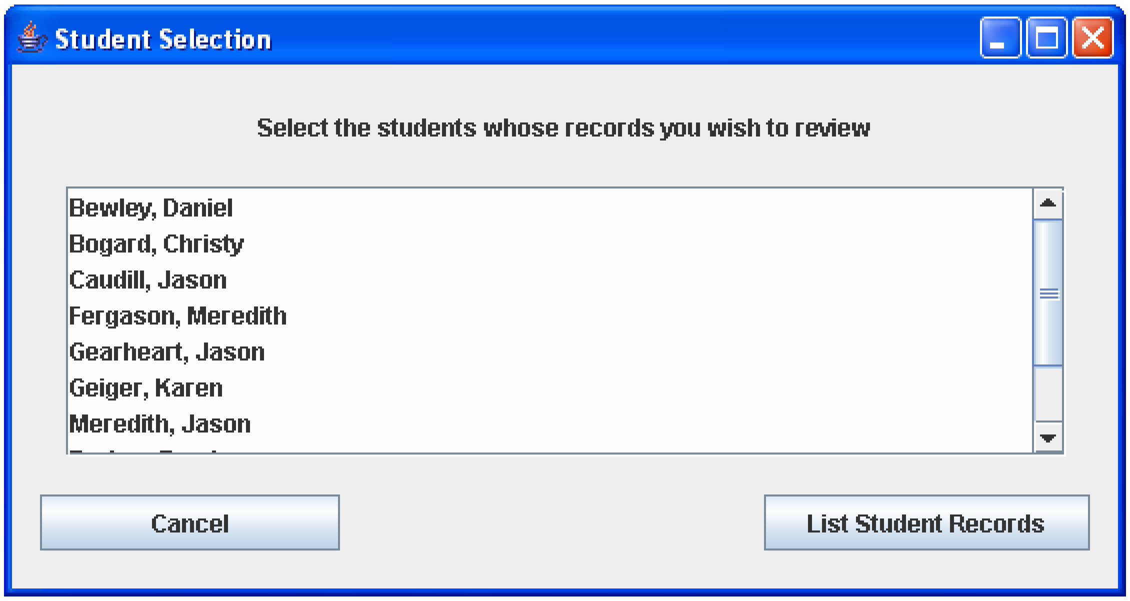
Figure 5-7: Instructor’s Interface

Each of the four administrative tasks is distinguished by a button contained within the Instructor’s Interface. Three of the four tasks instantiate an additional interface to obtain the additional information required, while the fourth directly performs the associated option. The first option, the Student Progress button, opens a separate interface in which the instructor selects from a list of current students which he or she wishes to review the progress of. Similarly, the Instructor Removal button opens a separate interface in which the instructor selects from a list of current instructors which he or she wishes to remove. The third option, the Instructor Setup option, opens a separate form in which the credentials of the new Instructor are entered. Finally, the Database Modification option directly opens the game’s corresponding database for editing, without requiring additional information from the instructor.

Using an independent interface for the instructor enables the educational game to be expanded for remote instructor access. By modifying the directory of the external game content to point to a centralized server rather than to the local terminal will allow instructors to be capable of accessing the administrative features remotely. Given the diversity of technology among educational institutions, this is an important game design attribute that enables the educational game to be adapted to the respected level.

<u>5.7 Student Selection</u>

The first administrative task, as described above, is the progress review of selected students. Choosing this option from the Instructor's Interface will open the Student Selection form containing a scrollable, alphabetical list of students who currently have a game account. An instructor then selects the student(s) in which he or she wishes to review and then clicks the List Student Records button.



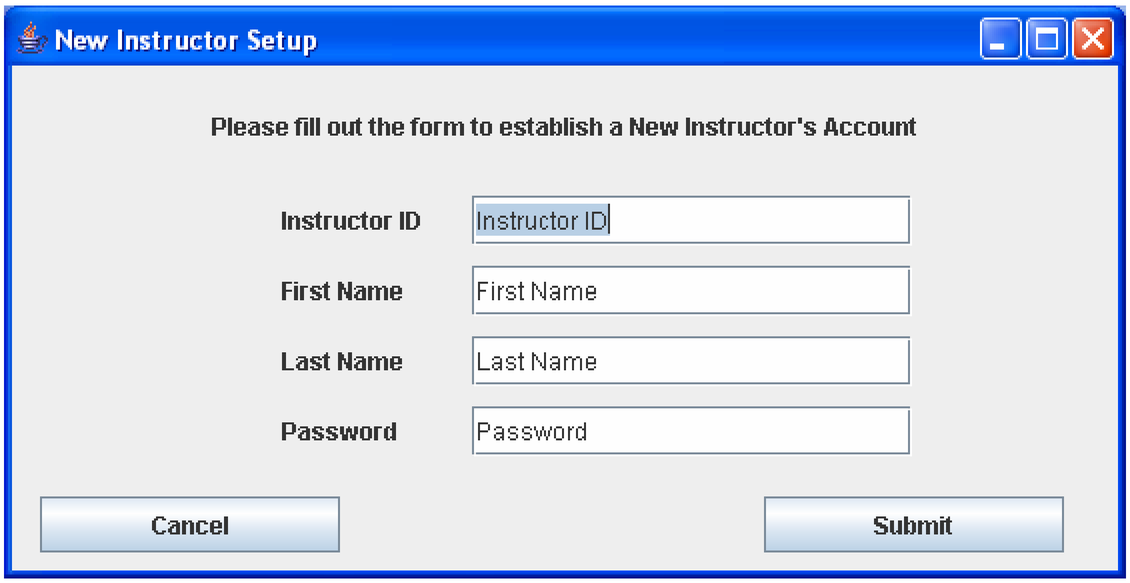
Bewley, Daniel
Bogard, Christy
Caudill, Jason
Fergason, Meredith
Gearheart, Jason
Geiger, Karen
Meredith, Jason

Figure 5-8: Student Selection

Once the instructor has selected to list the student records, the game pulls the appropriate records for each student selected, exports the results to a tab delimited text file, and returns the instructor back to the Instructor's Interface.

5.8 Add Instructors

The second administrative task capable through the Instructor's Interface is the ability to add additional instructors to the game. Selecting the Instructor Setup button opens a corresponding form prompting the administrator to enter a unique instructor id, the first and last name, and an associated password. Once the credentials are complete, the administrator then presses the Submit button to finalize the setup, and is returned to the Instructor's Interface. In the event the administrator chooses not to create the account, he or she can click the Cancel button to be returned to the Instructor's Interface.


New Instructor Setup

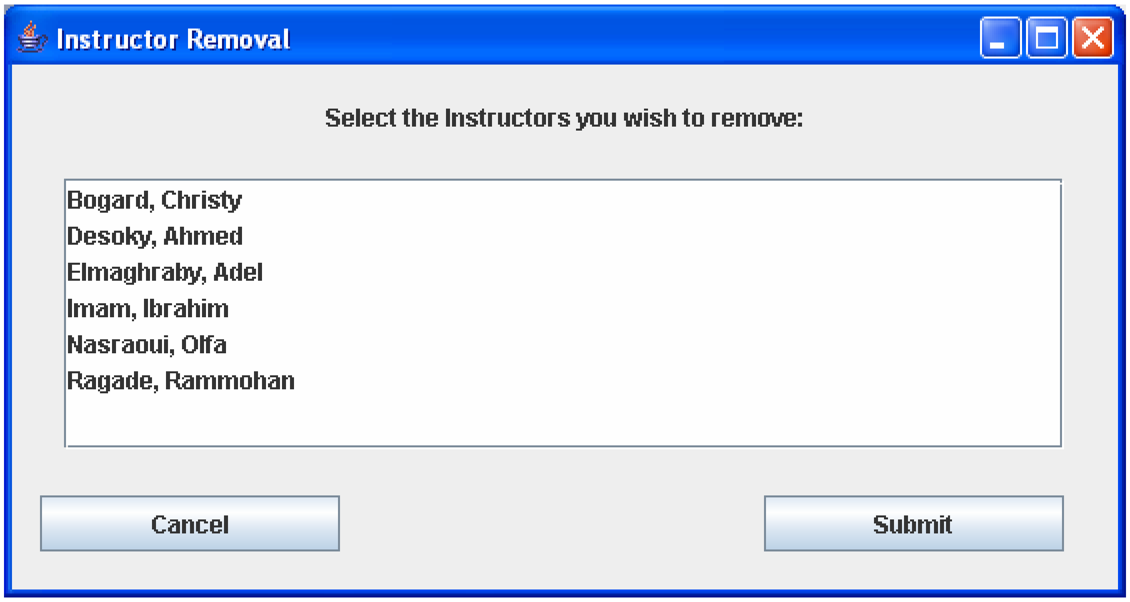
Please fill out the form to establish a New Instructor's Account

Instructor ID	Instructor ID
First Name	First Name
Last Name	Last Name
Password	Password

Figure 5-9: New Instructor Setup

5.9 Remove Instructors

Just as an instructor has the ability to add additional instructors to the game, the instructor can also remove instructors from the game by selecting the Instructor Removal button on the Instructor's Interface. Choosing this option from the Instructor's Interface will open the Instructor Removal form containing a scrollable, alphabetical list of current instructors. An instructor then selects the instructor(s) in which he or she wishes to remove and then clicks the Submit button to finalize the request. Once the instructor has submitted the request, he or she is returned to the Instructor's Interface. Again, a Cancel button is provided in the lower left-hand corner enabling the instructor to cancel the request prior to submission.



Bogard, Christy
Desoky, Ahmed
Elmaghraby, Adel
Imam, Ibrahim
Nasraoui, Olfa
Ragade, Rammohan

Figure 5-10: Instructor Removal

5.10 Database Modification

The Database Modification option, the fourth and final administrative task, directly opens the game's corresponding database for editing. Because the application can complete this task without requiring additional information from the instructor, no interface is necessary. The instructor remains at the Instructor's Interface.

5.11 Additional Considerations

It is important to note that the primary objective of the user interfaces is to demonstrate the necessary components and their functionality. It is not the intention of this thesis to portray the additional graphs that command the computer games currently on the market. These aesthetics are left as a future enhancement to the game. It is the intention of this thesis to demonstrate what aspects need to be included to address the sound software engineering architecture within a functional, elementary level educational game.

VI. DATABASE DESIGN

6.1 Overview of Database Design

The database design developed for the educational game created is simplistic in nature, but fully accomplishes the functionality necessary to meet the game specifications. It consists of four distinct tables, each representing a critical functionality developed within the gaming program. The Student Information Table contains records for each student game player. The Question Table contains the corresponding questions associated with the game. The Game Play Results Table contains each interaction between the users and the game. Finally, the Instructors Table contains a list of all the corresponding Instructors with administrative access to the game. These tables are discussed individually in the following sections.

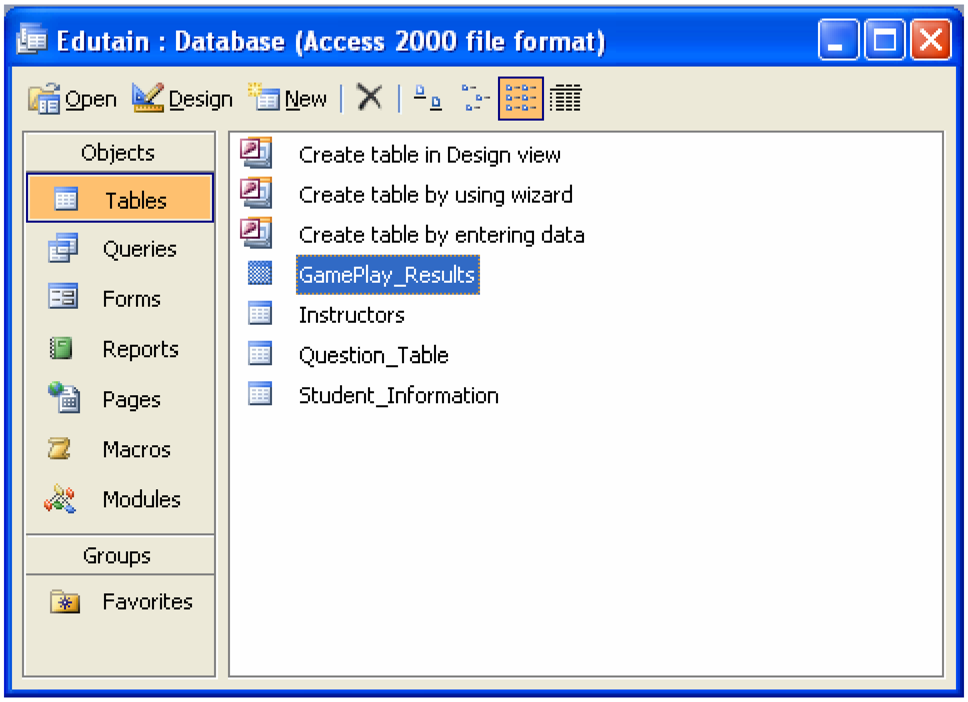
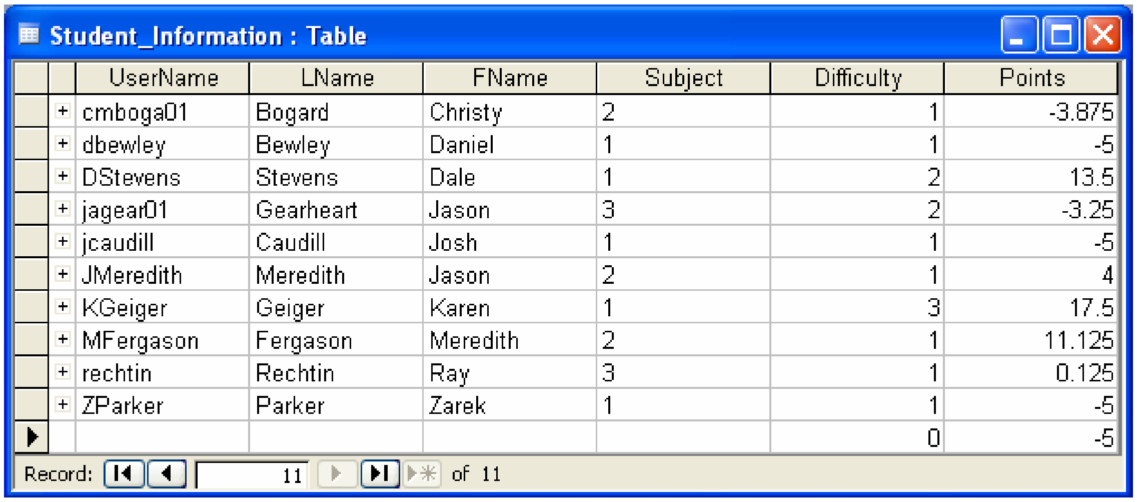


Figure 6-1: Database Tables

It is important to note that the database is compliant with Access 2000 File Format Specifications. While this is not the latest software file format for the database design, it offers a greater compatibility with more educational institutions software. Recognizing that educational institutions are faced with limited resources to purchase the most updated software applications, the database was created in an older file format so that institutions who have not upgraded, regardless of the reasons why, are still capable of benefiting from the educational game. Additionally, the updated Access 2003 File Format is backwards compatible, meaning databases designed for previous Access File Formats can still be read by the updated file format, ensuring that the educational game is not restricted to a limited target audience based on software compliance.

6.2 Student Information Table

The first table contained within the associated database is the Student Information Table. The Student Information Table contains a record for each student who has created a student account for the game. It contains six columns corresponding to the UserName, Last Name, First Name, Subject, Difficulty, and Points Earned. The UserName is the unique identifier in which the student uses to log into the game. The First and Last Name are used to identify each of the unique UserNames to the corresponding student. The Subject is a numerical value representing the corresponding questions in the Question Table, and the Difficulty is a numerical value representing a difficulty level of questions contained within each of the subjects. Finally, the Points represent the points currently earned by the student during game play.


Student_Information : Table

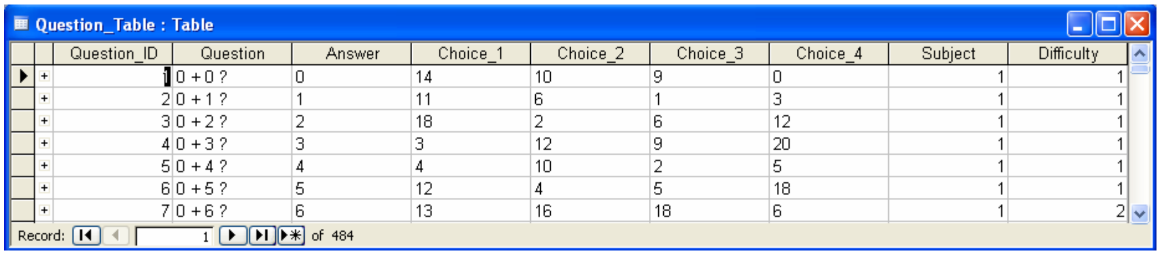
UserName	LName	FName	Subject	Difficulty	Points
cmboga01	Bogard	Christy	2	1	-3.875
dbewley	Bewley	Daniel	1	1	-5
DStevens	Stevens	Dale	1	2	13.5
jagear01	Gearheart	Jason	3	2	-3.25
jcaudill	Caudill	Josh	1	1	-5
JMeredith	Meredith	Jason	2	1	4
KGeiger	Geiger	Karen	1	3	17.5
MFergason	Fergason	Meredith	2	1	11.125
rechtin	Rechtin	Ray	3	1	0.125
ZParker	Parker	Zarek	1	1	-5
				0	-5

Record: 11 of 11

Figure 6-2: Student Information Table

6.3 Questions Table

Also contained within the database is the Questions Table. The Questions Table contains a list of all possible questions that can be asked during game play. Similar to each of the student records, each question has a corresponding unique Question ID. In addition to the unique Question ID, the record also contains the question being presented, the correct solution to the problem, four possible choices the user can select from, and the identifying subject and difficulty of the question.


Question_Table : Table

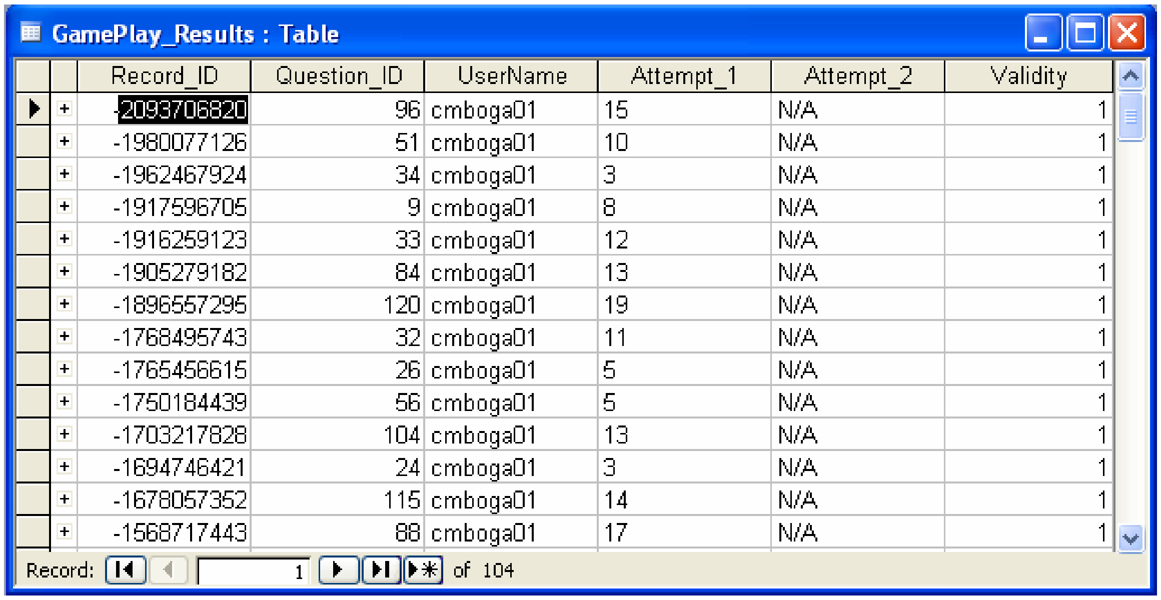
Question_ID	Question	Answer	Choice_1	Choice_2	Choice_3	Choice_4	Subject	Difficulty
1	0 + 0 ?	0	14	10	9	0	1	1
2	0 + 1 ?	1	11	6	1	3	1	1
3	0 + 2 ?	2	18	2	6	12	1	1
4	0 + 3 ?	3	3	12	9	20	1	1
5	0 + 4 ?	4	4	10	2	5	1	1
6	0 + 5 ?	5	12	4	5	18	1	1
7	0 + 6 ?	6	13	16	18	6	1	2

Record: 1 of 484

Figure 6-3: Questions Table

6.4 Game Play Results Table

The third table contained within the database is the Game Play Results Table. This table records the interactions of the student with the game. A unique, auto-generated number is assigned as the Record ID to uniquely identify each interaction with the game. The second column included in the table is the unique Question ID associated with the question being presented to the user. The UserName column indicates which student was presented with the question. Two columns are presented to record which answer the user selected for the first and second attempts respectively. Finally, a Validity column is included as an indicator of how well the student answered the question. If the student is able to correctly answer the question on the first attempt, the Validity column is assigned a value of one; if the student is able to correctly answer the question on the second attempt, the Validity column is assigned a value of half; finally, if the student was unable to correctly answer the question on either two of the attempts, the Validity column is assigned a value of zero.


GamePlay_Results : Table

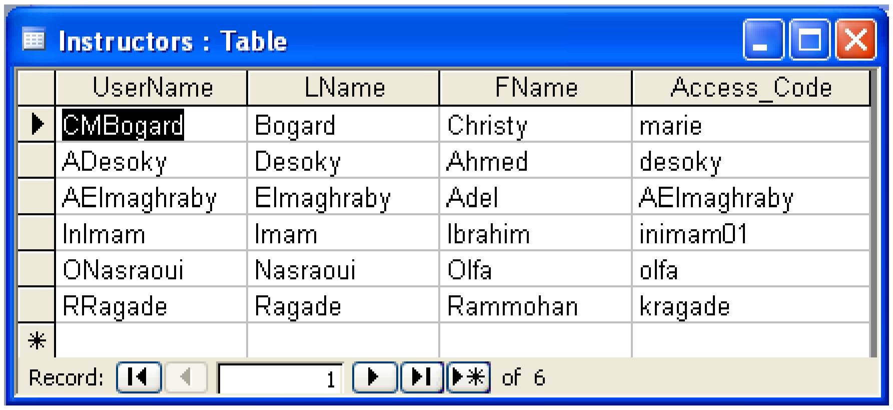
Record_ID	Question_ID	UserName	Attempt_1	Attempt_2	Validity
-2093706820	96	cmboga01	15	N/A	1
-1980077126	51	cmboga01	10	N/A	1
-1962467924	34	cmboga01	3	N/A	1
-1917596705	9	cmboga01	8	N/A	1
-1916259123	33	cmboga01	12	N/A	1
-1905279182	84	cmboga01	13	N/A	1
-1896557295	120	cmboga01	19	N/A	1
-1768495743	32	cmboga01	11	N/A	1
-1765456615	26	cmboga01	5	N/A	1
-1750184439	56	cmboga01	5	N/A	1
-1703217828	104	cmboga01	13	N/A	1
-1694746421	24	cmboga01	3	N/A	1
-1678057352	115	cmboga01	14	N/A	1
-1568717443	88	cmboga01	17	N/A	1

Record: 1 of 104

Figure 6-4: Game Play Results Table

6.5 Instructors

The last table included in the database is the Instructors Table. The Instructors Table includes a list of all of the instructors who have administrative access to the program. It includes the uniquely identifying UserName, the Instructor's corresponding First and Last Name, and the Access Code associated with the UserName to gain administrative access within the program.



UserName	LName	FName	Access_Code
CMBogard	Bogard	Christy	marie
ADesoky	Desoky	Ahmed	desoky
AElmaghraby	Elmaghraby	Adel	AElmaghraby
InImam	Imam	Ibrahim	inimam01
ONasraoui	Nasraoui	Olfa	olfa
RRagade	Ragade	Rammohan	kragade

Record: 1 of 6

Figure 6-5: Instructor's Table

6.6 Interdependencies

The four tables included in the database are subdivided into two distinct groups of relations, as outlined in Figure 6-5. First, and the simplest, is the instructor's relation. The Instructor's Table is isolated from the remainder of the database. It is self-contained, meaning it does not interact with any other table contained within the database. This design fits the intended purpose of the table: to provide a list of instructors granted administrative access to the program.

The second relation contained within the database contains the remaining three tables representing the game play and scoring functionalities of the database. Rather than duplicate information contained within the Questions Table and the Student Information Table, two critical relationships are defined between these two tables and the Game Play Results Table to link corresponding fields in each of the tables. First, rather than duplicating the question and its corresponding solution, possible choices, subject, and difficulty, only the unique Question ID is contained within the Game Play Results Table in order to reference the information already contained within the Questions Table. Similarly, the second relationship exists in order to reduce duplication from the Student Information Table. Rather than duplicating the information for a particular student, information that is continually changing, only the unique Student ID is contained within the Game Play results table.

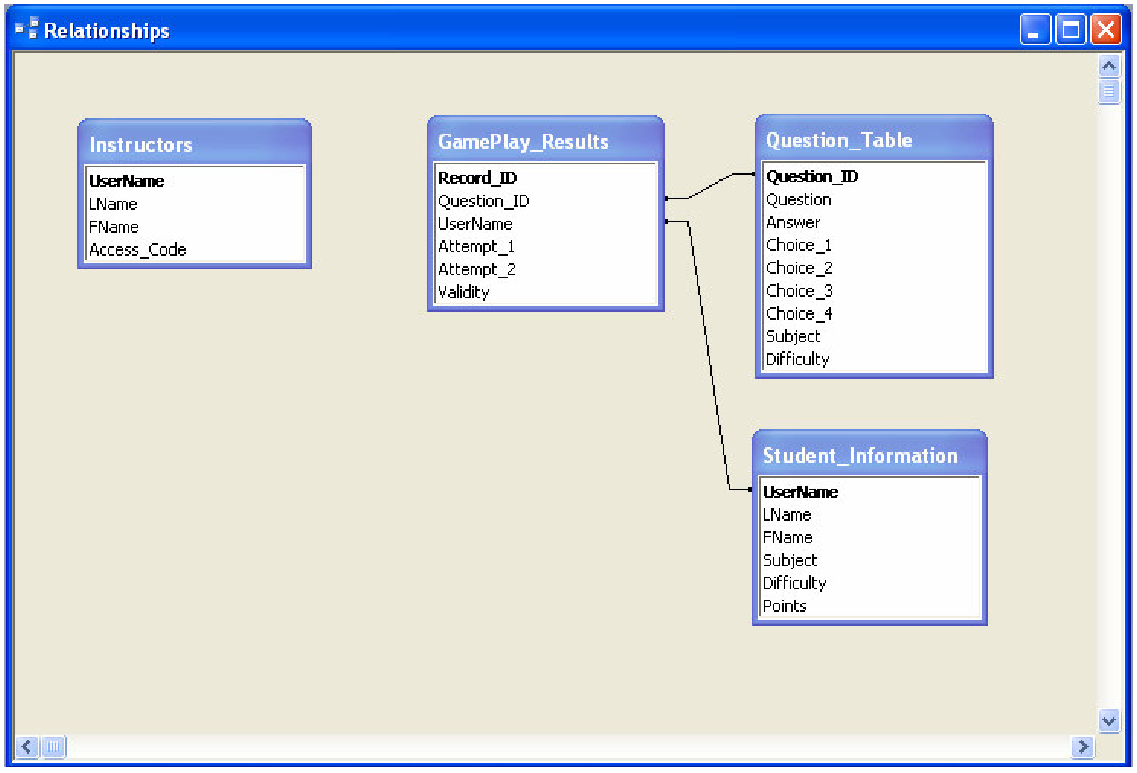


Figure 6-6: Table Relationships

VII. GAMING ARCHITECTURE

7.1 Gaming Architecture

Understanding the purpose of each component discussed thus far gives only a partial understanding of the game's architecture. It is important to understand how these components are designed as well as how they interact in order to comprehend how the game's design meets the specified requirements. In other words, the software architecture defines the structure of the source code that defines the program [1, 6, 22].

Within the developed educational game, there are four packages – database, file_access, game_play, and gui_Interface – each of which independently develops a component or requirement of the game. After reviewing each of these packages separately, an analysis of their interactions is discussed.

7.2 Database Package Architecture

The database package is responsible for establishing the connection to the external database, controlling all information retrieved from and passed to the external database, and closing the connection upon termination of the game. The package consists of four key classes. First, the DBConnection class is responsible for the connection and disconnection to the database. It is the only object class contained within the package. Upon entering the game, an instance of DBConnection is created. DBConnection verifies the correct system drivers are present and opens the connection to the database. When the user has completed their gaming interaction, DBConnection closes the termination to the database, and is then terminated.

DBAction is the second class contained with the database package. DBAction controls all of the information retrieved from and passed to the database, defining all of the possible interactions that can occur between the game itself and the external database. Because DBAction controls the interactions but is not an object itself, it is a static class, implying that an object of its type is never instantiated; only its class methods are called to complete the necessary database interaction.

In addition to DBConnection and DBAction, there are two support classes included within the package. First, SQLStatements is provided to correctly generate and format all sequel statements required by the DBAction class. This ensures standardization and compatibility across all statements. Removing the sequel statement constructs from the program enables ease of maintainability, as any necessary changes to the construct can be made from a single location without redundantly replicating the change throughout the DBAction class.

The second support class contained within the database package is the DBConstants class. In order to isolate the database architecture from the game, all database tables and constants are represented in a separate, static class. Thus, should there be dramatic changes to the database, only the table nomenclature would need to be modified from within the constants class, as all DBActions pull the constants from the constants class. Given such a purpose, it is important to note that the DBConstants class implements no methods, as the values contained are considered final, meaning they will not change during the course of game play.

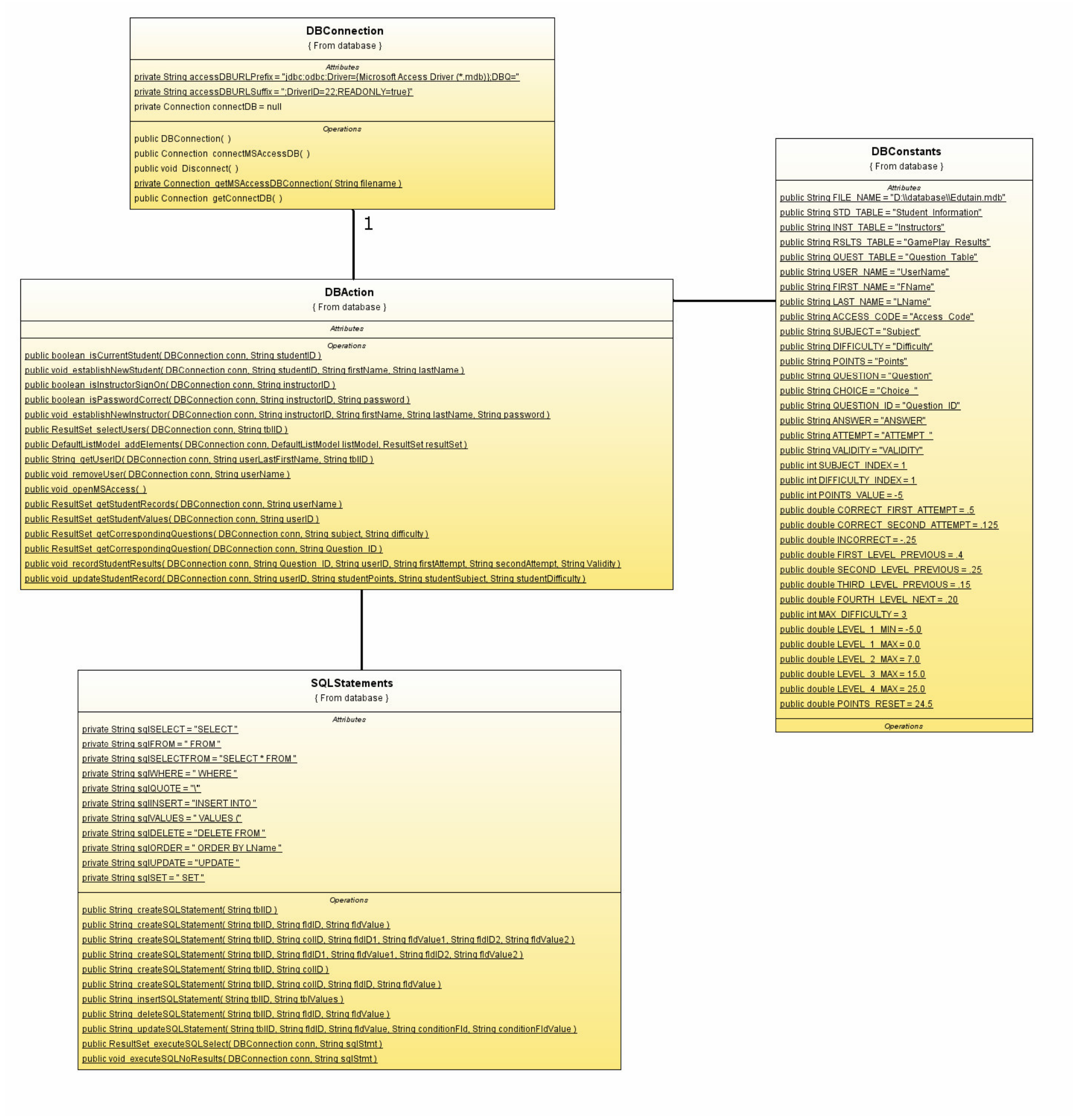

DBConnection
{ From database }
Attributes
private String accessDBURLPrefix = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ="
private String accessDBURLSuffix = ";DriverID=22;READONLY=true}"
private Connection connectDB = null
Operations
public DBConnection( )
public Connection connectMSAccessDB( )
public void Disconnect( )
private Connection getMSAccessDBConnection(String filename)
public Connection getConnectDB( )
1
DBAction
{ From database }
Attributes
Operations
public boolean isCurrentStudent(DBConnection conn, String studentID)
public void establishNewStudent(DBConnection conn, String studentID, String firstName, String lastName)
public boolean isInstructorSignOn(DBConnection conn, String instructorID)
public boolean isPasswordCorrect(DBConnection conn, String instructorID, String password)
public void establishNewInstructor(DBConnection conn, String instructorID, String firstName, String lastName, String password)
public ResultSet selectUsers(DBConnection conn, String tblID)
public DefaultListModel addElements(DBConnection conn, DefaultListModel listModel, ResultSet resultSet)
public String getUserID(DBConnection conn, String userLastFirstName, String tblID)
public void removeUser(DBConnection conn, String userName)
public void openMSAccess( )
public ResultSet getStudentRecords(DBConnection conn, String userName)
public ResultSet getStudentValues(DBConnection conn, String userID)
public ResultSet getCorrespondingQuestions(DBConnection conn, String subject, String difficulty)
public ResultSet getCorrespondingQuestion(DBConnection conn, String Question_ID)
public void recordStudentResults(DBConnection conn, String Question_ID, String userID, String firstAttempt, String secondAttempt, String Validity)
public void updateStudentRecord(DBConnection conn, String userID, String studentPoints, String studentSubject, String studentDifficulty)
DBConstants
{ From database }
Attributes
public String FILE_NAME = "D:\\database\\Edutain.mdb"
public String STD_TABLE = "Student_Information"
public String INST_TABLE = "Instructors"
public String RSLTS_TABLE = "GamePlay_Results"
public String QUEST_TABLE = "Question_Table"
public String USER_NAME = "UserName"
public String FIRST_NAME = "FName"
public String LAST_NAME = "LName"
public String ACCESS_CODE = "Access_Code"
public String SUBJECT = "Subject"
public String DIFFICULTY = "Difficulty"
public String POINTS = "Points"
public String QUESTION = "Question"
public String CHOICE = "Choice_"
public String QUESTION_ID = "Question_ID"
public String ANSWER = "ANSWER"
public String ATTEMPT = "ATTEMPT_"
public String VALIDITY = "VALIDITY"
public int SUBJECT_INDEX = 1
public int DIFFICULTY_INDEX = 1
public int POINTS_VALUE = -5
public double CORRECT_FIRST_ATTEMPT = .5
public double CORRECT_SECOND_ATTEMPT = .125
public double INCORRECT = -.25
public double FIRST_LEVEL_PREVIOUS = .4
public double SECOND_LEVEL_PREVIOUS = .25
public double THIRD_LEVEL_PREVIOUS = .15
public double FOURTH_LEVEL_NEXT = .20
public int MAX_DIFFICULTY = 3
public double LEVEL_1_MIN = -5.0
public double LEVEL_1_MAX = 0.0
public double LEVEL_2_MAX = 7.0
public double LEVEL_3_MAX = 15.0
public double LEVEL_4_MAX = 25.0
public double POINTS_RESET = 24.5
Operations
SQLStatements
{ From database }
Attributes
private String sqlSELECT = "SELECT "
private String sqlFROM = " FROM "
private String sqlSELECTFROM = "SELECT * FROM "
private String sqlWHERE = " WHERE "
private String sqlQUOTE = "\""
private String sqlINSERT = "INSERT INTO "
private String sqlVALUES = " VALUES ("
private String sqlDELETE = "DELETE FROM "
private String sqlORDER = " ORDER BY LName "
private String sqlUPDATE = "UPDATE "
private String sqlSET = " SET "
Operations
public String createSQLStatement(String tblID)
public String createSQLStatement(String tblID, String fldID, String fldValue)
public String createSQLStatement(String tblID, String colID, String fldID1, String fldValue1, String fldID2, String fldValue2)
public String createSQLStatement(String tblID, String fldID1, String fldValue1, String fldID2, String fldValue2)
public String createSQLStatement(String tblID, String colID)
public String createSQLStatement(String tblID, String colID, String fldID, String fldValue)
public String insertSQLStatement(String tblID, String tblValues)
public String deleteSQLStatement(String tblID, String fldID, String fldValue)
public String updateSQLStatement(String tblID, String fldID, String fldValue, String conditionFld, String conditionFldValue)
public ResultSet executeSQLSelect(DBConnection conn, String sqlStmt)
public void executeSQLNoResults(DBConnection conn, String sqlStmt)

Figure 7-1: Database Package Architecture

<u>7.3 File_Access Package Architecture</u>

The file_access package is responsible for exporting any information to an external file. Within the developed game, this class is used to export the student records to an external Microsoft Excel file so that instructors can retain the information outside of the game environment. Given only this single purpose, the file_access package consists of one functional class and one support class. File_Writer, the functional class contained within the package, creates the external file based on the tab delimited resultset passed to the method. The class then opens the corresponding application and file for the user. Similar to the DBAction class, the File_Writer class is a static class; it controls the interactions necessary to write to an external file, but is not an object itself.

The second class contained within the file_access package is the File_Constants support class. In order to isolate the operating system architecture and export file from the game, these constants are included in a separate, static class. Thus, changes required based on the given operating system can be made from a single location. Again, given such a purpose, it is important to note that the File_Constants class implements no methods, as the values contained are considered final.

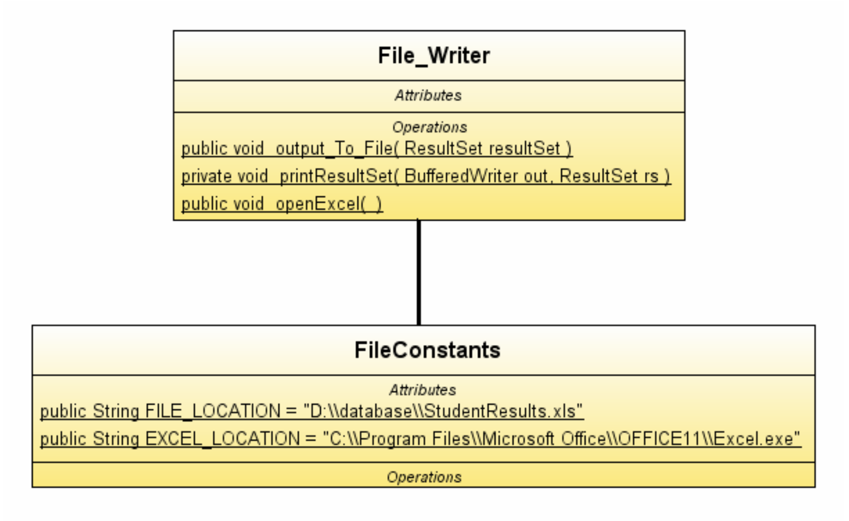

File_Writer
Attributes
Operations
public void output_To_File(ResultSet resultSet)
private void printResultSet(BufferedWriter out, ResultSet rs)
public void openExcel( )
FileConstants
Attributes
public String FILE_LOCATION = "D:\\database\\StudentResults.xls"
public String EXCEL_LOCATION = "C:\\Program Files\\Microsoft Office\\OFFICE11\\Excel.exe"
Operations

Figure 7-2: File_Access Package Architecture

7.4 Game_Play Package Architecture

The game_play package controls two primary functions of the game – correctly scoring the student responses to the game and loading the content of the game accordingly. As such, the game_play package consists of only two classes. First, the Game_Content class is responsible for the loading the appropriate questions and corresponding solution choices based on the current game level of the student. Once loaded, the Game_Content object waits for a student response, and then responds according to their progress.

The second class contained within the game_play package is the Game_Scoring class. The Game_Scoring class controls how the student is scored based on his or her given responses to the game. If the student is demonstrating comprehension, the Game_Scoring class informs the Game_Content class to provide more challenging questions. Conversely, if the student is lacking comprehension of the subject, then the Game_Scoring class informs the Game_Content class to provide less challenging questions. Further elaboration as to how the game responds to student interactions is discussed in Section 4.3 Game Content and Scoring.

Understanding that these two classes are highly cohesive within the game_play package indicates an increased level of aggregation. When the Game_Content class is instantiated, the Game_Scoring class will automatically be instantiated. This is an important element of the game_play package architecture, as neither of the classes can functionally exist without the other.

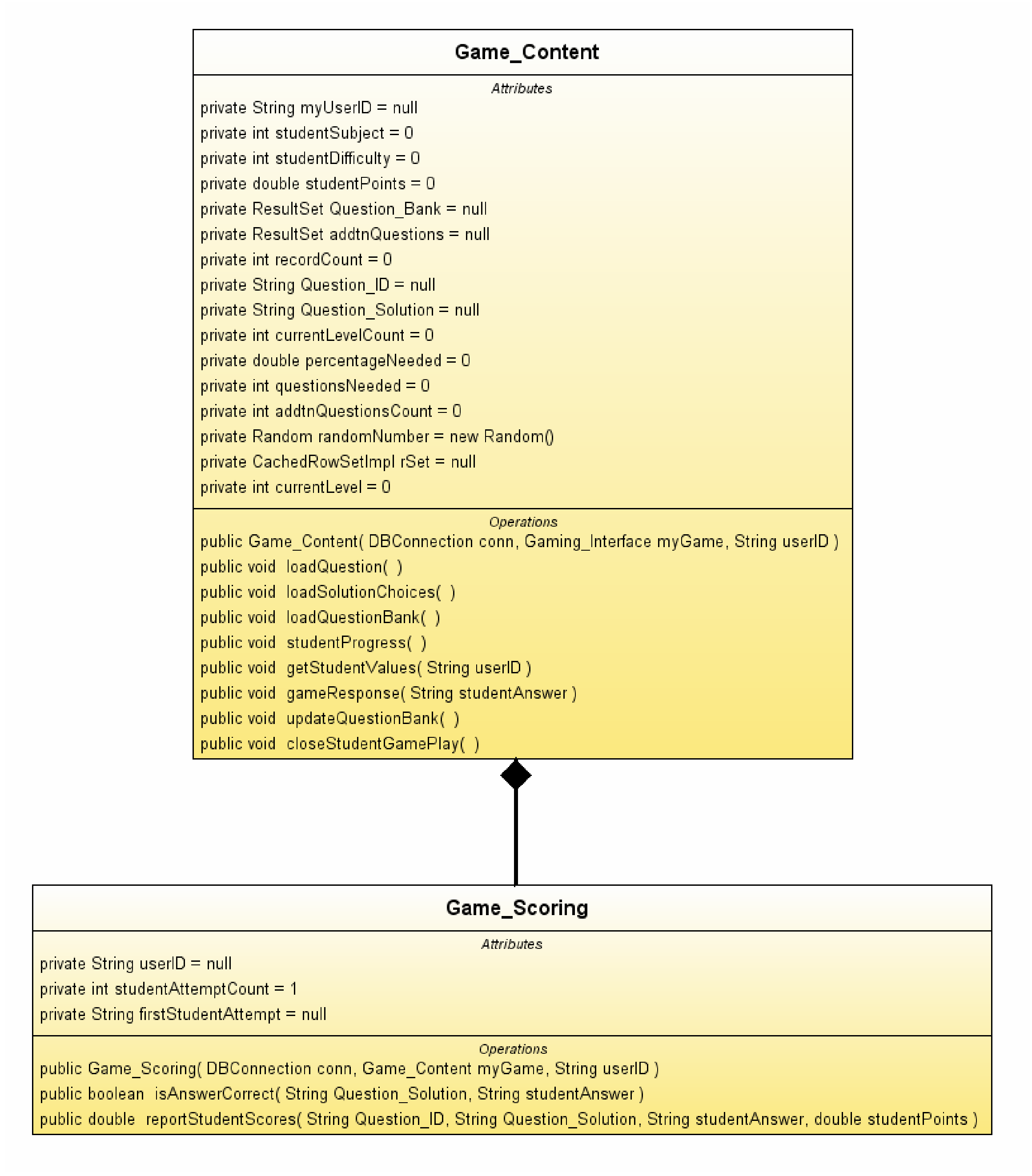

Game_Content
Attributes
private String myUserID = null
private int studentSubject = 0
private int studentDifficulty = 0
private double studentPoints = 0
private ResultSet Question_Bank = null
private ResultSet addtnQuestions = null
private int recordCount = 0
private String Question_ID = null
private String Question_Solution = null
private int currentLevelCount = 0
private double percentageNeeded = 0
private int questionsNeeded = 0
private int addtnQuestionsCount = 0
private Random randomNumber = new Random()
private CachedRowSetImpl rSet = null
private int currentLevel = 0
Operations
public Game_Content(DBConnection conn, Gaming_Interface myGame, String userID)
public void loadQuestion( )
public void loadSolutionChoices( )
public void loadQuestionBank( )
public void studentProgress( )
public void getStudentValues(String userID)
public void gameResponse(String studentAnswer)
public void updateQuestionBank( )
public void closeStudentGamePlay( )
Game_Scoring
Attributes
private String userID = null
private int studentAttemptCount = 1
private String firstStudentAttempt = null
Operations
public Game_Scoring(DBConnection conn, Game_Content myGame, String userID)
public boolean isAnswerCorrect(String Question_Solution, String studentAnswer)
public double reportStudentScores(String Question_ID, String Question_Solution, String studentAnswer, double studentPoints)

Figure 7-3: Game_Play Package Architecture

Because of the complex interweaving of the Game_Content and Game_Scoring classes, sequence diagrams are key to understanding the architecture between these two classes. There are two key scenarios that can exist between the two classes. First, the student has completed the previous question – regardless if completed correctly on the first attempt, second attempt, or incorrectly responded on both attempts – and the game content needs to load a new question. To do this, the Gaming_Interface calls the Game_Content class to establish the corresponding game response to update the educational content presented. The game response method recognizes that a new question is required, and as such, updates the corresponding question bank if necessary, determines and records the student's progress, loads a new question accordingly, and waits for a new response from the user.

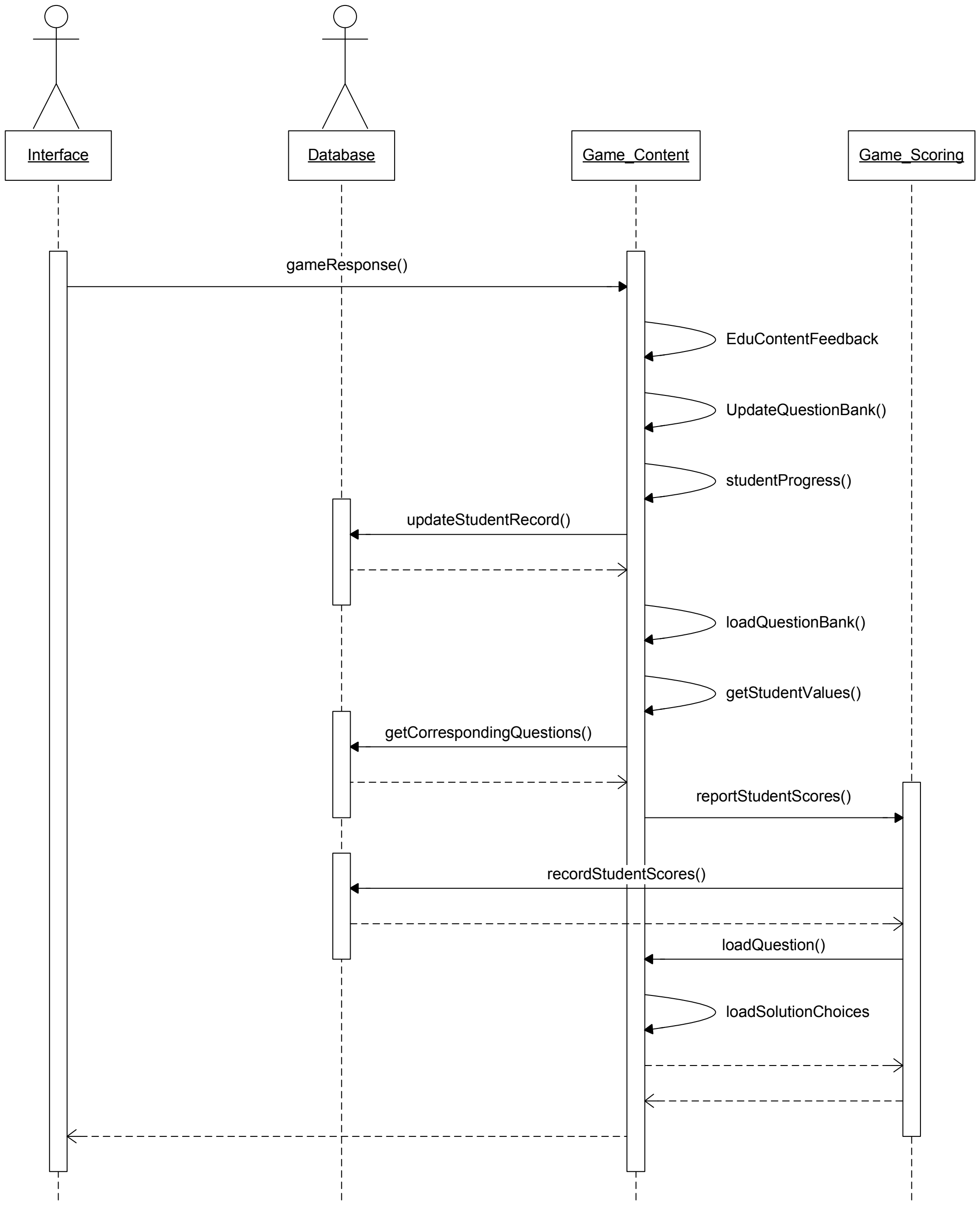


Figure 7-4: Sequence Diagram for Establishing New Game Content

The second key scenario is when the student has incorrectly responded to the game content on the first attempt. This scenario is different from the previous three because the game is to provide the student with the opportunity to attempt the question again. This scenario is similar to the first scenario, with a few critical adjustments. First, as with the first scenario, the Gaming_Interface calls the Game_Content class to establish the corresponding game response to update the educational content presented. The game response method recognizes that the student has incorrectly attempted the question once and needs to be provided with a second opportunity. As such, the Game_Content class checks to see if the questions need to be lowered in difficulty, determines and records the student's progress, and returns control back to the Gaming_Interface to wait for the user's second attempt to the question.

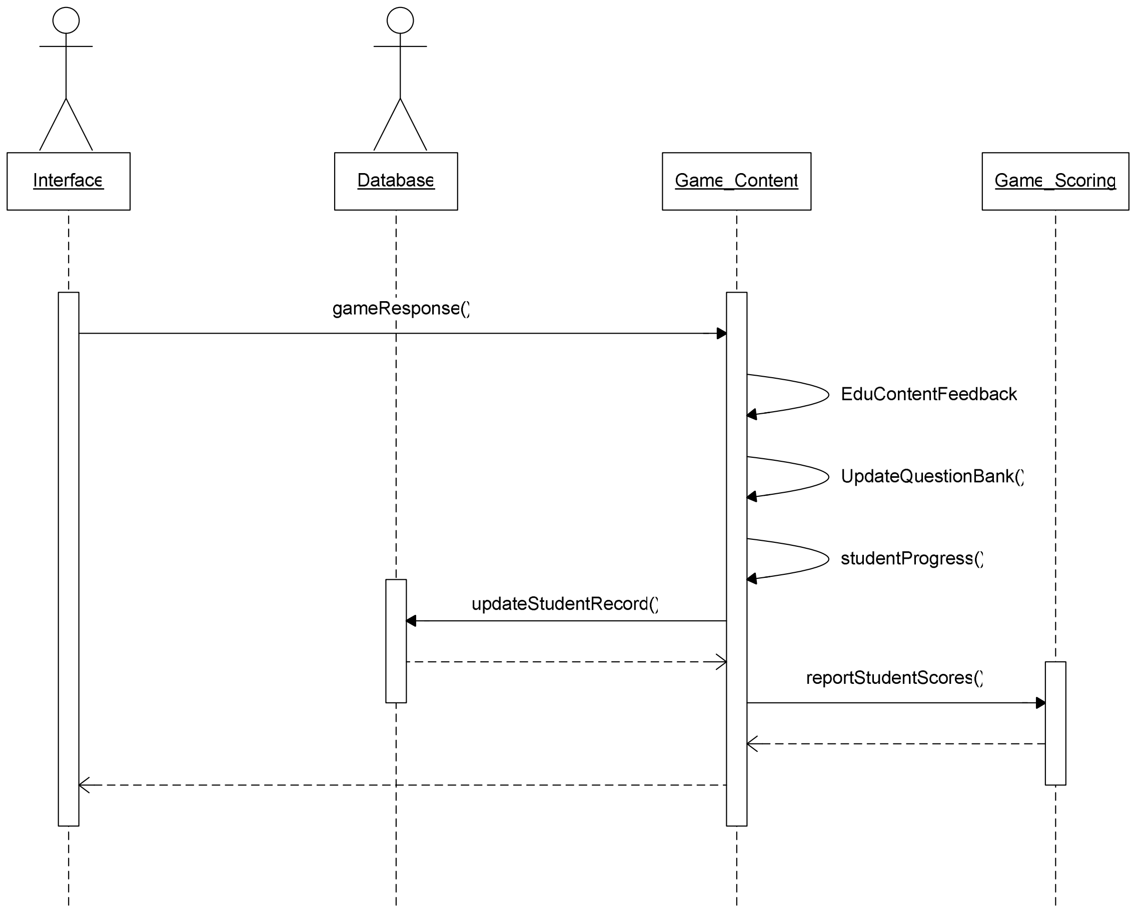


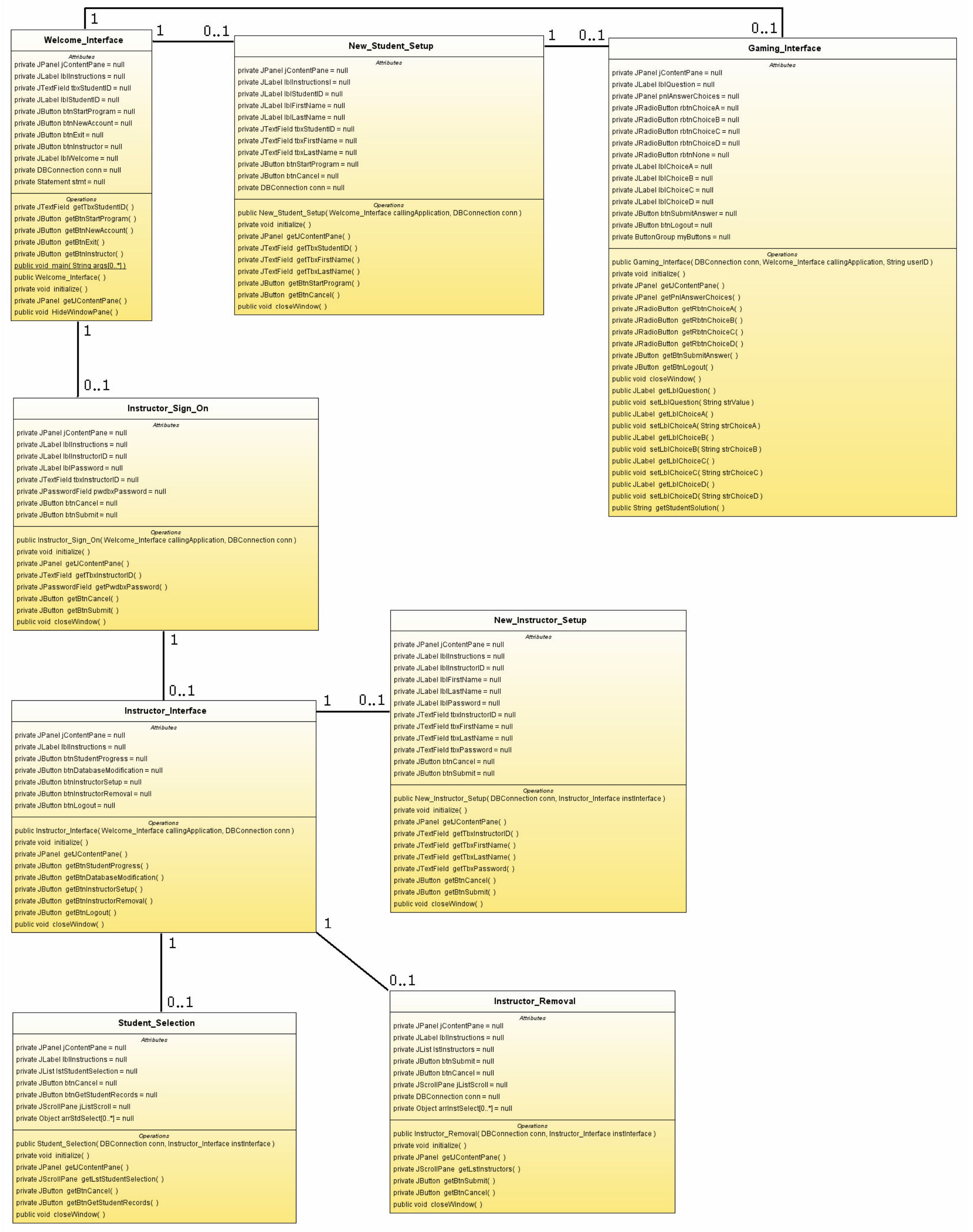
Figure 7-5: Sequence Diagram for Student's Second Attempt at the Question

7.5 Gui_Interface Package Architecture

The fourth and final package of the developed architecture is the gui_Interface package. The gui_Interface package controls all of the game's interfaces that control the interactions between the students. Each of the user interfaces contained within the gui_Interface package are discussed at length in Chapter 5.

It is important to understand that the gui_Interface package, which represents the user interfaces interactions, are architecturally designed to represent the game's flow. Thus, the game initially opens to the Welcome_Interface, which enables the respective interfaces based on the interactions with the user. If the user is determined to be a student, they can enter the game play portion of the game by entering his or her unique Student ID and clicking the Start Program button. Alternatively, the student can establish a new student account by clicking the New Account Setup button. This opens a new interface to get the necessary information from the user, then moves to the game play portion of the game.

The Welcome Interface also holds the ability to enable the instructor's perspective. By clicking on the small i button located in the lower left hand corner, the user is prompted with the Instructor's Sign On Interface. After entering the correct credentials, the instructor is moved to the Instructor's Interface. The Instructor's Interface contains all of the administrative abilities available to the instructor, including adding and removing additional instructors, reviewing student progress, and modifying the database associated with the game. Each of these game elements are invoked by either instantiating a corresponding interface or launching the corresponding application.



```
Welcome_Interface
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JTextField tbxStudentID = null
private JLabel lblStudentID = null
private JButton btnStartProgram = null
private JButton btnNewAccount = null
private JButton btnExit = null
private JButton btnInstructor = null
private JLabel lblWelcome = null
private DBConnection conn = null
private Statement stmt = null
Operations
private JTextField  getTbxStudentID( )
private JButton  getBtnStartProgram( )
private JButton  getBtnNewAccount( )
private JButton  getBtnExit( )
private JButton  getBtnInstructor( )
public void  main( String args[0..*] )
public Welcome_Interface( )
private void  initialize( )
private JPanel  getJContentPane( )
public void  HideWindowPane( )
```

```
New_Student_Setup
Attributes
private JPanel jContentPane = null
private JLabel lblInstructionsl = null
private JLabel lblStudentID = null
private JLabel lblFirstName = null
private JLabel lblLastName = null
private JTextField tbxStudentID = null
private JTextField tbxFirstName = null
private JTextField tbxLastName = null
private JButton btnStartProgram = null
private JButton btnCancel = null
private DBConnection conn = null
Operations
public New_Student_Setup( Welcome_Interface callingApplication, DBConnection conn )
private void  initialize( )
private JPanel  getJContentPane( )
private JTextField  getTbxStudentID( )
private JTextField  getTbxFirstName( )
private JTextField  getTbxLastName( )
private JButton  getBtnStartProgram( )
private JButton  getBtnCancel( )
public void  closeWindow( )
```

```
Gaming_Interface
Attributes
private JPanel jContentPane = null
private JLabel lblQuestion = null
private JPanel pnlAnswerChoices = null
private JRadioButton rbtnChoiceA = null
private JRadioButton rbtnChoiceB = null
private JRadioButton rbtnChoiceC = null
private JRadioButton rbtnChoiceD = null
private JRadioButton rbtnNone = null
private JLabel lblChoiceA = null
private JLabel lblChoiceB = null
private JLabel lblChoiceC = null
private JLabel lblChoiceD = null
private JButton btnSubmitAnswer = null
private JButton btnLogout = null
private ButtonGroup myButtons = null
Operations
public Gaming_Interface( DBConnection conn, Welcome_Interface callingApplication, String userID )
private void  initialize( )
private JPanel  getJContentPane( )
private JPanel  getPnlAnswerChoices( )
private JRadioButton  getRbtnChoiceA( )
private JRadioButton  getRbtnChoiceB( )
private JRadioButton  getRbtnChoiceC( )
private JRadioButton  getRbtnChoiceD( )
private JButton  getBtnSubmitAnswer( )
private JButton  getBtnLogout( )
public void  closeWindow( )
public JLabel  getLblQuestion( )
public void  setLblQuestion( String strValue )
public JLabel  getLblChoiceA( )
public void  setLblChoiceA( String strChoiceA )
public JLabel  getLblChoiceB( )
public void  setLblChoiceB( String strChoiceB )
public JLabel  getLblChoiceC( )
public void  setLblChoiceC( String strChoiceC )
public JLabel  getLblChoiceD( )
public void  setLblChoiceD( String strChoiceD )
public String  getStudentSolution( )
```

```
Instructor_Sign_On
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JLabel lblInstructorID = null
private JLabel lblPassword = null
private JTextField tbxInstructorID = null
private JPasswordField pwdbxPassword = null
private JButton btnCancel = null
private JButton btnSubmit = null
Operations
public Instructor_Sign_On( Welcome_Interface callingApplication, DBConnection conn )
private void  initialize( )
private JPanel  getJContentPane( )
private JTextField  getTbxInstructorID( )
private JPasswordField  getPwdbxPassword( )
private JButton  getBtnCancel( )
private JButton  getBtnSubmit( )
public void  closeWindow( )
```

```
New_Instructor_Setup
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JLabel lblInstructorID = null
private JLabel lblFirstName = null
private JLabel lblLastName = null
private JLabel lblPassword = null
private JTextField tbxInstructorID = null
private JTextField tbxFirstName = null
private JTextField tbxLastName = null
private JTextField tbxPassword = null
private JButton btnCancel = null
private JButton btnSubmit = null
Operations
public New_Instructor_Setup( DBConnection conn, Instructor_Interface instInterface )
private void  initialize( )
private JPanel  getJContentPane( )
private JTextField  getTbxInstructorID( )
private JTextField  getTbxFirstName( )
private JTextField  getTbxLastName( )
private JTextField  getTbxPassword( )
private JButton  getBtnCancel( )
private JButton  getBtnSubmit( )
public void  closeWindow( )
```

```
Instructor_Interface
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JButton btnStudentProgress = null
private JButton btnDatabaseModification = null
private JButton btnInstructorSetup = null
private JButton btnInstructorRemoval = null
private JButton btnLogout = null
Operations
public Instructor_Interface( Welcome_Interface callingApplication, DBConnection conn )
private void  initialize( )
private JPanel  getJContentPane( )
private JButton  getBtnStudentProgress( )
private JButton  getBtnDatabaseModification( )
private JButton  getBtnInstructorSetup( )
private JButton  getBtnInstructorRemoval( )
private JButton  getBtnLogout( )
public void  closeWindow( )
```

```
Student_Selection
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JList lstStudentSelection = null
private JButton btnCancel = null
private JButton btnGetStudentRecords = null
private JScrollPane jListScroll = null
private Object arrStdSelect[0..*] = null
Operations
public Student_Selection( DBConnection conn, Instructor_Interface instInterface )
private void  initialize( )
private JPanel  getJContentPane( )
private JScrollPane  getLstStudentSelection( )
private JButton  getBtnCancel( )
private JButton  getBtnGetStudentRecords( )
public void  closeWindow( )
```

```
Instructor_Removal
Attributes
private JPanel jContentPane = null
private JLabel lblInstructions = null
private JList lstInstructors = null
private JButton btnSubmit = null
private JButton btnCancel = null
private JScrollPane jListScroll = null
private DBConnection conn = null
private Object arrInstSelect[0..*] = null
Operations
public Instructor_Removal( DBConnection conn, Instructor_Interface instInterface )
private void  initialize( )
private JPanel  getJContentPane( )
private JScrollPane  getLstInstructors( )
private JButton  getBtnSubmit( )
private JButton  getBtnCancel( )
public void  closeWindow( )
```

Figure 7-6: Gui_Interface Package Architecture

7.6 Game Architecture

Understanding how each of the packages is architecturally designed lends itself to a discussion of the game architecture as a whole. When the game is instantiated, the Welcome_Interface calls upon the DBConnection class within the database package to establish a database connection, then waits for the user response. Depending on the user response, the Welcome_Interface launches the Gaming_Interface, the New_Student_Setup Interface, or the Instructor_Sign_On Interface. Each of these interfaces then corresponds to the interactions with the user by calling the appropriate methods from the DBAction class within the database package.

Once the student has entered the game play portion of the game, the Gaming_Interface corresponds with only the Game_Content class with the game_play package to control the appropriate response. The Game_Content class then becomes responsible to controlling the game scoring and corresponding content loaded by calling upon the Game_Scoring class and the DBAction class, respectively.

Alternatively, the instructor can instantiate the Instructor_Interface by verifying his or her credentials through the Instructor_Sign_On Interface. Once the instructor has entered the Instructor Interface, he or she can instantiate one of the administrative tasks by clicking on the corresponding button within the Instructor_Interface. Each of these subordinate interfaces then responds to the instructor's request by making the appropriate calls to the DBAction class.

Notice how closely the overall architecture of the game corresponds so closely to the gui_Interface package. This is to be expected, as the game responses are controlled through the user interactions within the corresponding interfaces.

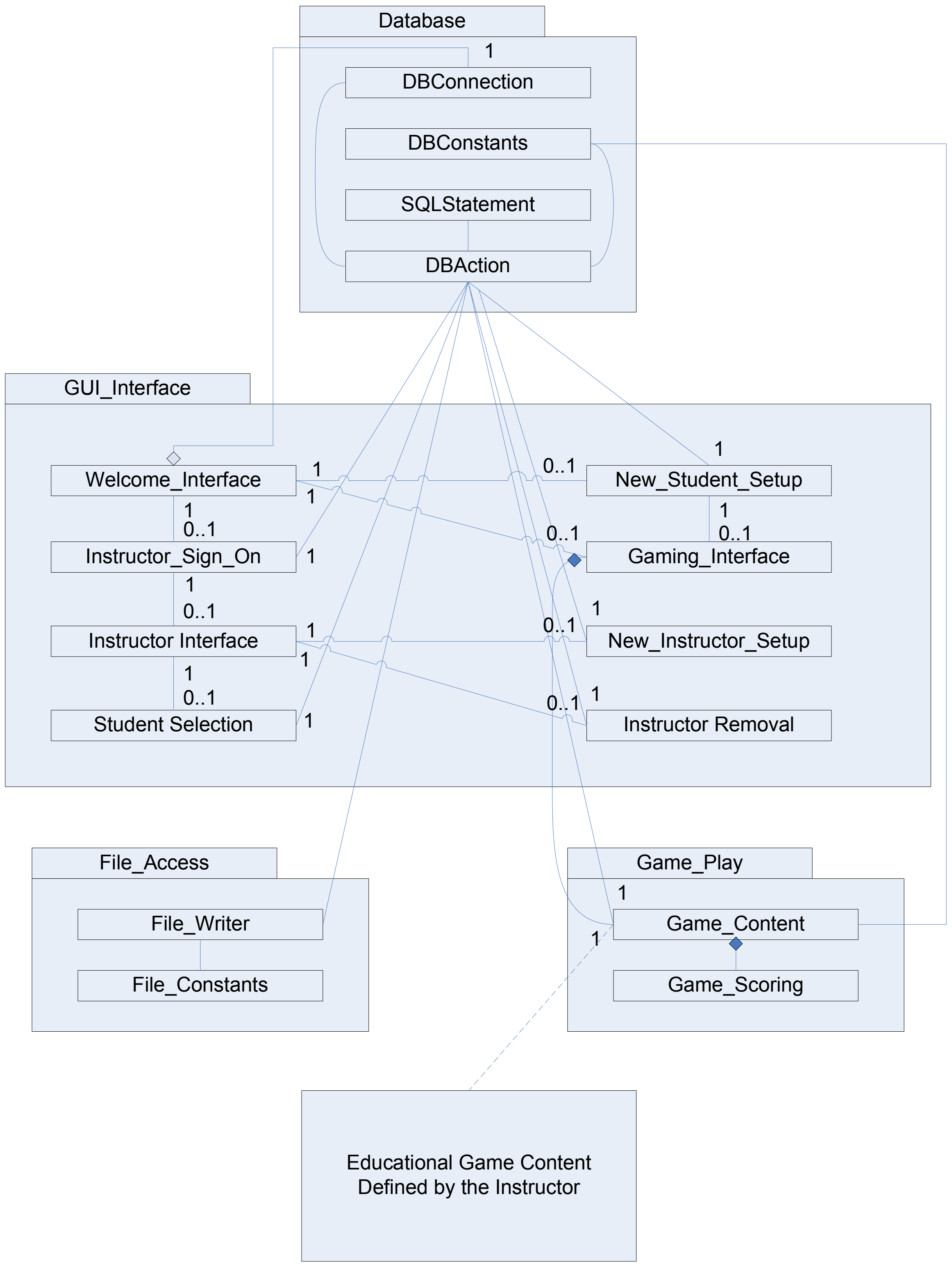


Figure 7-7: Game Architecture

7.7 Limitations

In addition to understanding the benefits achieved through the redesign of the educational game architecture, it is imperative that one understand the limitations of such a design relative to the different types of educational games available. As discussed in Chapter II, educational games can be divided into five general categories: Drill and Practice Games, Half and Half Games, Discovery Games, Content Games, and Non-Traditional Games.

Drill and Practice Games, the first type of game, place focus on continually presenting similar problems centered on a single concept. The student practices over and over until he or she can successfully demonstrate comprehension of the subject matter. This type of game lends itself to the modular architecture presented within the developed educational game because students are continually presented with a series of questions until mastery is demonstrated. Limited entertaining features and simplistic game environment make it ideal to separate content from game play. Such design enables designers to simply substitute different educational content into the game environment to meet the differing needs of instructors.

Half and Half Games are the second type of educational games. These games present educational content within an entertaining game environment. Though the complexity of the content has become more coupled with the game environment, this type of game still lends itself to the modular architecture presented. Currently, the content presented within the game is stored within an external database. The educational content contained within the Half and Half Game would need to be further sectored into two subsections. First, the game content would need to contain different entertaining scenarios in which could be used to establish the game environment. Separately, the actual educational content presented should be contained within another subsection. Thus, when the game is presented, the game engine selects a game environment, then populates it with the scenario content. Thus, even though the game has increased in complexity, one is capable of substituting different content scenarios without redesigning a new game.

The remaining three types of games, however, are far more difficult to develop based on the modular architecture presented. Discovery Games expand the Half and Half Games by shifting focus to the exploration aspect of the game. Thus, this type of game is centered around the development of a virtual world in which the user interacts. While some features and components can be easily reused among different games, it is virtually impossible to separate the content presented from the game environment. Thus, the focus is moved away from separating the content from the environment to separating the general game environment from the additional game components required.

Content Games, the fourth type of educational game, are also impaired by the complex coupling between the game environment and game content. This impairment is only further complicated by the increased emphasis on the entertaining environment aspect over the educational content presented. So, while the concept of a modular architecture can be applied to the game, it adds a level of complexity that often hinders the overall purpose of the game.

Finally, Non-Traditional Games represent the fifth type of educational game. Because these games were not originally developed for educational purpose, but have inadvertently presented some clear educational value, they often do not meet the architectural design of any educational game. These games are considered an anomaly in the educational game field and as such, are often accepted at face value.

7.8 Choosing the Implementation Language

In addition to designing an educational game with an architecture based on sound software engineering principles, it is also important that the game be developed in a language conducive to effectively communicating between necessary components for current and future educational games. However, there is currently no common implementation language used among developers, making the choice of implementation language worth further consideration.

The majority of current game development is with the C and C++ programming languages. However, both C and C++ have disadvantages that hinder game development. For example, C is considered the most efficient game development language, but is often too simplistic for complex games. C++, one of the most popular game development languages, has supporting components for virtually every aspect of game development, but has many of the lower level bugs with memory allocation and bounds checking from its inception from the C language.

Because of the limitations of C and C++, other languages are quickly entering the industry, each with its own advantages and disadvantages. For example, C++.Net incorporates many of the libraries and engines that eliminate memory allocation and bounds checking errors, but lacks the speed that can be achieved through the use of unmanaged languages. C#, a clean high productivity development environment integrates almost seamlessly with many currently available languages, but lacks the portability to platforms outside of Windows. VisualBasic.NET enables Rapid Application Development (RAD), Graphical User Interface (GUI), and easy integration of ActiveX controls and database elements, which are central components of many games, but lacks the ability to develop complex game environments needed in game development. Java offers ease of modular design through object oriented programming, but is not well supported by current game engines and game libraries [6, 14, 19, 20].

Thus, with the lack of consensus on the best game development languages, component development must either be developed independent of the game language or must be developed cross-platform to ensure its effective integration with current games. After reviewing the advantages and disadvantages of current gaming programming languages, two programming languages stood out as the best possible candidates for development of the functional educational game. First, Microsoft's Visual Studio .NET framework is one of the best currently available alternatives to addressing the cross-platform language barrier. .NET programs reside as modules within the common language runtime (CLR). The CLR decomposes language-specific source code to create runtime executables using a common intermediate language. Within the Visual Studio .NET framework, Visual Basic still remains as the best programming language for programmable databases, a key component of the functional educational game developed.

The second alternative considered was Java. Java offers ease in implementing modular programs through its object oriented programming. Additionally, the Java Virtual Machine enables programs to be developed independent of the computing platform, thus broadening the scope of compatible operating systems and limiting the additional software required. While the Java language lacks broad support by current game engines and game libraries, this is considered an inhibiting factor at this time. As more games are developed using Java, support will become more widely available. Thus, after careful analysis, Java was selected as the implementation language.

VIII. CONCLUSIONS AND FUTURE ENHANCEMENTS

While poorly designed architecture restricts the application's functionality to a single game being deployed, a modular architecture designed with low coupling and high cohesion can increase maintainability, survivability, reusability, and portability. This thesis recommends necessary architecture changes to best facilitate future game advancements and demonstrates sound software engineering principles through the development of a functional, elementary level educational game.

To meet the recommended modular architecture, the developed educational game is divided into three subsections – the game interfaces, the game play environment, and the educational content contained within. Each of these subsections function independently of the remaining sections, ensuring low coupling among the different packages. Within each package, there are several classes, each of which contributes a significant functionality to the specific component, representing the high cohesion among packages.

In addition to demonstrating the recommended modular architecture, the functional educational game developed implements three of the components listed by BECTa and TEEM as critical components to educational games and either not implemented or implemented in only a limited number of current leading educational games. These components are incremental advancement of problem difficulty to continually challenge students at the appropriate level, to record student progress to enable instructors to analyze a student's strengths and weaknesses, and the capability to tailor the game to an instructor's individual classroom specifications.

It is important to note that the developed educational game is intended to demonstrate the recommended modular architecture. As such, it does not implement all of the components listed as critical by BECTa and TEEM. An evaluation of the developed educational game compared with the list of components reveals several lacking components. Given the scope of the developed educational game, these lacking components are considered future enhancements.

Additionally, there are several crucial components included in professional, computer-based educational games that are omitted from the developed game. First, details concerning educational content within the game play environment are not addressed. The modular architecture of the developed game facilitates exchangeable educational content, thus enabling an instructor to satisfy any content-specific requirements he or she may have. This includes entertainment related content, gender specific or gender neutral content, and graphics related to content presentation associated with specific educational games.

Secondly, there are additional security concerns related to educational games not addressed within the developed game. For example, instructors signing into the instructor interface are authenticated with only their username and associated non-encrypted password. Professional educational games should include additional authentication and protection to prevent unauthorized individuals from altering game data.

Key Components \ Educational Game	Erksmoff
Record of student progress	√
Adaptable level of challenge	√
Non-identically repeated experiences	√
Ability to save and restart games	
Suitable stopping points throughout game play	√
Instructor's manual including information on structure content and underlying game models	
Game scenarios mimic realistic expectations and physical properties of the real world	
User interface and instructions that do not require elaborate written instructions	
Limited noise and distractions for non-users	
Player interaction that enables users to choose what to do within limits, while still following rules	
Encouraging environment that motivates students	
Play environment that offers complements to 'real' play	
Sophisticated user interface and content to match game players' expectations.	

Figure 8-1: Evaluation of Developed Educational Game

IX. REFERENCES

1. Albin, Stephen T. 2003. *The Art of Software Architecture: Design Methods and Techniques*. Indianapolis: Wiley Publishing.

2. Anderson, CA and BJ Bushman. 2001. Effects of Violent Video Games on Aggressive Behavior, Aggressive Cognition, Aggressive Affect, Physiological Arousal and Prosocial Behavior: A Meta-Analysis Review of the Scientific Literature. *Psychological Science.* Vol 12: 353-359.

3. Arter, Judith A and Jay McTighe. 2001. *Scoring Rubrics in the Classroom.* Thousand Oaks, CA: Corwin Press.

4. Bensley, L and Van Eenwky. 2001. Video Games and Real-Life Aggression: Review of the Literature. *Journal of Adolescent Health.* Vol 29: 244-257.

5. Bringing Educational Creativity To All. 2001. Computer Games in Education Project. UK: BECTa.

6. Bruegge, Bernd & Allen H. Dutoit. 2004. *Object-Oriented Software Engineering: Using UML, Patterns and Java (2nd edition).* Upper Saddle River, NJ: Pearson Prentice Hall.

7. Entertainment Software Association. 2005. 2005 Essential Facts about the Computer and Video Game Industry. E^3 Annual Conference. Los Angeles, CA.

8. Garris, Rosemary and Robert Ahlers. 2002. Games, Motivations, and Learning: A Research and Practice Model. *Simulation and Gaming.* Vol 33 (4): 441-467.

9. Griffiths, M D. 1999. Violent Video Games and Aggression: A Review of the Literature. *Aggression and Violent Behavior.* Vol 4: 203-212.

10. Harris, J. 1999. Secondary School Students' Use of Computers at Home. *British Journal of Educational Technology,* Vol 30: 331-339.

11. Harvey, James. 1995. The Market for Educational Software. *RAND.* US Department of Education.

12. Informa Telecoms & Media. 2005. *Dynamics of Games (5th edition).*London: Informa Media Group.

13. Interactive Digital Software Association. 2005. State of the Industry: Report 2003-2004. *IDSA Newsletter.* Washington, D.C: IDSA.

14. Maidment, Robert and Russell Bronstein. 1973. *Simulation Games: Design and Implementation.* Columbus, OH: Merrill.

15. McFarlane, Angela, Anne Sparrowhawk, and Ysanne Heald. 2002. Report on the Educational Use of Games: An exploration by TEEM of the contribution which games can make to the education process. Cambridge, MA: Teachers Evaluating Educational Multimedia.

16. Mitchell, Alice and Carol Savill-Smith. 2004. The Use of Computer and Video Games for Learning: A review of literature. London, UK: Learning and Skills Development Agency.

17. National Governors Association. 2002. *The Fiscal Survey of States.* Washington, DC: National Governors Association.

18. Orey, Michael, Mary Ann Fitzgerald, and Robert Maribe Branch. 2004. Issues and Trends in Instructional Technology. *Educational Media and Technology Yearbook 2004*. Westport, CT: Greenwood Publishing.

19. Roschelle, Jeremy, et. al. 1999. Developing Educational Software Components. *Web-Based Learning and Collaboration.* Los Alamitos, CA: IEEE Computer Society Press.

20. Sawyer, Ben. 1996. *The Ultimate Game Developer's SourceBook.* Scottsdale, AZ: Coriolis Group Books.

21. Siraj-Blatchford, John and David Whitebread. 2003. *Supporting Information and Communications Technology in the Early Years.* Berkshire, UK: Open University Press.

22. Sommerville, Ian. 2001. Software Engineering. Harlow: Pearson Education Limited.

23. Swords, Tara. 2005. At the Top of Their Game. *Dell Insight.* (Jan): 6-11.

24. Taylor, John and Rex Walford. 1978. *Learning and the Simulation Game.* Beverly Hills: SAGE Publications, Inc.

25. Trotter, A. 2003. Budget Crises Leads to Delays for Technology. *Education Week.* Vol 22 (34): 1-2.