**Agile Dinosaur Team Final Report**

**Project Overview**

Our project is an educational quizzing game focused on teaching people about a variety dinosaurs. We use a question bank of nearly 50 questions in the forms of multiple choice and true or false. Quizzes are 10 questions long, each using a random 10 questions from our question bank and avoiding repeat questions until most have been already been used. On the main menu the user can check their current score, the number of games played, and their overall high score achieved so far during their current play session. After the user selects an answer to their question they will be told if they were correct or not, and if they were incorrect they will be informed of the correct solution. This allows the user to learn the facts and have a greater chance of getting other questions correct by piecing their new knowledge together.

Our development team consisted of Joe Dain, Caroline Ganier, John Lasheski, Wayne Tolson, and Bradley Wetzel.

* The team lead and project manager was Bradley Wetzel.
* Data parsing was handled by Joe Dain.
* Our GUI was developed by Caroline Ganier.
* The file design and our unit tests were both created by John Lasheski.
* All of our diagrams were made by Wayne Tolson.

**Project Management**

Throughout our development we used an Agile Methodology to manage our project. This meant we were willing and able to work on the fly and adapt as necessary. If new ideas popped up later in the cycle or plans changed, we embraced it and continued forward progress.

To communicate within our group we utilized a variety of platforms. These included gmail, wiggio, github, and tinychat. Our group had set a mandatory weekly meeting, via tinychat, for Tuesdays at 7:30 PM EST. This will allow us to communicate all at once and address any issues or concerns and plan out what work needed to be completed by the end of the week. We used wiggio to announce major updates, and ask any questions that arose throughout the week. Github allowed us to ensure version control and provided us with a backlog of our work in case we ever needed to double check something from our past work. We uploaded all of our important files here so that they could be referenced by anyone on the team at will.

**Original Timeline and Milestones**

|  |  |
| --- | --- |
| **Important Dates** | **Milestones** |
| September 5th | Quiz question bank completed (Entire Group)  Test Plan laid out  Test Plan peer review completed |
| September 12th | Data parsing working correctly (Joe Dain)  Design layout agreed upon |
| September 19th | GUI constructed (Caroline Ganier)  Phase I source code compiled  Phase I source review completed |
| September 26th | Unit testing completed (John Lasheski)  Phase II source code compiled |
| October 3rd | Final UML diagrams completed (Wayne Tolson)  Phase III source code compiled |
| October 10th | Final Product completed  Final Peer review completed |

This timeline served us well, and for the most part we stuck to it. Data parsing took us an additional week to get working in full, but that was one reason we placed it so early on in our desired milestones. We knew it may require additional attention, and addressing it earlier allowed for changes to be made down the road. The GUI changed throughout the weeks as well. This was to accommodate additional features, such as the high score button, as well as to adjust visuals and functionality. Our first set of unit tests were already completed a week ahead of schedule. We continued to add tests as we moved forward and additional parts were added to the program.

**Required Hardware & Software**

Our program is a desktop application compiled in Java 1.8. It was developed using the NetBeans and Eclipse IDE's. The WindowBuilder plug-in was used within Eclipse to assist with GUI development.

**User Guide**

After starting the application, select the "Start a Game" option from the Main Menu to begin playing.

This will open a new window with your first quiz question.

After selecting the answer which you believe to be correct, hit the "Check Answer" button to see if you got it right. If you were incorrect the program will let you what the actual answer was.

Proceed onward by clicking the "Next Question" button, or you can choose to quit by clicking the "End Quiz" button.

During the quiz you can check what your current score is by hitting the "Check Score" button on the main menu.

After answering 10 questions, the quiz will end and alert you of final score.

You can continue to play and the "Check Score" button will track your number of games played, your current score, and your highest scoring game so far from your current session.

When you are done playing, select the "Quit" button on the main menu to end the program.

**Test Scope & IO Expectations**

We used scenarios to test all of our features.

**Scenario 1 by Wayne Tolson: Opening the program and successfully completing a quiz**

1. Corey Jones starts the program by executing the quiz application
2. The program opens and displays the main menu with options to start a quiz and other functions
3. Corey clicks the “Start a Game” button
4. Corey selects the answer he believes to be correct
5. The program checks if Corey was correct
6. The program displays a message stating whether the answer was correct or incorrect, and updates the score if necessary.
7. Corey clicks the “Next Question” button
8. The program, at the end of the quiz, shows the final score and how many answers were correct and incorrect
9. Corey clicks the “Quit” button
10. The program closes completely

**Scenario 2 created by Caroline Ganier: A successful attempt to open the program, take a quiz, exit, and play again**

Detailed Example:

1. Pat Lee Double initiates the execution for the the quiz program.
2. The quiz program opens.
3. The quiz program displays the main menu with options [3 options: ‘Start a Game’, ‘Check Scores’, and ‘Quit’].
4. Pat Lee clicks the ‘Start a Game’ option.
5. The program shows the first question.
6. Pat selects an answer.
7. The program updates the score for this quiz.
8. The program displays a pop up showing status of answer (correct or incorrect, with right answer shown)
9. Pat Lee clicks End Quiz.
10. The program displays the main menu.
11. Pat Lee clicks Quit.
12. The program closes completely.
13. Pat Lee Double initiates the execution for the quiz program.
14. The quiz program opens.
15. The quiz program displays the main menu with options [3 options: ‘Start a Game’, ‘Check Score’, and ‘Quit’].
16. Pat Lee clicks the ‘Start a Game’.
17. The program shows the first question.
18. Pat selects an answer.
19. The program updates the score for this quiz.
20. Pat Lee clicks the 'Next Question' option and continues playing.

High Level Example:

1. Pat starts the game.
2. Pat selects the desired quiz.
3. Pat plays a game.
4. Pat exits the game.
5. Pat decides to play again, and restarts the game.
6. Pat selects a quiz to play.
7. Pat plays more quiz games.

**Scenario 3 created by John Lasheski: Operating the main menu**

Start A New Quiz - User presses the start quiz button.

Program attempts to load a random quiz question from the data file.

Throws exception and displays error message box if unsuccessful.

returns to main menu.

Quit - User presses the Quit button.

Program should exit gracefully.

Check Score- User presses the Check Score button.

Pop up displays current score, number of games played, and high score across all games from this session.

If no games have been played yet, prompt the user to play.

**Test Approach & Practices**

During development we tested very frequently. Each time a code deliverable was made, we ensured it passed all of our tests first. We also looked for bugs and potential issues as we coded to ensure our product runs as clean as possible. By posting our code to github and our questions on wiggio, we were able to work as a team to solve all issues that arose.

Our team will utilized manual and automated tests. The team individually tested the code, as well as using friends and family to simulate end-users who were able to find issues which we had not thought to address. Our unit test collection grew as our code did, and we worked to incorporate automated tests for all new features. We used regression testing as part of our plan to ensure the addition of new code did not negatively impact our previous work.

The only exception we had which needed to be handled was an I/O exception in the case that our file could not be found and read. To handle this, a message box simply pops up to alert the user of the issue.

For our final submission all unit tests ran flawlessly and no bugs were found.

**Design and Alternate Designs**

For our program we utilized the Model-View-Controller (MVC) design.  The MVC set up worked very well for designing our program.  It encouraged breaking up program functionality into 3 different areas, and this separation allowed for cleaner code and also made it easier to separate things logically when working with a group of programmers.

The overall design for our project was modeled after the Model-View-Controller design pattern. The Model component stored important information retrieved by the Controller classes and displayed by the view classes. The Model component is comprised of the Quiz class, which is in charge of handling the quiz format, the Question class, which is in charge of handling the question format, and the QuestionParser class, which is in charge of parsing the questions from a text file. The View component generates a depiction of the data stored within the model classes. This component comprises of classes that is in charge of setting up the look for both the main menu and the quiz itself named MainMenuView and QuizView respectively. The Controller component sends commands to the model to update its state as well as how the program looks. This component includes the Quiz\_Application class, which starts the entire program, and the Quiz\_Controller class, which controls how the quiz is operated.

At its start, the program offers the user three options to choose: start the quiz, display the current and high score, and exit the program. When the quiz starts, the program pulls a question at random from its bank and displays it to the user. They then select which answer they think is correct and the program checks the selected answer and tells them if it is right or wrong. If they would like to exit the quiz, they can click the “End Quiz” button; otherwise they can click the “Next Question” button to continue. Once 10 questions have been asked, the program tells the user their final score and, if they’ve done more than one quiz, if they’ve beaten the high score. To exit the program, simply select the “Quit” button.   
 Originally we had thought to make multiple quizzes, each broken up into a pre-defined 10 questions, each with a focus on a certain dinosaur or aspect of prehistoric life. After discussion within the group, we opted to make a singular quiz which would last for 10 questions and just pull random (non-repeating) questions from our full test bank of questions. This method provides a more varied experience for the user and provides additional replay value.

**Development History**

From the outset of the project we knew that this would be a GUI application that used some logic operations to interact with both the user and the quiz data. Therefore it was decided that we would use the Model View Controller framework for our design. This is a design pattern that had been around for quite a while and there was plenty of documentation on the subject. Our team had all heard of the pattern and knew that it would fill our needs nicely.

Our initial design was sketched out using UML diagramming. This was a good tool for getting ideas on paper and to help find missing links and functionality. Our design consisted of six core classes and an enumeration.

Quiz\_Application – contains main

Question – model object – most basic form of data in the quiz

Quiz – model class used to hold and manipulate the question data for the quiz

Quiz\_Controller – controller class used to talk back and forth between the views and the model

QuizView – a view class used for in game GUI

MainMenuView – a view class for launching quizzes

QuestionTypeEnum – an enumeration to help determine question type

Throughout the development process we found that the controller class and the model objects didn't need much modification from their original design parameters. The exception being the separation of the QuestionParser into its own class. This differs from the view objects. Our initial design of the view classes was purposely vague. It wasn't until we started building out the screens that we knew what buttons would be needed and what helper methods would be needed. These classes changed much from our initial UML blue prints. In addition to the view class changes we of course had small changes to make to the other classes such as helper methods and other variables to help manage state. Given more time, these changes probably could have been foreseen in the design stage, but we are still content that our final product remains faithful to our original design and intent.

A full history of our development can be seen on our github page at [www.github.com/JMLjml/CMSC495/commits/master](file:///C:\Users\Bradley\Desktop\capstone\www.github.com\JMLjml\CMSC495\commits\master)

Our github aliases are as follows:

Joseph Dain- joedain

Caroline Ganier- CarolineGanier

John Lasheski- JMLjml

Wayne Tolson- wtolson1

Bradley Wetzel- bswmagic

**Conclusions**

The Agile Dinosaur Project was a fun and interesting project to complete. This was my first time acting as a project manager, which meant I dealt with a lot of new experiences. I had to make choices that would affect the project's outcome as a whole, and in turn would affect my group members. I learned that an open line of communication is key. Not being afraid to ask questions and brainstorming as a group was very helpful to us, and I'm glad we were able to meet on a regular basis to keep on track with on progress. Overall our group had a very positive vibe and that definitely helped with completing our work and acting as a cohesive unit. A design strength of our chosen MVC design pattern was that it simplified tasking out work to multiple programmers. With how our work was divided it generally meant that only one person would be accessing a certain set of files at a time so there were not many version control issues.

Paleontology is a side interest of mine, and in all likelihood, I will continue to expand and improve this project in my spare time. This could mean a variety of improvements such as the inclusion of pictures, an expansion of support for more question types, different difficulty levels, and of course a lot more quiz questions. One aspect I would really love to add would be more in depth feedback on the questions when you check for the correct answer. For example if the question was "Dinosaur means '\_\_\_\_\_ lizard' in Greek." instead of just saying "The correct answer was 'Terrible'." I'd like it to expand upon this and say something more along the lines of "(Correct/Incorrect), Dinosaur means 'Terrible Lizard' in Greek. When the term was coined, 'terrible' was used more commonly as a term to mean 'fearfully great' than today's meaning of 'really bad'." A change such as this would enhance the learning feel of the game and help the user to learn and remember these facts.

Overall, I had a wonderful time developing this project with my group, and I feel we all learned a lot through the development process. It was a new experience, and one which I really enjoyed.