What did you build during this phase?

We have implemented our main game screen consisting of the user input console, the JTextArea nested in a JScrollArea that will contain the questions we ask the player, and the graphical game area that displays the enemies. The graphical area consists of our GameArea class that extends the JPanel class, allowing us to draw onto the JPanel as needed.

Our current implementation of the working aspects of the game can be found here:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/Game.java
The game window that assembles the component pieces together can be found here:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/userInterface/Window.java
The graphical game area that is an extension of JPanel can be found here:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/GameArea.java

We have implemented the Enemy class which contains the image as well as coordinates for the enemies our players will face.
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/Enemy.java

We have implemented a QuestionCreator class that reads from our XML file database, pulling out a question (either randomly or by difficulty) and parses it into a Question object.
QuestionCreator class:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/QuestionCreator.java

We have created a Question interface and have coded some temporary classes that implement it.
Question interface:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/Question.java
Temporary question classes:
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/ReturnValueQuestion.java
https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/backEnd/TrueFalseQuestion.java

We have created an inputMatcher class that checks the answers that the player provides against the answers to the questions that are pending answering in the question area. This class also keeps track of and updates the player's score. Also contains a HashSet of all the questions that have been spawned.

https://github.com/csc301-winter-2016/project-team10/blob/master/301Project/src/userInterface/inputMatcher.java

High-level design of your software.

The game applet mainly uses JTextArea, JScrollArea and JPanel to display the game area.

Graphical representations for game sprites are shown by extending JPanel to paint the sprites that we want onto the screen.

The game itself is spiritually reminiscent of Space Invaders but instead of moving around and shooting the enemies, they will be defeated by answering questions related to programming.

We will need a player object which will keep the values of the players remaining health and score.

The enemies will be an extension of MovingSprite superclass that implements the Sprite interface. Each enemy object contains its image (using BufferedImage) and its x-, y-coordinate values.

We have a Question class and a QuestionCreator class: the QuestionCreator reads from an XML file to pull questions and answers randomly, then converts them into Question objects. Each row of the xml file will pertain to one question, having the question string, the answer string and the difficulty value of the question. We can then base the scoring for the question off the difficulty value. The time allowance to answer the question can also be scaled off of the difficulty value.

We will have a JLabel score board in the game area panel.

We will have walls to buffer against the enemies (we will implement this at a later time)

The player will input answers into a JTextArea, which contains a KeyListener object to handle the player's input. The input is then passed off to an inputMatcher object which compares the player's answer to the answers of active questions.

We may implement a pause function for the game.

Technical highlights: interesting bugs, challenges, lessons learned, observations, etc.

We faced a few issues with the GitHub repo, we broke the project while trying to solve conflicts; this threw off the project until we reverted to a previous version and then merged the code we wanted again.

We had issues with conflicts as people did not or forgot to update from the upstream master repo, so we will have to be careful about keeping our local repositories up to date before we try to commit changes.

Learning how to draw an image and move it on the game screen was fun but also time consuming.

Making the JScrollArea automatically scroll took some time to decide what implementation to use: there was a number of ways to implement it, each with different benefits and limitations.

One interesting challenge we faced was how to format the main game screen using Swing. With these libraries we could have used Grid, Box, Card, and Flow layouts each with with their own respective advantages and disadvantages. For example: FlowLayout is often used for inline Swing panels and elements. These different layout managers were all very difficult to implement and time consuming to learn. Moreover there were many complexities involved in learning these formats ie. different functions and variables to position elements. We eventually settled on using GroupLayout. Furthermore, we learned possible ways to use LayoutManagers for future iterations of our project and which ones specifically work well for different purposes.

Another technical aspect we learned was the similarity between JTextArea and JTextPane. The two are very similar to each other in functionality however it was learned that the latter had extensions that would be very useful for future iterations of our project (functions to colour and manipulate text).

What are your plans for the next phase?

We want to change the JTextAreas to JTextPanes so that we can take advantage of the greater functionality of the JTextPane.

We will want to implement the protective walls and implement the interaction between enemies, the wall sections and the player sprite. We want to implement the functionality for synchronizing question-spawning with enemy-spawning as well. This means we will have to have the player object fully implemented as well.

We will want to finish full functionality for our inputMatcher class so that it operates as described previously.

We will add more questions to our XML file.

Finish implementation of Question interface and the classes that implement it.

We will add tutorials to cover the questions material.