COP 5339 Object Oriented Software Design Project
Design Specifications (Revised)

Team #6: Alejandro Reyes, Leigh Chin, and Tsz Shing Tsoi

Project Title: Exciting Trivia Game

Important Details: This offline application is a single player game that will run on Java Virtual Machine. It will be developed using Java Swing Library. Additional two players are simulated (not real players).

Project Source Codes: See Attachment B.

Application Requirements - Updated

Functional Specification with GUI[1]

Upon launching the application, the user will get a splash screen with "Welcome to Exciting Trivia Game" message. Then a game menu will display a text field to indicate where the user will enter his or her name and a dropdown menu for the user to pick a difficulty level (novice, intermediate or expert). Default will be novice. The game menu will have four additional buttons: Start, View Scores, Reset and Exit. The Start, View Scores and Reset buttons will initially be deactivated until the user's name is provided. Exit will close the application, View Scores will launch a screen with the user's top 10 saved scores and the corresponding ranking, and Reset will erase all user data and scores. Start continues to begin the game and opens the game screen.

On the upper left corner of the game screen, a message will display in a text area, "Tom has entered the game. Jane has entered the game." to make the user aware of the other simulated players. On the upper right corner will be a tally marking the points earned by each player in a text area. On the center of the screen will be a read only large text area that will display a question followed by some choices a, b, c or d. Under the text area, there will be six buttons. One button for each choice of the question, a next question button and an end game button. The user can end the game at any time. The next button will display a trivia question followed by the possible choices. Once the next button is clicked, the button will be disabled until someone gets the answer or everyone has answered incorrectly. Each player can only choose once. The first player to answer correctly gets the point. A message will display on the large text area on the center with the order of the answers submitted by each player and the correct answer. At this point, the next question button is enabled. Once clicked, the large text area will display the next question.

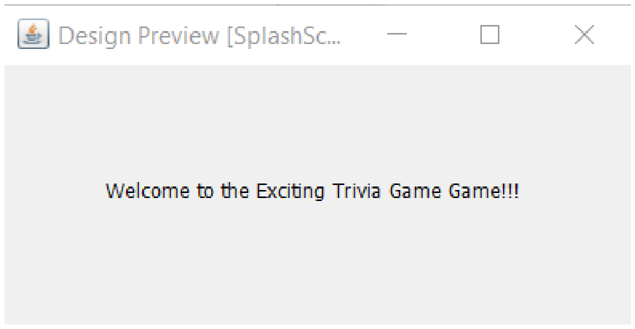
Once the user clicks the end game button, the end game screen will appear with the scores and ranking. This screen will have one Close button to close the end game screen and return back to the game menu. If the user plays long enough to answer all the questions, the game will end, and the end game screen will appear with the scores and ranking.

[1] Initial functional specifications and use cases are included in Attachment A.

Use Cases with GUI[1]

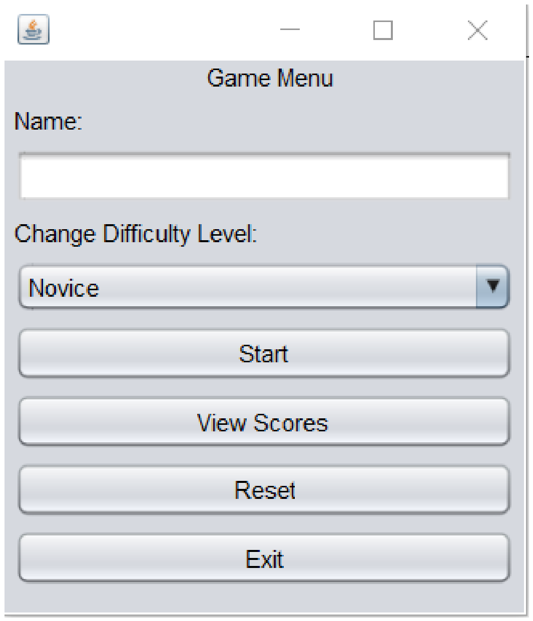
UC1. Launch Application

1. User executes the application.
2. System displays a splash screen with the following message:



Welcome to the Exciting Trivia Game Game!!!

3. After the splash screen ends, system displays the game menu with the following items:



4. User enters his or her name in the text field.
5. User clicks the dropdown menu underneath the “Change Difficulty Level” label.
6. System displays the following list of options:

```
Novice
Intermediate
Advanced
```

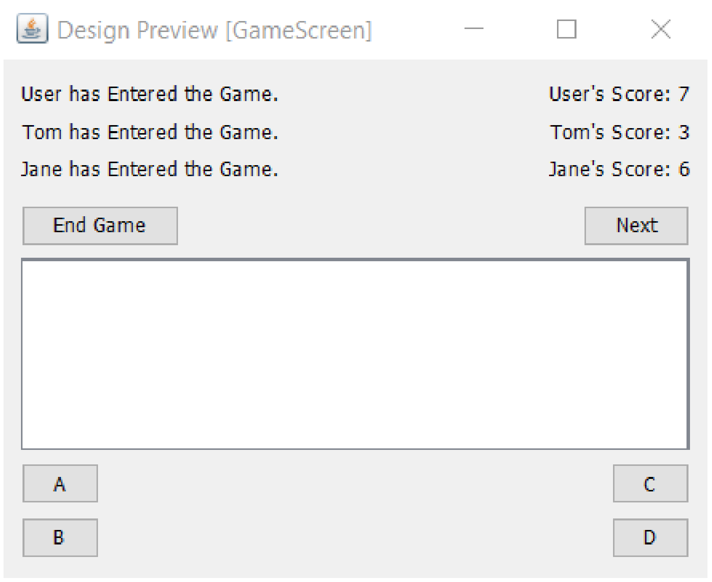
7. User selects “Novice” from the list of options.
8. All game menu buttons are activated.

UC1. Launch Application – Variation #1

1.1. Start at **UC1. Launch Application** Step 2.
1.2. System detects an existing user's name and loads the user data. Splash Screen ends.
1.3. Continue at **UC1. Launch Application** Step 8.

UC2. Start a New Game

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User clicks the "Start" button.
3. System displays the game screen as illustrated below:



Design Preview [GameScreen]

User has Entered the Game. User's Score: 7
Tom has Entered the Game. Tom's Score: 3
Jane has Entered the Game. Jane's Score: 6

4. System displays players joining the game in a text area in sequence on the upper left corner of the game screen.
5. System displays the current scores of each player on the upper right corner of the game screen in a text area.
6. System displays the end game button and next button above the text area in the center of the screen, and four buttons for choices a, b, c and d below the text area in the center of the screen.
7. System displays the following message in a text area in the center of the screen:

```
Welcome! Please click "Next" to begin the Exciting Game!
```

UC3. Scoring a Point

1. User carries out **UC2. Start a New Game**.
2. User clicks the "Next" button.
3. System displays a new question in a text area following by four answer choices a, b, c or d.
4. User clicks on any of the buttons {a, b, c, d} to select the answer to the question.
5. After all players select an answer or 10 seconds have elapsed, the system will display the answers in a text area picked by each player in descending order of quickest to answer along with the correct answer.
6. System will increment the score for the winner of the round and display the updated score on the upper right corner.

UC4. End the Game

1. User carries out **UC3. Scoring a Point**.
2. User clicks on the “End” button.
3. System displays the end game screen showing each player and their scores and ranking, and a “Back to Game Menu” button.
4. User clicks the “Back to Game Menu” button.
5. System displays the game menu.

UC4. End the Game – Variation #1:

1.1. Start at **UC4. End the Game** Step 1.
1.2. User clicks Next button until all questions are exhausted.
1.3. Continue at **UC4. End the Game** Step 3.

UC5. Reset User Data

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User clicks the “Reset” buttons.
3. System displays the following message:

```
All user data and scores will be erased.
Continue? Yes / No
```

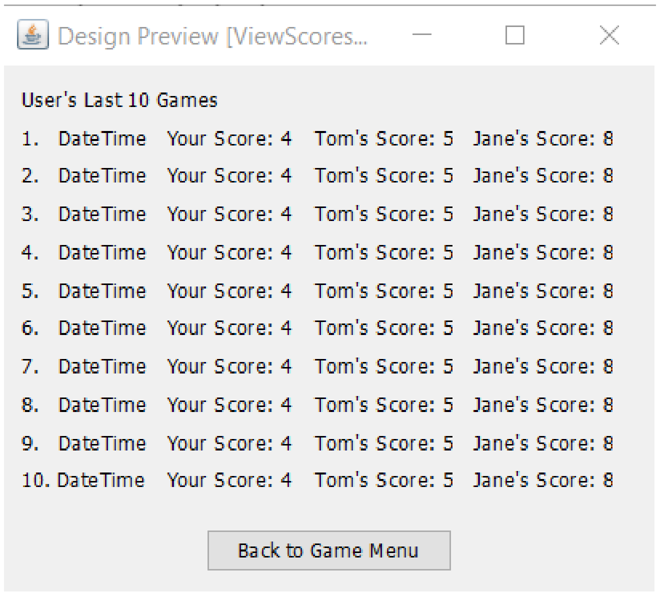
4. User clicks the “Yes” button.
5. System erases all user data and scores and displays the game menu.

UC5. Reset User Data – Variation #1

1.1. Start at **UC5. Reset User Data** Step 3.
1.2. User selects “No”.
1.3. System displays the game menu.

UC6. View Scores

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User clicks the “View Scores” button.
3. System displays the view score screen showing the top 10 saved scores and rankings, follow by a “Back to Game Menu” button:



User's Last 10 Games

1.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
2.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
3.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
4.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
5.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
6.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
7.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
8.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
9.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8
10.	DateTime	Your Score: 4	Tom's Score: 5	Jane's Score: 8

4. User clicks the “Back to Game Menu” button.
5. System displays the game menu.

UC7. Change Difficulty Level

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User selects the dropdown menu underneath the “Change Difficulty Level” label.
3. System displays the following list of options:

```
Novice
Intermediate
Advanced
```

4. User selects a new difficult level from the list of options.
5. The difficult level is changed.
6. System displays the game menu.

UC8. Exit the Application

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User clicks the “Exit” button.
3. The application closes.

Main Actors

1. User
2. System

Glossary

End Game Button: A button used to display scores and stop playing the game.

End Game Screen: A screen where the user can see the score after he or she finished the game and return to the Game Menu. This screen is similar to the View Scores Screen except the current scores are shown.

Exit Button: A button to exit the application.

Game Menu: A set up menu where the user can enter his or her name, view recent scores, clear the history or select a difficulty level and begin a new game.

Game Screen: A screen where the user can play the game. The user can answer a trivia question and score a point.

Next Button: A button used to display the next question in the game playing session.

Reset Button: A button to clear the users game history.

Score Points: The number of questions answered correctly.

Splash Screen: The initial screen shown to the user when the application is first launched.

Simulated Player: A system player. The system will act as a competitor and try to make the game challenging for the user.

Start Button: A button used to begin playing a new game.

System: The Exciting Trivia Game application.

User: A real player or human who is playing the Exciting Trivia Game.

View Scores Button: Displays a new screen with the user's recent scores.

View Scores Screen: The screen to display the user's recent scores.

CRC Cards

Note: Classes from Java APIs are not included. Please see Java documentation and class diagrams for details of these classes from Java APIs.

Domain Classes

Class: GameSession
Responsibilities:
- Manage application states
- Manages UI screens

Collaborators: GameSystem, Game, SplashScreen, GameMenu, GameScreen, ViewScoreScreen, JPanel, JFrame

Class: GameSystem
Responsibilities:
- Manage application
- Manage user data
- Manage question list
- Start a game

Collaborators: Game, Player, QuestionList, DataSaver, DataLoader

Class: Game
Responsibilities:
Manage states of a game play
Manage data (e.g. scores, winners) during a game play
Collaborators: Judge, Player, Timer, Question, Iterator

Class: DataSaver
Responsibilities:
Save player data to a file
Collaborators: Player

Class: DataLoader
Responsibilities:
Load player data from a file
Load questions from a file
Collaborators: Player, Question

Class: QuestionList
Responsibilities:
Manage questions
Collaborators: Question, Iterator

Class: Question
Responsibilities:
Maintain a question and answers
Collaborators: None

Class: Player
Responsibilities:
Manage player information (e.g. name, difficulty, scores)
Collaborators: None

Class: SimulatedPlayer
Responsibilities:
A Player but answers are generated based on correct answer
Collaborators: None

Class: Judge
Responsibilities:
Keep track of answers from Player
Determine winner of the current round
Collaborators: Player

UI Classes

Class: GameMenu
Responsibilities:
Hold a JTextField to enter users name for score keeping.
Hold a Start JButton to begin a new game.
Hold a View score JButton to see score history.
Hold a Reset JButton to clear the scores.
Hold an Exit JButton to leave the game.
Collaborators: JPanel, JButton, JLabel, JTextField, JComboBox

Class: SplashScreen
Responsibilities:
Display a welcome message to the user using a JLabel.
Collaborators: JPanel, JLabel

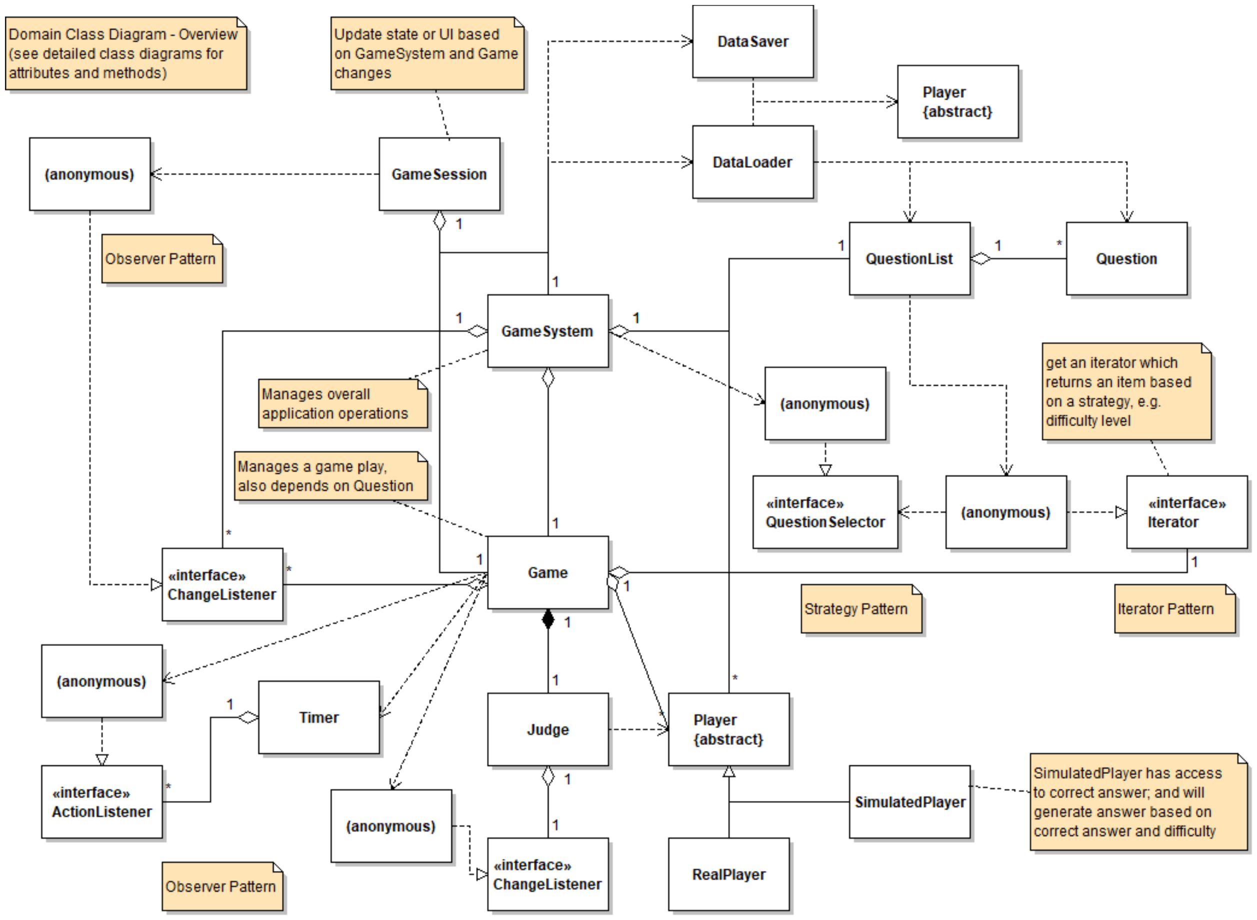
Class: GameScreen
Responsibilities:
Display a question in a JTextArea.
Display the results of a round in JTextArea.
Display current scores for each player in a JLabel.
Introduce players at start of the game in a JLabel.
Display four JButton for each answer choice available (a, b, c, d).
Collaborators: JPanel, JButton, JLabel, JTextArea

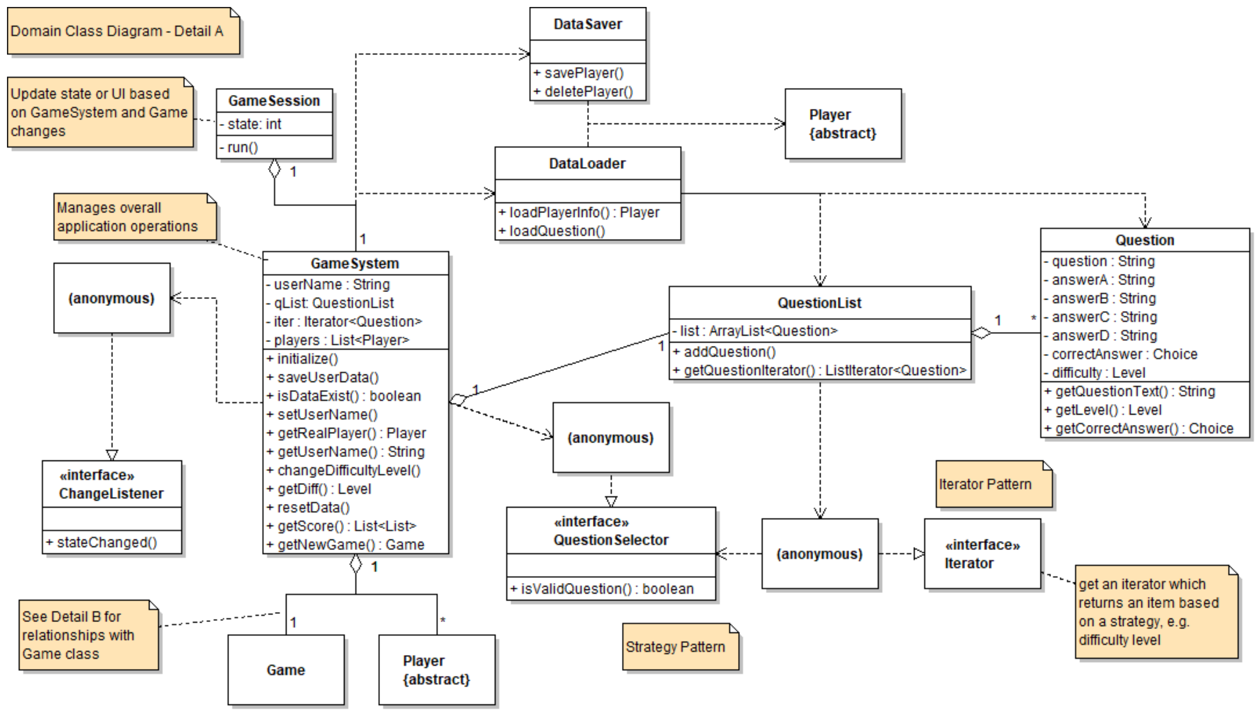
Class: ViewScore
Responsibilities:
Display the results of the current game in a JLabel.
Display a history of games in a collection of JLabels.
Hold a JButton to return the user to the MenuGame Screen.
Collaborators: JPanel, JButton, JLabel

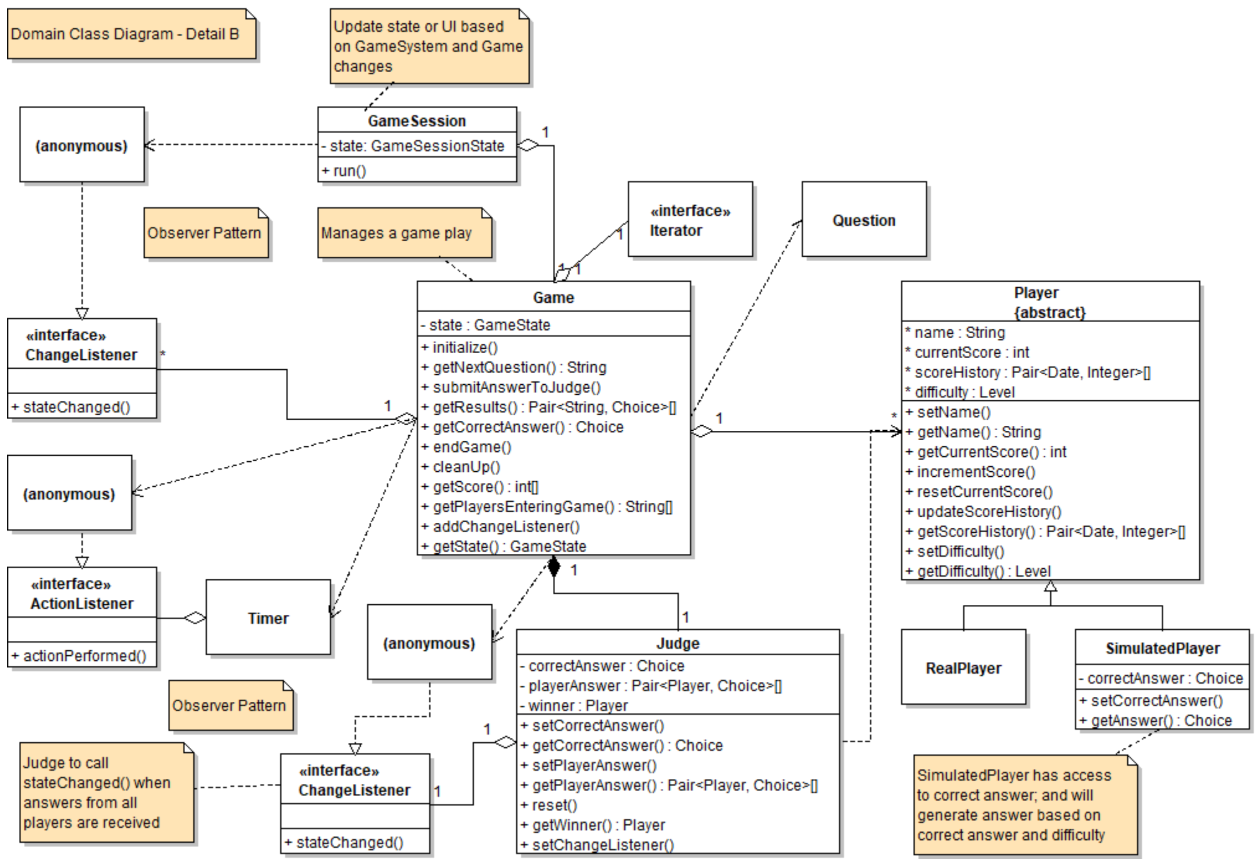
UML Diagrams

Class Diagrams

Domain Class Diagrams


Domain Class Diagram - Overview (see detailed class diagrams for attributes and methods)
Update state or UI based on GameSystem and Game changes
DataSaver
Player {abstract}
DataLoader
(anonymous)
GameSession
Observer Pattern
QuestionList
Question
GameSystem
Manages overall application operations
(anonymous)
get an iterator which returns an item based on a strategy, e.g. difficulty level
Manages a game play, also depends on Question
«interface» QuestionSelector
(anonymous)
«interface» Iterator
«interface» ChangeListener
Game
Strategy Pattern
Iterator Pattern
(anonymous)
Timer
Judge
Player {abstract}
«interface» ActionListener
(anonymous)
«interface» ChangeListener
SimulatedPlayer
SimulatedPlayer has access to correct answer, and will generate answer based on correct answer and difficulty
RealPlayer
Observer Pattern


Domain Class Diagram - Detail A
Update state or UI based on GameSystem and Game changes
GameSession
- state: int
- run()
DataSaver
+ savePlayer()
+ deletePlayer()
Player {abstract}
DataLoader
+ loadPlayerInfo() : Player
+ loadQuestion()
Manages overall application operations
GameSystem
- userName : String
- qList: QuestionList
- iter : Iterator<Question>
- players : List<Player>
+ initialize()
+ saveUserData()
+ isDataExist() : boolean
+ setUserName()
+ getRealPlayer() : Player
+ getUserName() : String
+ changeDifficultyLevel()
+ getDiff() : Level
+ resetData()
+ getScore() : List<List>
+ getNewGame() : Game
(anonymous)
«interface» ChangeListener
+ stateChanged()
QuestionList
- list : ArrayList<Question>
+ addQuestion()
+ getQuestionIterator() : ListIterator<Question>
Question
- question : String
- answerA : String
- answerB : String
- answerC : String
- answerD : String
- correctAnswer : Choice
- difficulty : Level
+ getQuestionText() : String
+ getLevel() : Level
+ getCorrectAnswer() : Choice
(anonymous)
«interface» QuestionSelector
+ isValidQuestion() : boolean
(anonymous)
«interface» Iterator
Iterator Pattern
get an iterator which returns an item based on a strategy, e.g. difficulty level
Strategy Pattern
See Detail B for relationships with Game class
Game
Player {abstract}
1
1
1
1
1
*
1
1
*


Domain Class Diagram - Detail B
Update state or UI based on GameSystem and Game changes
GameSession
- state: GameSessionState
+ run()
(anonymous)
Observer Pattern
Manages a game play
«interface» Iterator
Question
«interface» ChangeListener
+ stateChanged()
Game
- state : GameState
+ initialize()
+ getNextQuestion() : String
+ submitAnswerToJudge()
+ getResults() : Pair<String, Choice>[]
+ getCorrectAnswer() : Choice
+ endGame()
+ cleanUp()
+ getScore() : int[]
+ getPlayersEnteringGame() : String[]
+ addChangeListener()
+ getState() : GameState
Player {abstract}
* name : String
* currentScore : int
* scoreHistory : Pair<Date, Integer>[]
* difficulty : Level
+ setName()
+ getName() : String
+ getCurrentScore() : int
+ incrementScore()
+ resetCurrentScore()
+ updateScoreHistory()
+ getScoreHistory() : Pair<Date, Integer>[]
+ setDifficulty()
+ getDifficulty() : Level
(anonymous)
«interface» ActionListener
+ actionPerformed()
Timer
(anonymous)
Judge
- correctAnswer : Choice
- playerAnswer : Pair<Player, Choice>[]
- winner : Player
+ setCorrectAnswer()
+ getCorrectAnswer() : Choice
+ setPlayerAnswer()
+ getPlayerAnswer() : Pair<Player, Choice>[]
+ reset()
+ getWinner() : Player
+ setChangeListener()
RealPlayer
SimulatedPlayer
- correctAnswer : Choice
+ setCorrectAnswer()
+ getAnswer() : Choice
Observer Pattern
Judge to call stateChanged() when answers from all players are received
«interface» ChangeListener
+ stateChanged()
SimulatedPlayer has access to correct answer; and will generate answer based on correct answer and difficulty
1
1
1
*
1
1
*
1
1
1
1
1

Pattern Mapping

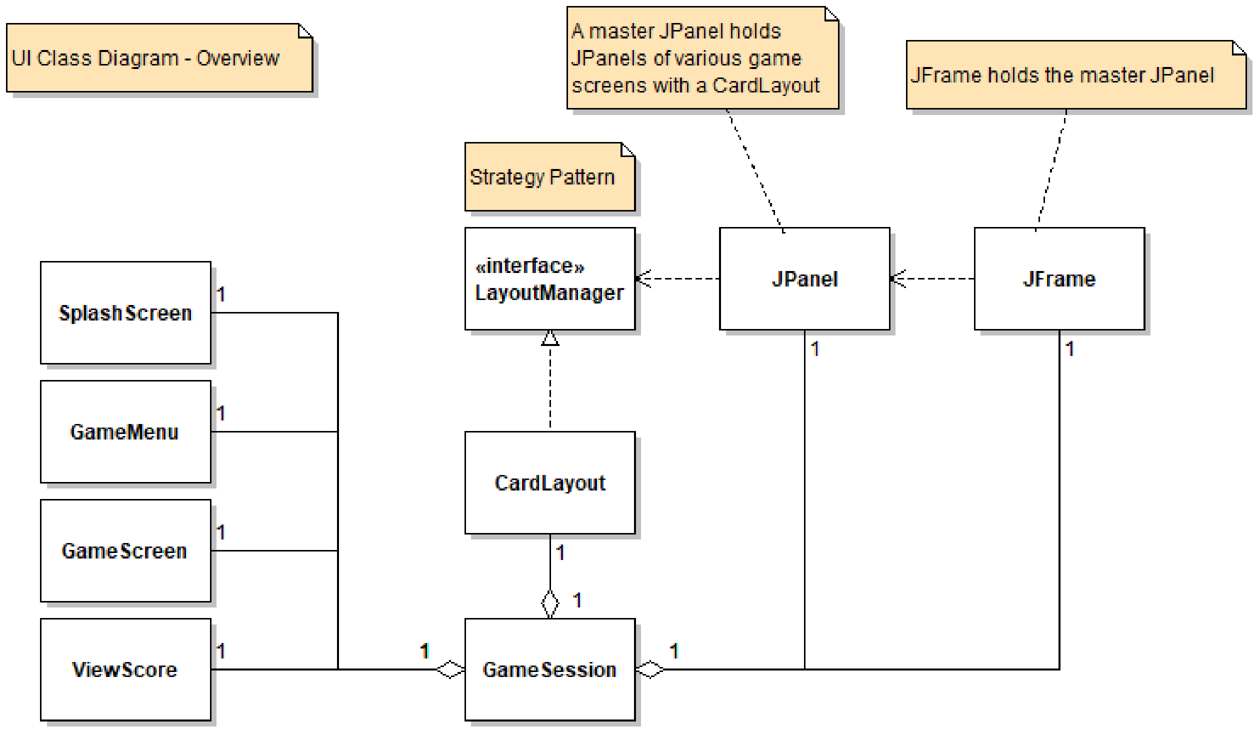
Observer Pattern (GameSystem/Game/Judge)	
Name in Design Pattern	**Actual Name**
Subject	GameSystem, Game, Judge
Observer	ChangeListener
ConcreteObserver	Anonymous class that implements the ChangeListener interface type
attach()	addChangeListener()
notify()	stateChanged()

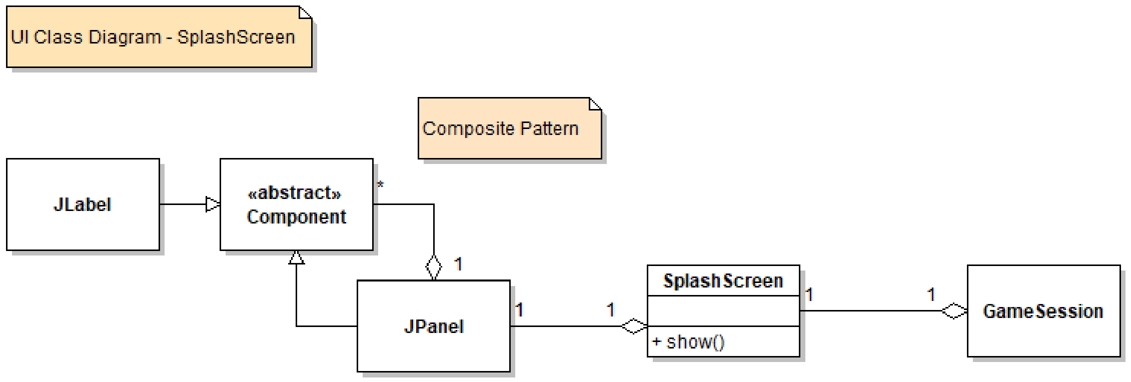
Observer Pattern (Timer)	
Name in Design Pattern	**Actual Name**
Subject	Timer
Observer	ActionListener
ConcreteObserver	Anonymous class that implements the ActionListener interface type
attach()	addActionListener()
notify()	actionPerformed()

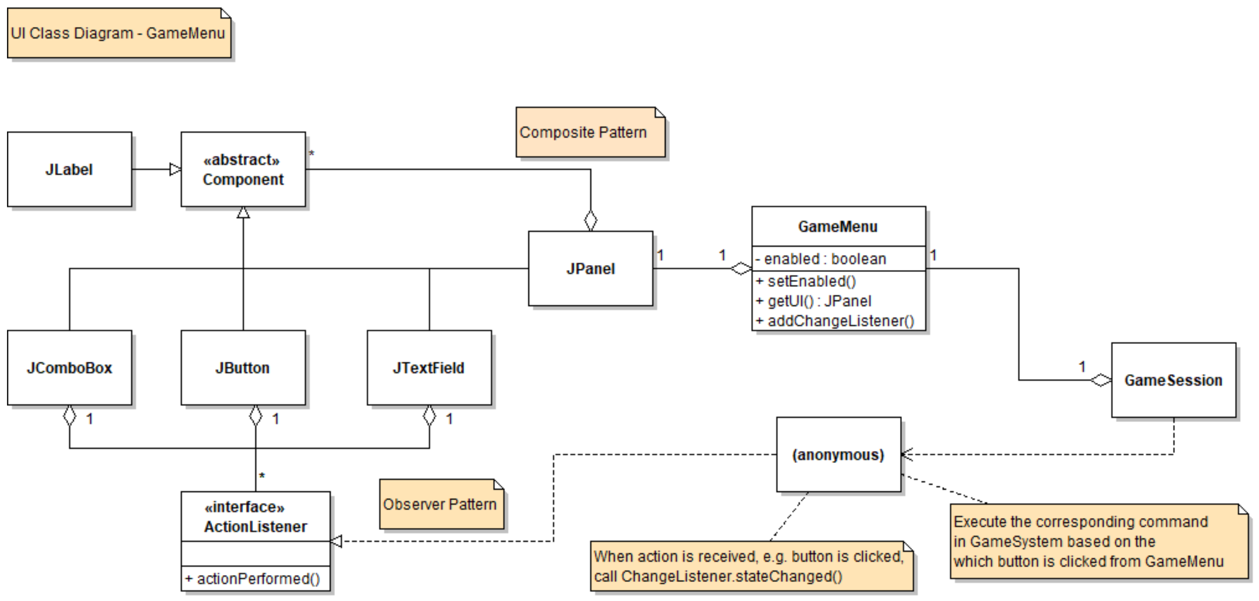
Iterator Pattern (QuestionList)	
Name in Design Pattern	**Actual Name**
ConcreteAggregate	QuestionList
Iterator	Iterator
ConcreteIterator	Anonymous class implementing Iterator
createIterator()	getQuestionIterator()
next()	next()
isDone()	Opposite of hasNext()
currentItem()	return value of hasNext()

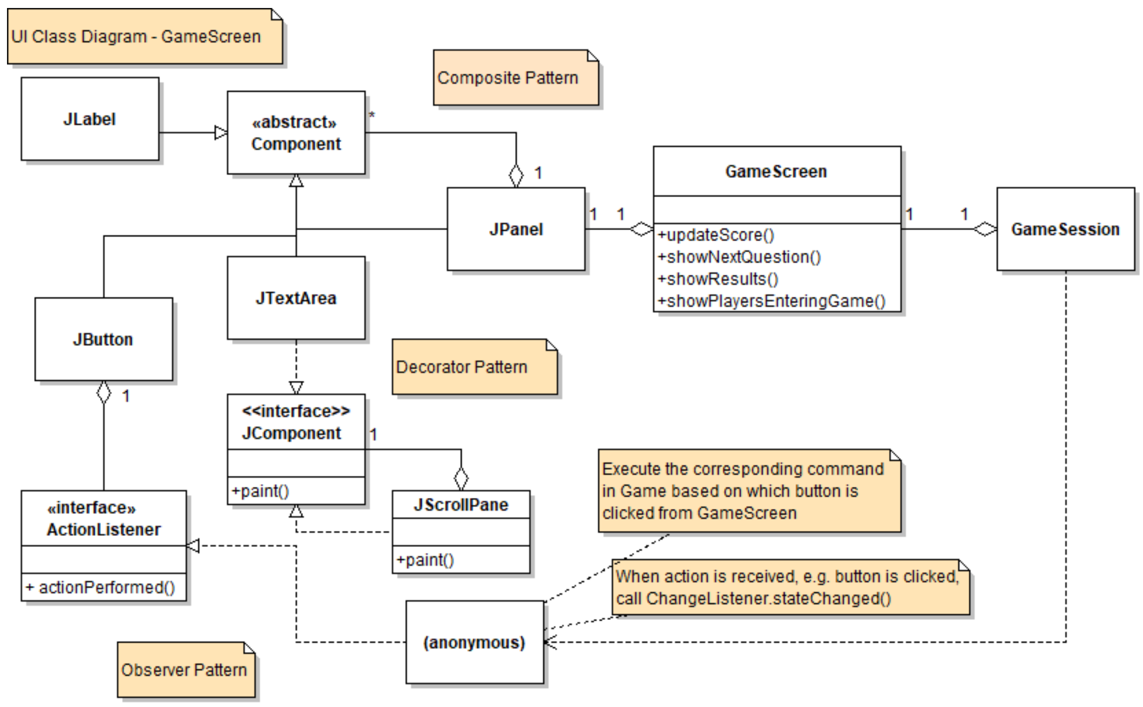
Strategy Pattern (QuestionSelector)	
Name in Design Pattern	**Actual Name**
Context	Anonymous class implementing Iterator and using QuestionSelector to determine the next item
Strategy	QuestionSelector
ConcreteStrategy	Anonymous class implementing the QuestionSelector interface
doWork()	IsValidQuestion()

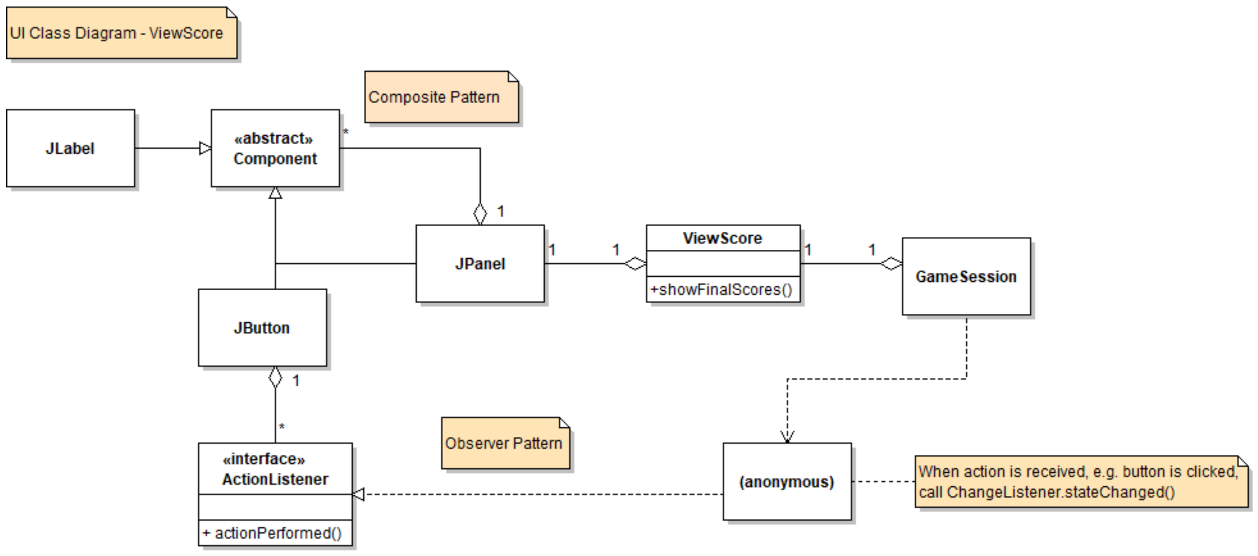
UI Class Diagrams


UI Class Diagram - Overview
A master JPanel holds JPanels of various game screens with a CardLayout
JFrame holds the master JPanel
Strategy Pattern
«interface» LayoutManager
JPanel
JFrame
SplashScreen
GameMenu
GameScreen
ViewScore
CardLayout
GameSession
1
1
1
1
1
1
1
1
1
1




UI Class Diagram - GameMenu
Composite Pattern
JLabel
«abstract» Component
*
JPanel
1
1
GameMenu
- enabled : boolean
+ setEnabled()
+ getUI() : JPanel
+ addChangeListener()
1
1
GameSession
JComboBox
JButton
JTextField
1
1
1
*
«interface» ActionListener
+ actionPerformed()
Observer Pattern
(anonymous)
When action is received, e.g. button is clicked, call ChangeListener.stateChanged()
Execute the corresponding command in GameSystem based on the which button is clicked from GameMenu


UI Class Diagram - GameScreen
Composite Pattern
JLabel
«abstract» Component
*
1
JPanel
1
1
GameScreen
+updateScore()
+showNextQuestion()
+showResults()
+showPlayersEnteringGame()
1
1
GameSession
JTextArea
JButton
1
Decorator Pattern
<<interface>> JComponent
+paint()
1
JScrollPane
+paint()
«interface» ActionListener
+ actionPerformed()
Execute the corresponding command in Game based on which button is clicked from GameScreen
When action is received, e.g. button is clicked, call ChangeListener.stateChanged()
(anonymous)
Observer Pattern



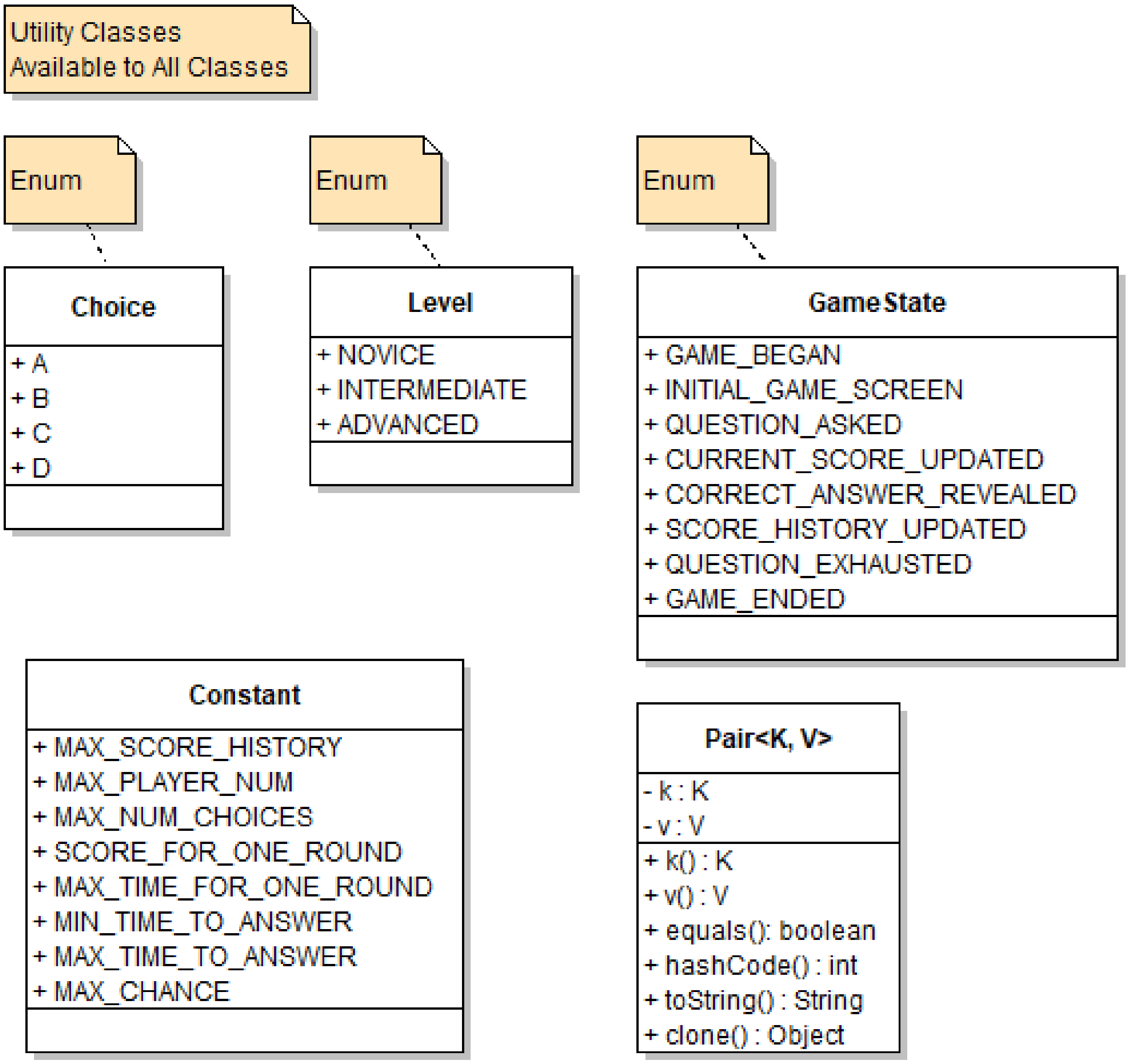
Pattern Mapping

Observer Pattern (JComponent)	
Name in Design Pattern	**Actual Name**
Subject	JButton, JTextField, JComboBox
Observer	ActionListener
ConcreteObserver	Anonymous class that implements the ActionListener interface type
attach()	addActionListener()
notify()	actionPerformed()

Decorator Pattern	
Name in Design Pattern	**Actual Name**
Component	JComponent
ConcreteComponent	JTextArea
Decorator	JScrollPane
Method()	Paint()

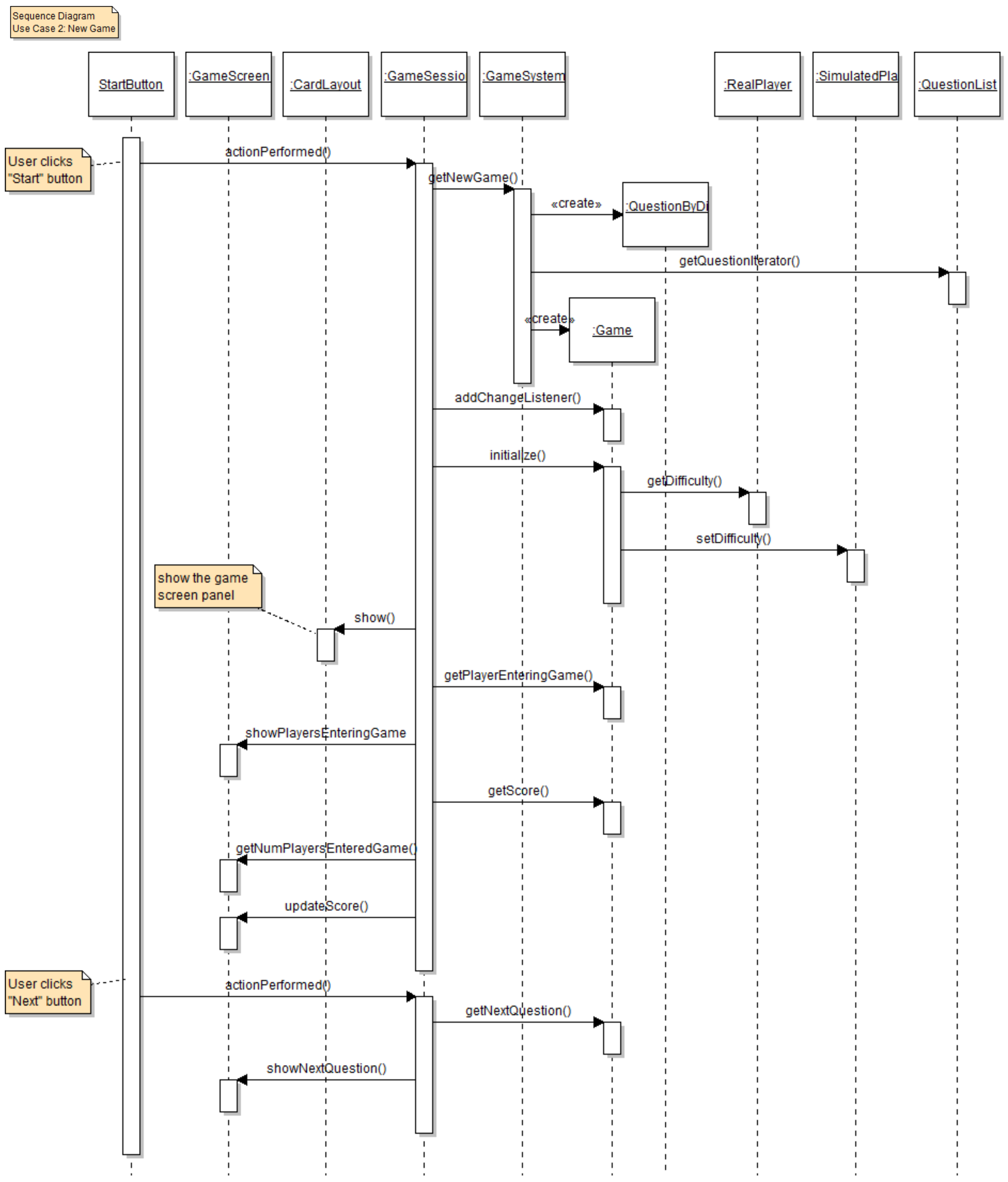
Composite Pattern	
Name in Design Pattern	**Actual Name**
Primitive	Component
Composite	JPanel
Leaf	JButton, JTextField, JComboBox, JTextArea
method()	getPreferredSize()

Other Class Diagram

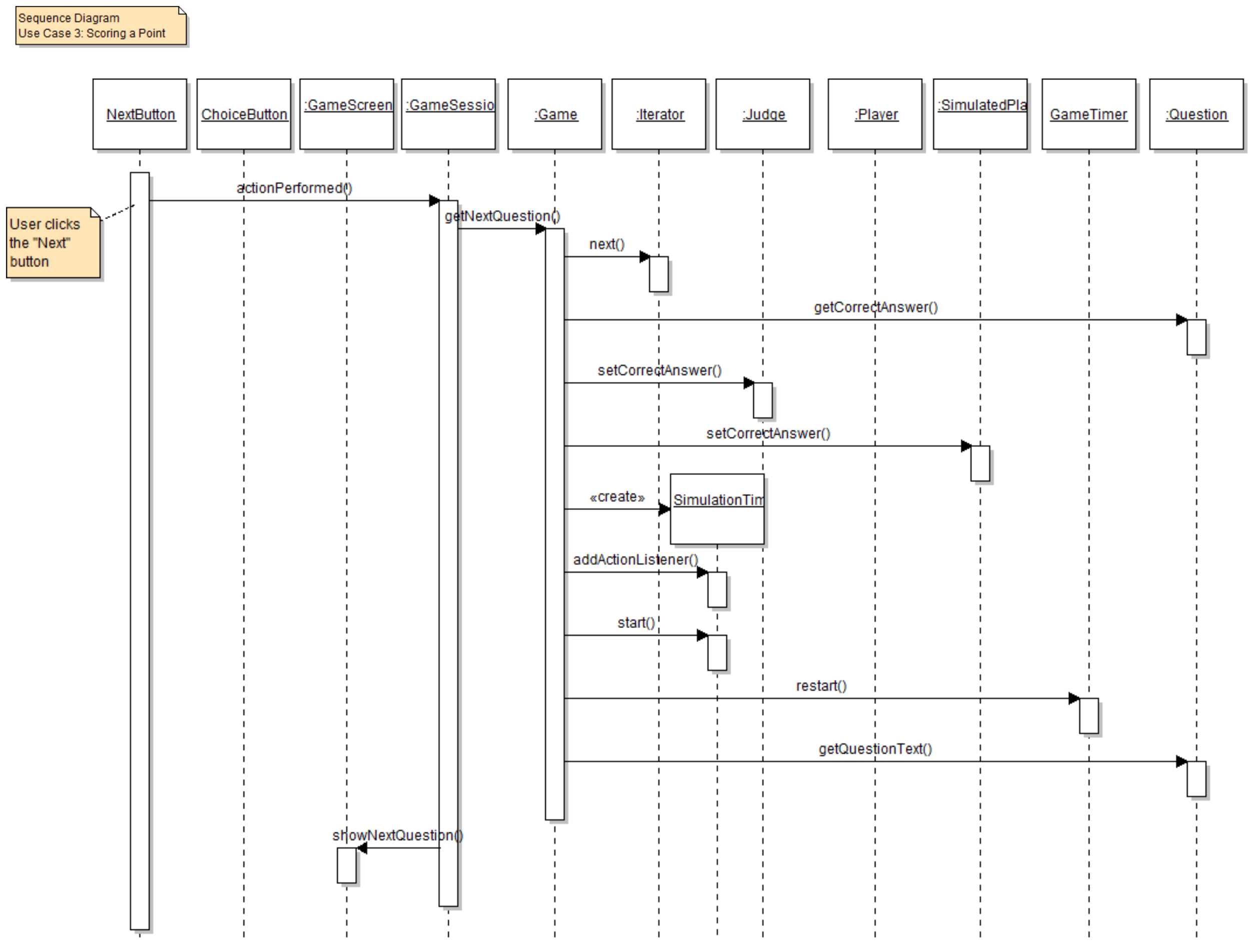


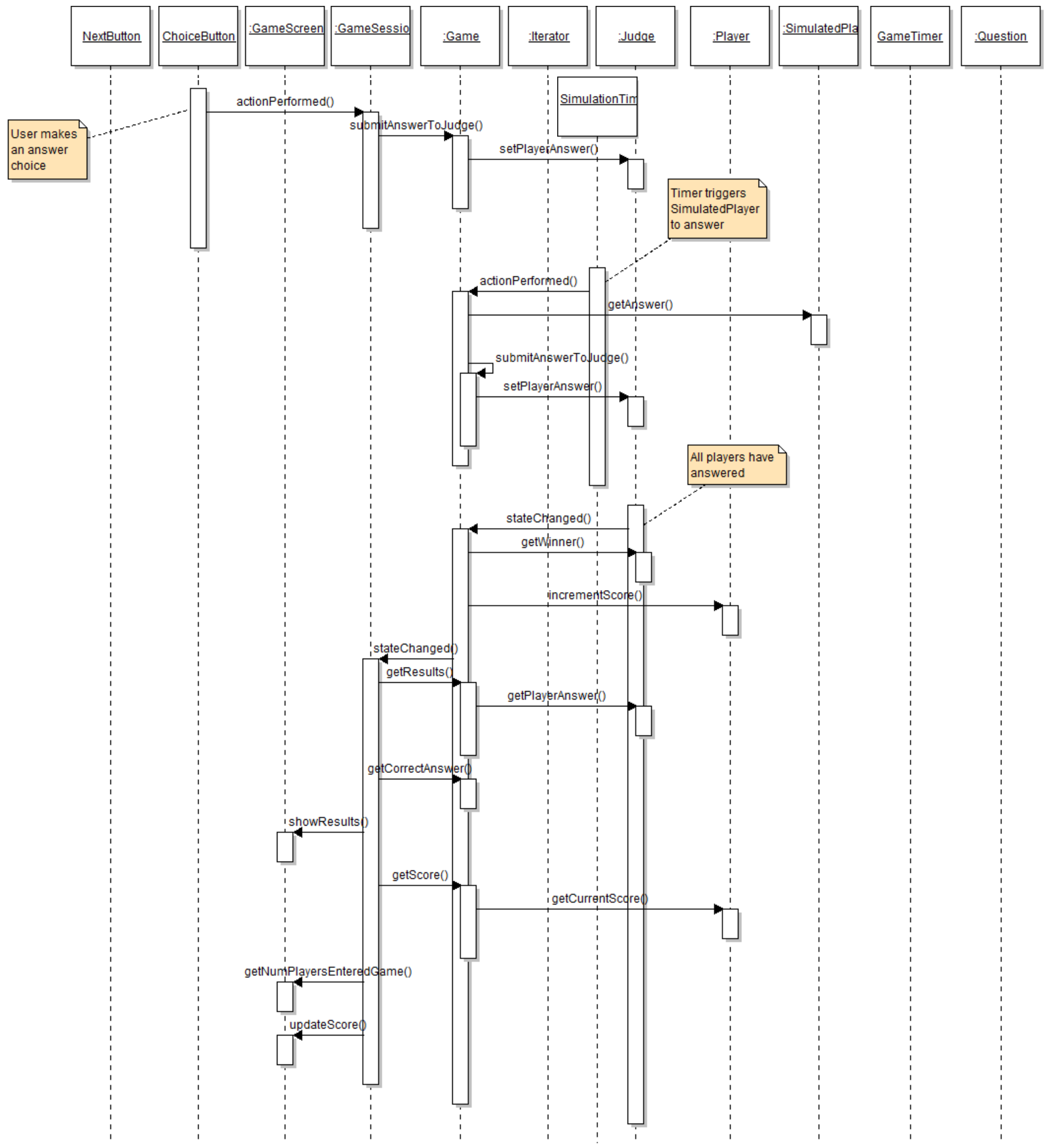
Sequence Diagrams

UC2. Start a New Game

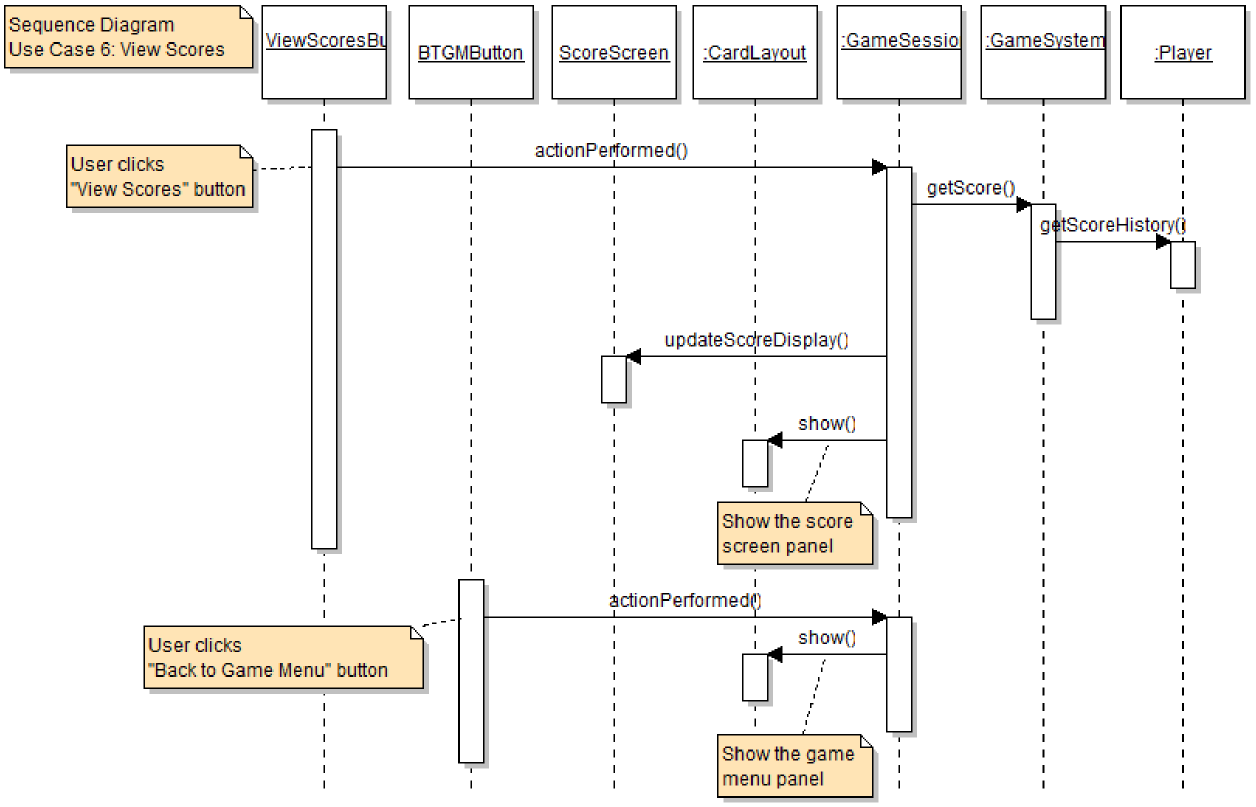


UC3. Scoring a Point

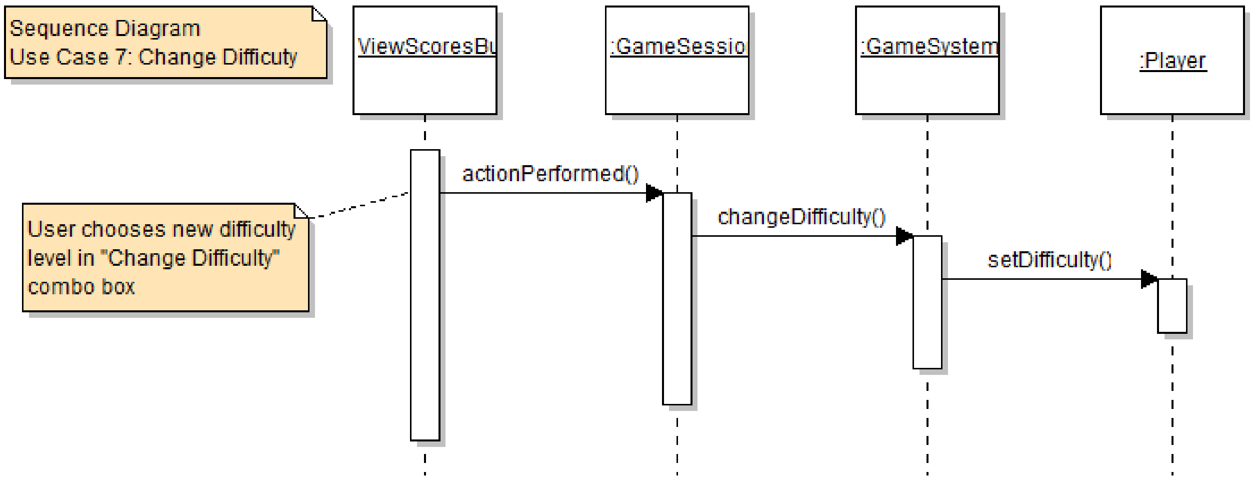



NextButton
ChoiceButton
:GameScreen
:GameSessio
:Game
:Iterator
:Judge
:Player
:SimulatedPla
GameTimer
:Question
SimulationTim
User makes an answer choice
actionPerformed()
submitAnswerToJudge()
setPlayerAnswer()
Timer triggers SimulatedPlayer to answer
actionPerformed()
getAnswer()
submitAnswerToJudge()
setPlayerAnswer()
All players have answered
stateChanged()
getWinner()
incrementScore()
stateChanged()
getResults()
getPlayerAnswer()
getCorrectAnswer()
showResults()
getScore()
getCurrentScore()
getNumPlayersEnteredGame()
updateScore()

UC6. View Scores

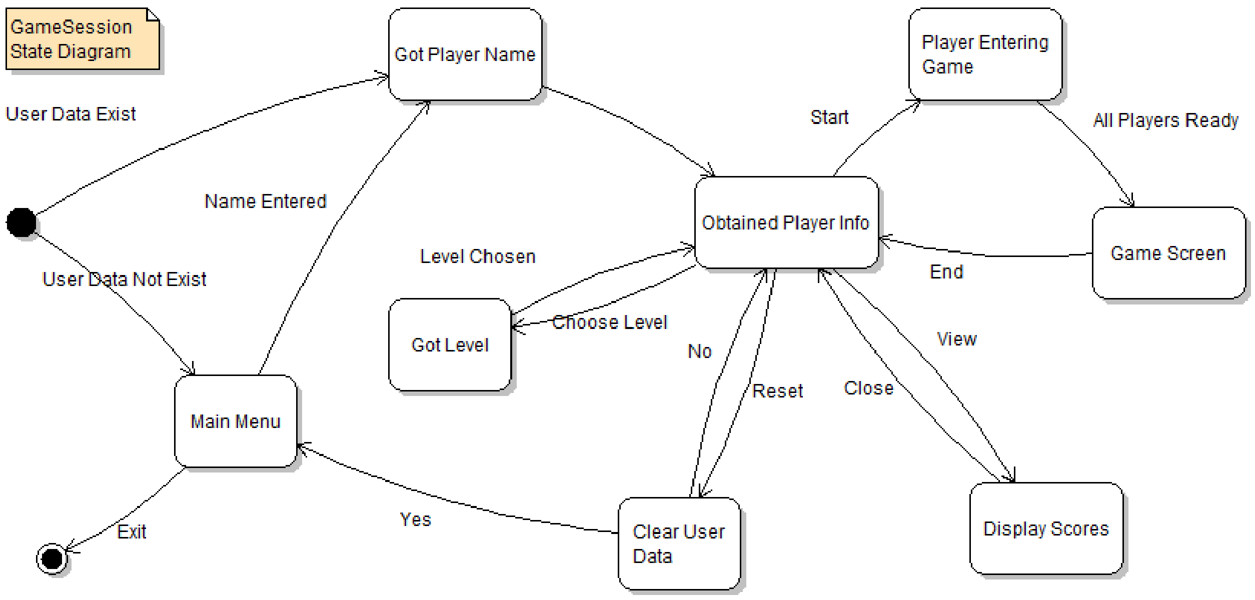


UC7. Change Difficulty Level

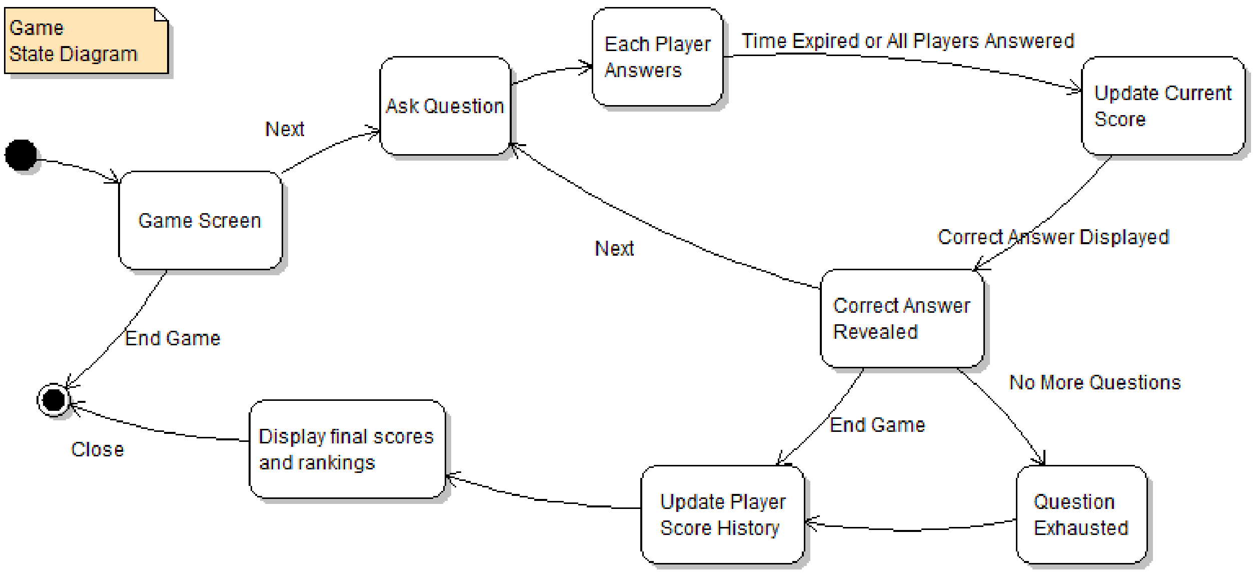


State Diagrams

GameSession



Game



Revision History

Version	Data	Description	Revised By:
0.1	1/31/2020	Initial draft, functional specification, UCs 2, 3, 4, 8	AR
0.2	2/3/2020	Added UCs 1, 5, edited UC2	TST
0.3	2/7/2020	Added UCs 6, 7	LC
0.4	2/11/2020	Edited document for consistency	TST
1.0	2/12/2020	Formatted document for review and submission	TST
1.1	3/10/2020	Modified according to professor's comments	AR
1.2	3/10/2020	Revised requirements with GUI	TST
1.3	3/15/2020	Renamed document, added section placeholders, adding pattern mapping, inserted initial application requirements as attachment, added glossary items	TST
1.4	3/17/2020	Added sequence diagrams and state diagrams	LC
1.5	3/19/2020	Inserted GUI from AR, revised use cases, added domain CRC, class diagrams, revised sequence diagrams	TST
2.0	3/19/2020	Inserted UI class diagrams and UI CRC from AR, made final revisions for submission	TST
2.1	4/24/2020	Updated use cases and all class diagrams except UI class diagrams per design	TST
2.2	4/27/2020	Updated domain class diagram to incorporate QuestionSelector strategy pattern	TST
2.2	4/27/2020	Updated UI class diagram, UI class design patterns, and copied the code over.	AR

Appendix A: Initial Application Requirements

Functional Specification

This will be a single player application which runs offline. Upon launching the application, the user will get a splash screen with “Welcome to Exciting Trivia Game” message. Then a game menu will display where the user will enter his or her name and picks a difficulty level (novice, intermediate or expert). Default will be novice. The game menu will have four additional buttons: Start, View Scores, Reset and Exit. The Start, View Scores and Reset buttons will initially be deactivated until the user’s name is provided. Exit will close the application, View Scores will launch a screen with the user’s top 10 saved scores and the corresponding ranking, and Reset will erase all user data and scores. Start continues to begin the game and opens the game screen.

On the upper left corner of the game screen, a message will display, “Tom has entered the game. Jane has entered the game.” to make the user aware of the other simulated players. On the upper right corner will be a tally marking the points earned by each player. On the center of the screen will be a read only large text box that will display a question followed by some choices a, b, c or d. Under the text box, there will be six buttons. One button for each choice of the question, a next question button and an end game button. The user can end the game at any time. The next button will display a trivia question followed by the possible choices. Once the next button is clicked, the button will be disabled until someone gets the answer or everyone has answered incorrectly. Each player can only choose once. The first player to answer correctly gets the point. A message will display on the large text box on the center with the order of the answers submitted by each player and the correct answer. At this point, the next question button is enabled. Once clicked, the large text box will display the next question.

Once the user clicks end game, the end game screen will appear with the scores and ranking. This screen will have one Close button to close the end game screen and return back to the game menu. If the user plays long enough to answer all the questions, the game will end, and the end game screen will appear with the scores and ranking.

Use Cases

UC1. Launch Application

1. User executes the application.
2. System displays a splash screen with the following message:

```
Welcome to Exciting Trivia Game
```

3. After the splash screen ends, system displays the game menu with the following items:

```
Your Name: (a text box)
Change Difficulty Level:
     Novice (default)
     Intermediate
     Expert
Start (initially deactivated)
View Scores (initially deactivated)
Reset (initially deactivated)
Exit
```

4. User enters his or her name in the textbox and selects “Novice”.
5. All game menu items are activated.

UC1. Launch Application – Variation #1

1.1. Start at **UC1. Launch Application** Step 2.
1.2. System detects an existing user’s name and loads the user data. Splash Screen ends.
1.3. Continue with **UC1. Launch Application** Step 5.

UC2. Start a New Game

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User select “Start”.
3. System displays the game screen.
4. System displays the following messages in sequence on the upper left corner of the game screen:

```
<username> has entered the game…
Tom has entered the game…
Jane has entered the game…
```

5. System displays the current scores (initially zero) of each player on the upper right corner of the game screen.
6. System displays the following message in the center of the screen:

```
Welcome! Please click “Next” to begin the Exciting Game!
```

7. System displays the following six options on the bottom of the game screen:

```
A    B    C    D    Next    End
```

UC3. Scoring a Point

1. User carries out **UC2. Start a New Game**.
2. User clicks the “Next” button.
3. System displays a new question following by four answer choices a, b, c or d.
4. User clicks on any of the buttons {a, b, c, d} to select the answer to the question.
5. After all players select an answer or 10 seconds have elapsed, the system will display the answers picked by each player in descending order of quickest to answer along with the correct answer.
6. System will increment the score for the winner of the round and display the updated score on the upper right corner.

UC4. End the Game

1. User carries out **UC3. Scoring a Point**.
2. User clicks on the “End” button.
3. System displays the end game screen showing each player and their scores and ranking, and a “Close” button.
4. User clicks the “Close” button.
5. System displays the game menu.

UC4. End the Game – Variation #1:

1.1. Start at **UC4. End the Game** Step 1.
1.2. All questions are exhausted in the current game.
1.3. Continue with **UC4. End the Game** Step 3.

UC5. Reset User Data

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User selects “Reset”.
3. System displays the following message:

```
All user data and scores will be erased.
Continue? Yes / No
```

4. User selects “Yes”.
5. System erases all user data and scores and displays the game menu.

UC5. Reset User Data – Variation #1

1.1. Start at **UC5. Reset User Data** Step 3.
1.2. User selects “No”.
1.3. System displays the game menu.

UC6. View Scores

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User selects "View Scores".
3. System displays the top 10 saved scores and rankings.
4. User is prompted to:

```
Press Enter
```

5. System displays the game menu.

UC7. Change Difficulty Level

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User selects "Change Difficulty Level".
3. User is prompted to enter new level:

```
Difficulty Level:
     Novice
     Intermediate
     Expert
```

4. The difficult level is changed.
5. System displays the game menu.

UC8. Exit the Application

1. User carries out **UC1. Launch Application** or other use cases which return to the game menu.
2. User clicks the "Exit" button.
3. The application closes.

Appendix B: Project Source Codes

Program Source Codes

.\src\main\java\exciting\ExcitingGameTester.java

```
package exciting;

import exciting.gui.GameSession;

/**
 * ExcitingGameTester tests the Exciting Game application.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public class ExcitingGameTester {

    /**
     * Main function to test the Exciting Game.
     *
     * @param args (command line arguments are not used)
     */
    public static void main(String[] args) {
        GameSession session = new GameSession();
        session.run();
    }
}
```

.\src\main\java\exciting\game\Game.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.system.Question;
import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.GameState;
import exciting.util.Level;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import javax.swing.Timer;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
```

```
 * Game manages a game play.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public class Game {

    /**
     * Construct a Game object and initialize all variables.
     *
     * @param iter parameter description
     * @param players parameter description
     * @postcondition state == GameState.GAME_BEGAN
     */
    public Game(Iterator<Question> iter, List<Player> players) {
        questionIter = iter;
        this.players = new ArrayList<>(players);
        judge = new Judge();
        judge.setChangeListener((e) -> endCurrentRound());
        listeners = new ArrayList<>();
        gameTimer = new Timer(Constant.MAX_TIME_FOR_ONE_ROUND, (e) ->
endCurrentRound());
        gameTimer.setRepeats(false);
        setState(GameState.GAME_BEGAN);
    }

    /**
     * Initialize a game play and set the difficulty levels of all simulated
     * players.
     *
     * @precondition state == GameState.GAME_BEGAN || state ==
     * GameState.GAME_ENDED
     * @postcondition state == GameState.INITIAL_GAME_SCREEN
     */
    public void initialize() {
        if (state == GameState.GAME_BEGAN
                || state == GameState.GAME_ENDED) {
            // get real player difficulty level
            int realPlayerIndex = getRealPlayerIndex();
            Level lv;
            if (realPlayerIndex >= 0) {
                lv = players.get(realPlayerIndex).getDifficulty();
            } else {
                lv = Level.NOVICE;
            }

            // set simulated player difficulty level
            players.stream().filter(
                    (p) -> (p.getClass() == SimulatedPlayer.class)
            ).forEach((p) -> {
                p.setDifficulty(lv);
            });

            // reset all scores
            players.forEach(
                    (p) -> p.resetCurrentScore()
            );
            setState(GameState.INITIAL_GAME_SCREEN);
        }
    }

    /**
     * Get the next question from the question iterator.
     *
     * @return the next question in a formatted String if the next question
     * exists. If next question does not exist or precondition is not met,
     * return null
     * @precondition state == GameState.INITIAL_GAME_SCREEN || state ==
     * GameState.CORRECT_ANSWER_REVEALED
     * @postcondition state == GameState.QUESTION_ASKED if next question
exists
     * @postcondition state == GameState.QUESTION_EXHAUSTED if next question
     * does not exist
     */
    public synchronized String getNextQuestion() {
        if (state == GameState.INITIAL_GAME_SCREEN
                || state == GameState.CORRECT_ANSWER_REVEALED) {
            judge.reset();
            if (questionIter.hasNext()) {
                Question question = questionIter.next();
                judge.setCorrectAnswer(question.getCorrectAnswer());

                // set simulated player correct answer
                players.stream().filter(
                        (p) -> (p.getClass() == SimulatedPlayer.class)
                ).forEach(
                        (p) -> {
                            ((SimulatedPlayer)
p).setCorrectAnswer(question.getCorrectAnswer());
                            startSimulation((SimulatedPlayer) p);
                        });
                gameTimer.restart();
                setState(GameState.QUESTION_ASKED);
                return question.getQuestionText();
            } else {
                setState(GameState.QUESTION_EXHAUSTED);
                return null;
            }
        } else {
            return null;
        }
    }

    /**
     * Submit answer to the judge for a specific player.
     *
     * @param player the player to submit answer from
     * @param ans the answer of the player
     * @precondition state == GameState.QUESTION_ASKED
     */
    public synchronized void submitAnswerToJudge(Player player, Choice ans) {
        if (state == GameState.QUESTION_ASKED) {
            judge.setPlayerAnswer(player, ans);
        }
    }

    /**
     * Get the answer chosen by each player in a list.
     *
     * @return the answer chosen by each player, in the order the player
     * submitted the answer to the judge. If precondition is not met, return
     * null
     * @precondition state == GameState.CURRENT_SCORE_UPDATED
     */
    public List<Pair<String, Choice>> getResults() {
        if (state == GameState.CURRENT_SCORE_UPDATED) {
            return judge.getPlayerAnswer().stream().map(
                    (p) -> {
                        return new Pair<String, Choice>(p.k().getName(),
p.v());
                    }
            ).collect(Collectors.toList());
        } else {
            return null;
        }
    }

    /**
     * Get the correct answer to the current question.
     *
     * @return the correct answer. If precondition is not met, return null
     * @precondition state == GameState.CURRENT_SCORE_UPDATED
     */
    public Choice getCorrectAnswer() {
        if (state == GameState.CURRENT_SCORE_UPDATED) {
            setState(GameState.CORRECT_ANSWER_REVEALED);
            return judge.getCorrectAnswer();
        } else {
            return null;
        }
    }

    /**
     * Terminate the current game and update the score history to each
player.
     *
     * @precondition state == GameState.CORRECT_ANSWER_REVEALED || state ==
     * GameState.QUESTION_EXHAUSTED
     * @postcondition state == GameState.SCORE_HISTORY_UPDATED
     */
    public synchronized void endGame() {
        if (state == GameState.CORRECT_ANSWER_REVEALED
                || state == GameState.QUESTION_EXHAUSTED) {
            // update all player history
            players.forEach((p) -> p.updateScoreHistory());
            setState(GameState.SCORE_HISTORY_UPDATED);
        }
    }

    /**
     * Reset the judge object and current scores of all players.
     *
     * @precondition state == GameState.SCORE_HISTORY_UPDATED
     * @postcondition state == GameState.GAME_ENDED
     */
    public void cleanUp() {
        if (state == GameState.SCORE_HISTORY_UPDATED) {
            // reset judge and current player scores
            judge.reset();
            players.forEach(
                    (p) -> p.resetCurrentScore()
            );
            setState(GameState.GAME_ENDED);
        }
    }

    /**
     * Get the current scores of all players.
     *
     * @return the list of all current scores of all players, in the order
the
     * player is initialized
     */
    public List<Integer> getScore() {
        return players.stream().map(
                (p) -> p.getCurrentScore()
        ).collect(Collectors.toList());
    }

    /**
     * Get the list of names of all players.
     *
     * @return the list of names of all players, in the order the player is
     * initialized
     */
    public List<String> getPlayersEnteringGame() {
        return players.stream().map(
                (p) -> p.getName()
        ).collect(Collectors.toList());
    }

    /**
     * Add a change listener.
     *
     * @param listener the listener to be added
     */
    public void addChangeListener(ChangeListener listener) {
        listeners.add(listener);
    }

    /**
     * Get the current state of the game object.
     *
     * @return the current state of the game object
     */
    public GameState getState() {
        return state;
    }
    // private methods and helper functions:
    /**
     * End the current round of a game play. If there is a winner of the
current
     * round, increase the score of the winner. It also notifies all
     * ChangeListeners that a change has occurred.
     *
     * @precondition state == GameState.QUESTION_ASKED
     * @postcondition state == GameState.CURRENT_SCORE_UPDATED
     */
    private synchronized void endCurrentRound() {
        if (state == GameState.QUESTION_ASKED) {
            gameTimer.stop();
            if (judge.getWinner() != null) {

judge.getWinner().incrementScore(Constant.SCORE_FOR_ONE_ROUND);
            }
            //System.out.println("Round Ended");
            setState(GameState.CURRENT_SCORE_UPDATED);

            // run state changed for all listeners
            listeners.forEach((l) -> l.stateChanged(new ChangeEvent(this)));
        }
    }

    /**
     * Start simulating the simulated player and randomly select a time to
     * submit answer to the judge.
     *
     */
    private void startSimulation(SimulatedPlayer player) {
        Random ran = new Random();
        int timeToAnswer = Constant.MIN_TIME_TO_ANSWER
                + ran.nextInt(Constant.MAX_TIME_TO_ANSWER -
Constant.MIN_TIME_TO_ANSWER);
        Timer simulationTimer = new Timer(timeToAnswer,
                (e) -> submitAnswerToJudge(player, player.getAnswer()));
        simulationTimer.setRepeats(false);
        simulationTimer.start();
    }

    /**
     * Get the index of the first occurrence of a real player.
     *
     * @return the index of the first occurrence of a real player. If no real
     * player exists, return -1
     */
    private int getRealPlayerIndex() {
        for (int i = 0; i < players.size(); i++) {
            if (players.get(i).getClass() == RealPlayer.class) {
                return i;
            }
        }
        return -1;
    }
```

```java
    /**
     * Set the current state of the game object.
     *
     * @param state the state to be set to
     */
    private void setState(GameState state) {
        this.state = state;
    }

    // instance variables:
    private final Iterator<Question> questionIter;
    private final ArrayList<Player> players;
    private final Judge judge;
    private final Timer gameTimer;
    private final ArrayList<ChangeListener> listeners;
    private GameState state;

}
```

.\src\main\java\exciting\game\Judge.java

```java
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.List;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * Judge keeps track of player's answers and determines the winner.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public class Judge {

    /**
     * Construct a Judge object.
     *
     * @postcondition playerAnswer is initialized
     */
    public Judge() {
        correctAnswer = null;
        playerAnswer = new ArrayList<>();
        winner = null;
        listener = null;
    }

    /**
     * Set the correct answer.
     *
     * @param ans the correct answer
     */
    public void setCorrectAnswer(Choice ans) {
        correctAnswer = ans;
    }

    /**
     * Get the correct answer.
     *
     * @return the correct answer
     */
    public Choice getCorrectAnswer() {
        return correctAnswer;
    }

    /**
     * Set the answer of a player. If the player has already submitted an
answer
     * in the current round, no changes would be made. If all users have
     * answered, it notifies the ChangeListener that a change has occurred.
     *
     * @param player the player who submits the answer
     * @param ans the answer of the player
     */
    public void setPlayerAnswer(Player player, Choice ans) {
        if (!isPlayerAnswered(player)) {
            playerAnswer.add(new Pair<>(player, ans));
            if (ans == correctAnswer && winner == null) {
                winner = player;
            }
            // if all players have answered
            if (playerAnswer.size() == Constant.MAX_PLAYER_NUM && listener !=
null) {
                listener.stateChanged(new ChangeEvent(this));
            }
        }
    }

    /**
     * Get the list of players who submitted answer and their answer. The
list
     * is in the order of when players submitted their answer.
     *
     * @return the list of players with answers
     */
    public List<Pair<Player, Choice>> getPlayerAnswer() {
        return List.copyOf(playerAnswer);
    }

    /**
     * Reset all answers and winner in the Judge object.
     *
     */
    public void reset() {
        correctAnswer = null;
        playerAnswer.clear();
        winner = null;
    }

    /**
     * Get the winner of the current round.
     *
     * @return the winner of the current round. If no winners, return null
     */
    public Player getWinner() {
        return winner;
    }

    /**
     * Set the change listener.
     *
     * @param listener the listener to be set
     */
    public void setChangeListener(ChangeListener listener) {
        this.listener = listener;
    }

    // helper function:
    /**
     * Check if the player has already answered.
     *
     * @param player the player to check
     * @return true if the player has answered, and false otherwise
     */
    private boolean isPlayerAnswered(Player player) {
        boolean answered = false;
        for (Pair p : playerAnswer) {
            if (p.k() == player) {
                answered = true;
                break;
            }
        }
        return answered;
    }

    // instance variables:
    private Choice correctAnswer;
    private final ArrayList<Pair<Player, Choice>> playerAnswer; // the list
of player-answer pair
    private Player winner;
    private ChangeListener listener;
}
```

.\src\main\java\exciting\game\Player.java

```
/*
```

```
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;
import exciting.util.Level;
import exciting.util.Pair;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

/**
 * Player manages player information.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public abstract class Player implements Serializable {

    /**
     * Construct a Player object given a name.
     *
     * @param name the name of the player
     * @postcondition difficulty is initialized to NOVICE
     * @postcondition scoreHistory is initialized
     */
    public Player(String name) {
        this(name, Level.NOVICE);
    }

    /**
     * Construct a Player object given a name and difficulty level.
     *
     * @param name the name of the player
     * @param difficulty the level of the player
     * @postcondition scoreHistory is initialized
     */
    public Player(String name, Level difficulty) {
        this.name = name;
        this.difficulty = difficulty;
        currentScore = 0;
        scoreHistory = new ArrayList<>();
    }

     /**
     * Set the name of the player.
     *
     * @param name the name of the player
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Get the name of the player.
     *
     * @return the name of the player
     */
    public String getName() {
        return name;
    }

    /**
     * Get the current score of the player.
     *
     * @return current score of the player
     */
    public int getCurrentScore() {
        return currentScore;
    }

    /**
     * Increment the current score of the player.
     *
     * @param scoreIncrement the score to be added to the current score
     */
    public void incrementScore(int scoreIncrement) {
        currentScore += scoreIncrement;
    }

    /**
     * Reset the current score of the player.
     *
     */
    public void resetCurrentScore() {
        currentScore = 0;
    }

    /**
     * Add the current score to the score history.
     *
     */
    public void updateScoreHistory() {
        scoreHistory.add(new Pair<>(new Date(), currentScore));
    }

    /**
     * Get the most recent scores of the player, up to the number
     * specified in the parameter number. The list is ordered from
     * the oldest score to the newest score.
     *
     * @param number the number of history items to be retrieved
     * @return the most recent scores of the player. If no scores,
     * return an empty list
     */
    public List<Pair<Date, Integer>> getScoreHistory(int number) {
        return List.copyOf(scoreHistory.subList(Math.max(0,
scoreHistory.size() - number), scoreHistory.size()));
    }

    /**
     * Set the difficulty level of the player.
     *
     * @param difficulty the difficulty level of the player
     */
    public void setDifficulty(Level difficulty) {
        this.difficulty = difficulty;
    }
    /**
     * Get the difficulty level of the player.
     *
     * @return the difficulty level of the player
     */
    public Level getDifficulty() {
        return difficulty;
    }

    // instance variables:
    private String name;
    private int currentScore;
    private ArrayList<Pair<Date, Integer>> scoreHistory; // list of score
history
    private Level difficulty; // difficulty level of the player
}
```

.\src\main\java\exciting\game\RealPlayer.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;
import exciting.util.Level;

/**
 * RealPlayer manages a human player's information.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public class RealPlayer extends Player {

    /**
     * Construct a RealPlayer object given a name.
     *
     * @param name the name of the player
     */
    public RealPlayer(String name) {
        super(name);
    }

    /**
     * Construct a RealPlayer object given a name and difficulty level.
     *
     * @param name the name of the human player
     * @param difficulty the level of the human player
     */
    public RealPlayer(String name, Level difficulty) {
        super(name, difficulty);
    }

}
```

.\src\main\java\exciting\game\SimulatedPlayer.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.Level;
import java.util.Random;

/**
 * SimulatedPlayer manages a simulated player's information.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public class SimulatedPlayer extends Player {

    /**
     * Construct a SimulatedPlayer object given a name.
     *
     * @param name the name of the simulated player
     */
    public SimulatedPlayer(String name) {
        super(name);
    }

    /**
     * Set the correct answer for the simulated player.
     *
     * @param ans the correct answer to a question
     */
    public void setCorrectAnswer(Choice ans) {
        correctAnswer = ans;
    }

    /**
     * Get an answer from the simulated player based on the difficulty level.
     * The higher the difficulty level, the higher the chance the correct
answer
     * is returned. For example, for Constant.MAX_CHANCE = 4,
     * Constant.MAX_NUM_CHOICES = 4, and lv = Level.NOVICE (lv.ordinal() =
0),
     * the chance of returning the correct answer is: (lv.ordinal() + 1) /
     * Constant.MAX_CHANCE + (1 - (lv.ordinal() + 1) / Constant.MAX_CHANCE) *
1
     * / MAX_NUM_CHOICES = 1 / 4 + (1 - 1 / 4) * 1 / 4 = 0.4375
     *
     * @return the answer from the simulated player
     */
    public Choice getAnswer() {
        Random ran = new Random();
        int chance = ran.nextInt(Constant.MAX_CHANCE);
        Level lv = super.getDifficulty();
        if (chance <= lv.ordinal()) {
            return correctAnswer;
        } else {
            return Choice.values()[ran.nextInt(Constant.MAX_NUM_CHOICES)];
        }
    }

    // instance variables:
    private Choice correctAnswer;

}
```

.\src\main\java\exciting\gui\GameMenu.java

```
package exciting.gui;

import exciting.util.Level;
import java.util.ArrayList;
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.KeyListener;
import java.util.EventListener;

/**
 * Game Menu UI class is the starting point for the user of the Game. The
user
 * will be able to start a new game or view score history and more from this
UI.
 *
 * @author Alejandro Reyes
 * @version 1.1
 */
public class GameMenu {

    /**
     * Constructor of the Game Menu UI class and all its components.
     *
     */
    public GameMenu() {
        pnGameMenu = new JPanel();
        lbGameMenu = new JLabel();
        lbName = new JLabel();
        tfName = new JTextField();
        lbChangeDifficulty = new JLabel();
        cbDifficultyLevel = new JComboBox<>();
        btStart = new JButton();
        btViewScores = new JButton();
        btReset = new JButton();
        btExit = new JButton();

        lbGameMenu.setText("Game Menu");
        lbName.setText("Name:");
        lbChangeDifficulty.setText("Change Difficulty Level:");
        cbDifficultyLevel.setModel(new DefaultComboBoxModel<>(new
String[]{Level.NOVICE.toString(), Level.INTERMEDIATE.toString(),
Level.ADVANCED.toString()}));

        btStart.setText("Start");
        btViewScores.setText("View Scores");
        btReset.setText("Reset");
        btExit.setText("Exit");

        componentLayout();

        // add actionable components to array
        componentArray = new ArrayList<>();
        componentArray.add(tfName); // 0 JTextField
        componentArray.add(cbDifficultyLevel); // 1 JCombobox
        componentArray.add(btStart); // 2 JButton
        componentArray.add(btViewScores); // 3 JButton
        componentArray.add(btReset); // 4 JButton
        componentArray.add(btExit); // 5 JButton
    }

    /**
     * Gets the game menu UI.
     *
     * @return returns JPanel with all of the components for the UI.
     */
    public JPanel getGameMenuPanel() {
        return pnGameMenu;
    }

    /**
     * Get the users name entered by the user in the game menu UI.
     *
     * @return returns users name.
     * @precondition username is not empty.
     */
    public String getUserName() {
        return tfName.getText();
    }

    /**
     * Gets the difficult level entered by the user on the game menu UI.
     *
     * @return returns a level, Novice, Intermediate or Advanced.
     */
    public Level getDifficultyLevel() {
        if (cbDifficultyLevel.getSelectedItem().getClass() == String.class) {
            return Level.valueOf(((String)
cbDifficultyLevel.getSelectedItem()).toUpperCase());
        } else {
            return null;
        }
    }

    /**
     * Set the users name on the screen if the system is able to recognize
the
     * user as a previous player.
     */
    public void setUserName(String name) {
        tfName.setText(name);
    }

    /**
     * Set the users difficulty level on the screen if the system is able to
     * recognize the user as a previous player.
     *
     */
    public void setDifficultyLevel(Level lv) {
        cbDifficultyLevel.setSelectedItem(lv.toString());
    }

    /**
     * Enable or disable (if no user information) the menu items
     *
     * @param b True to enable and false to disable the game menu.
     */
    public void setEnabledMenu(boolean b) {
        cbDifficultyLevel.setEnabled(b);
        btReset.setEnabled(b);
        btStart.setEnabled(b);
        btViewScores.setEnabled(b);
    }

    /**
     * @precondition Listener must be instance of KeyListener for TextFields
and
     * ActionListener for ComboBoxes or Buttons, otherwise no changes are
made.
     */
    public void addListener(EventListener listener, JItem item) {
        if (componentArray.get(item.ordinal()).getClass() == JTextField.class
                && listener instanceof KeyListener) {
            ((JTextField)
componentArray.get(item.ordinal())).addKeyListener((KeyListener) listener);
        } else if (componentArray.get(item.ordinal()).getClass() ==
JComboBox.class
                && listener instanceof ActionListener) {
            ((JComboBox)
componentArray.get(item.ordinal())).addActionListener((ActionListener)
listener);
        } else if (componentArray.get(item.ordinal()).getClass() ==
JButton.class
```

```
                && listener instanceof ActionListener) {
            ((JButton)
componentArray.get(item.ordinal())).addActionListener((ActionListener)
listener);
        }
    }

    /**
     * Inserts all components in the JPanel in the proper location.
     *
     */
    private void componentLayout() {
        GroupLayout layout = new GroupLayout(pnGameMenu);
        layout.setAutoCreateContainerGaps(true);
        layout.setAutoCreateGaps(true);
        pnGameMenu.setLayout(layout);
        layout.setHorizontalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                        .addComponent(lbGameMenu)
                        .addGroup(layout.createParallelGroup(GroupLayout.Alig
nment.LEADING)
                                .addComponent(lbName)
                                .addComponent(tfName)
                                .addComponent(lbChangeDifficulty)
                                .addComponent(cbDifficultyLevel, 0,
GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(btStart,
GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(btViewScores,
GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(btReset,
GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addComponent(btExit,
GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        )
        );
        layout.setVerticalGroup(
                layout.createSequentialGroup()
                        .addComponent(lbGameMenu)
                        .addComponent(lbName)
                        .addComponent(tfName, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                        .addComponent(lbChangeDifficulty)
                        .addComponent(cbDifficultyLevel,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE)
                        .addComponent(btStart)
                        .addComponent(btViewScores)
                        .addComponent(btReset)
                        .addComponent(btExit)
        );
    }

    // constants to select actionable components to add listeners to
    enum JItem {
        NAME_TEXTFIELD,
        DIFFICULT_LEVEL_CB,
        START_BUTTON,
        VIEW_SCORES_BUTTON,
        RESET_BUTTON,
        EXIT_BUTTON;
    }

    // instance variables:
    private final ArrayList<JComponent> componentArray;
    private final JPanel pnGameMenu;
    private final JButton btExit;
    private final JButton btReset;
    private final JButton btStart;
    private final JButton btViewScores;
    private final JComboBox<String> cbDifficultyLevel;
    private final JLabel lbChangeDifficulty;
    private final JLabel lbGameMenu;
    private final JLabel lbName;
    private final JTextField tfName;
}
```

.\src\main\java\exciting\gui\GameScreen.java

```
package exciting.gui;

import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.Pair;
import java.awt.Dimension;
import javax.swing.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.EventListener;
import java.util.List;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * Game screen is the screen where the user reads questions and provides
answers
 * to score points and attempt to win the game.
 *
 * @author Alejandro Reyes
 * @version 1.1
 */
public class GameScreen {

    /**
     * Constructor of the Game Screen UI class and all its components.
     *
     */
    public GameScreen() {
        pnGameScreen = new JPanel();
        pnGameScreen.setPreferredSize(new Dimension(400, 375));
        lbUser = new JLabel();
        lbAIPlayer1 = new JLabel();
        lbAIPlayer2 = new JLabel();
        lbUserScore = new JLabel();
        lbAIPlayer1Score = new JLabel();
        lbAIPlayer2Score = new JLabel();
        jScrollPane1 = new JScrollPane();
        taDisplay = new JTextArea();
        taDisplay.setEditable(false);
        btEndGame = new JButton();
        btNext = new JButton();
        btA = new JButton();
        btB = new JButton();
        btC = new JButton();
        btD = new JButton();

        taDisplay.setColumns(20);
        taDisplay.setRows(10);
        jScrollPane1.setViewportView(taDisplay);

        btEndGame.setText("End Game");
        btNext.setText("Next");
        btA.setText("A");
        btB.setText("B");
        btC.setText("C");
        btD.setText("D");

        componentLayout();

        // add actionable components to array
        componentArray = new ArrayList<>();
        componentArray.add(btA); // 0 JButton
        componentArray.add(btB); // 1 JButton
        componentArray.add(btC); // 2 JButton
        componentArray.add(btD); // 3 JButton
        componentArray.add(btEndGame); // 4 JButton
        componentArray.add(btNext); // 5 JButton

        numPlayersEnteredGame = 0;

    }

    /**
     * Update the score for the user to see current score.
     *
     * @param score is a list of player scores.
     * @param num is the number of scores to display.
     * @precondition score has the real player score first and Jane's score
     * last.
     * @postcondition score values display on the screen correctly.
     */
    public void updateScore(List<Integer> score, int num) {
        if (score.size() >= Constant.MAX_PLAYER_NUM) {
            lbUserScore.setText(" ");
            lbAIPlayer1Score.setText(" ");
            lbAIPlayer2Score.setText(" ");
            if (num >= 1) {
                lbUserScore.setText("Score: " + score.get(0));
            }
            if (num >= 2) {
                lbAIPlayer1Score.setText("Score: " + score.get(1));
            }
            if (num >= 3) {
                lbAIPlayer2Score.setText("Score: " + score.get(2));
            }
        }
    }

    /**
     * Set the next question on the screen for the user.
     *
     * @param aformattedQuestion is a formatted question to display on the
     * screen.
     */
    public void showNextQuestion(String aformattedQuestion) {
        taDisplay.setText(aformattedQuestion);
    }

    /**
     * Show the results of a round on the screen for the user to see.
     *
     * @param result has the answer provided by each player of the game.
     * @param currectAnswer is the correct answer to the question of the
round.
     */
    public void showResults(List<Pair<String, Choice>> result, Choice
currectAnswer) {
        String str = new String();
        for (Pair p : result) {
            str += p.k() + " answered " + p.v() + "\n";
        }
        str += "Correct answer is: " + currectAnswer;
        taDisplay.setText(str);
    }

    /**
     * Gets the number of players playing the game.
     *
     * @return returns the number of players playing the game.
     */
    public int getNumPlayersEnteredGame() {
        return numPlayersEnteredGame;
    }

    /**
     * Reset the number of players playing the game to zero.
     */
    public void resetNumPlayersEnteredGame() {
        numPlayersEnteredGame = 0;
    }

    /**
     * Shows the players entering the game. Simulation to give the user the
     * impression that other players are coming in to join the game.
     *
     * @param name is a list of the players. One real player and 2 AI
players.
     */
    public void showPlayersEnteringGame(List<String> name) {
        lbUser.setText(" ");
        lbAIPlayer1.setText(" ");
        lbAIPlayer2.setText(" ");
        if (name.size() >= Constant.MAX_PLAYER_NUM) {
            lbUser.setText(name.get(0) + " has Entered the Game.");
            onePlayerEntered();

            Timer t = new Timer(2000, (ActionEvent e) -> {
                lbAIPlayer1.setText(name.get(1) + " has Entered the Game.");
                onePlayerEntered();
                if (numPlayerListener != null) {
                    numPlayerListener.stateChanged(new ChangeEvent(this));
                }
            });
            t.setRepeats(false);
            t.start();

            Timer tt = new Timer(4000, (ActionEvent e) -> {
                lbAIPlayer2.setText(name.get(2) + " has Entered the Game.");
                onePlayerEntered();
                if (numPlayerListener != null) {
                    numPlayerListener.stateChanged(new ChangeEvent(this));
                }
            });
            tt.setRepeats(false);
            tt.start();
        }
    }

    /**
     * @precondition Listener must be instance of ActionListener for Buttons,
     * and ChangeListener for others.
     */
    public void addListener(EventListener listener, JItem item) {
        if (item == JItem.NUM_PLAYER_CHANGE
                && listener instanceof ChangeListener) {
            numPlayerListener = (ChangeListener) listener;
        } else if (componentArray.get(item.ordinal()).getClass() ==
JButton.class
                && listener instanceof ActionListener) {
            ((JButton)
componentArray.get(item.ordinal())).addActionListener((ActionListener)
listener);
        }
    }

    /**
     * Gets the game screen UI.
     *
     * @return returns JPanel with all of the components for the UI.
     */
    public JPanel getGameScreenPanel() {
        return pnGameScreen;
    }

    /**
     * Inserts all components in the JPanel in the proper location.
     *
     */
    private void componentLayout() {
        GroupLayout layout = new GroupLayout(pnGameScreen);
        pnGameScreen.setLayout(layout);
        layout.setHorizontalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                                .addContainerGap()
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.LEADING)
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addComponent(lbUser)
                                                .addPreferredGap(LayoutStyle.
ComponentPlacement.RELATED, 197, Short.MAX_VALUE)
                                                .addComponent(lbUserScore))
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addComponent(lbAIPlayer2)
                                                .addPreferredGap(LayoutStyle.
ComponentPlacement.RELATED, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                                .addComponent(lbAIPlayer2Scor
e))
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addComponent(lbAIPlayer1)
                                                .addPreferredGap(LayoutStyle.
ComponentPlacement.RELATED, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                                .addComponent(lbAIPlayer1Scor
e))
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addGroup(layout.createParall
elGroup(GroupLayout.Alignment.LEADING)
                                                        .addComponent(btEndGa
me)
                                                        .addComponent(btA)
                                                        .addComponent(btB))
                                                .addPreferredGap(LayoutStyle.
ComponentPlacement.RELATED, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                                .addGroup(layout.createParall
elGroup(GroupLayout.Alignment.LEADING)
                                                        .addComponent(btNext,
GroupLayout.Alignment.TRAILING)
                                                        .addComponent(btC,
GroupLayout.Alignment.TRAILING)
                                                        .addComponent(btD,
GroupLayout.Alignment.TRAILING)))
                                        .addComponent(jScrollPane1))
                                .addContainerGap())
        );
        layout.setVerticalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                                .addContainerGap()
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(lbUser)
                                        .addComponent(lbUserScore))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(lbAIPlayer1,
GroupLayout.DEFAULT_SIZE, GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                        .addComponent(lbAIPlayer1Score))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(lbAIPlayer2)
                                        .addComponent(lbAIPlayer2Score))
                                .addGap(21, 21, 21)
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(btEndGame)
                                        .addComponent(btNext))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(jScrollPane1,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(btA)
                                        .addComponent(btC))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.BASELINE)
                                        .addComponent(btB)
                                        .addComponent(btD))
                                .addContainerGap(40, Short.MAX_VALUE))
        );
    }

    /**
     * Method to update number of players entered game.
     */
    private synchronized void onePlayerEntered() {
        numPlayersEnteredGame++;
    }

    // constants to select components to add listeners to
    enum JItem {
        A_BUTTON,
        B_BUTTON,
        C_BUTTON,
        D_BUTTON,
        END_GAME_BUTTON,
        NEXT_BUTTON,
        NUM_PLAYER_CHANGE;
    }

    // instance variables:
    private final ArrayList<JComponent> componentArray;
    private final JPanel pnGameScreen;
    private final JButton btA;
    private final JButton btB;
    private final JButton btC;
    private final JButton btD;
    private final JButton btEndGame;
    private final JButton btNext;
    private final JLabel lbAIPlayer1;
    private final JLabel lbAIPlayer1Score;
    private final JScrollPane jScrollPane1;
    private final JTextArea taDisplay;
    private final JLabel lbAIPlayer2;
    private final JLabel lbAIPlayer2Score;
    private final JLabel lbUser;
    private final JLabel lbUserScore;

    // added variables to keep track how many players have entered the game
    private int numPlayersEnteredGame;
    private ChangeListener numPlayerListener;
}
```

.\src\main\java\exciting\gui\GameSession.java

```
package exciting.gui;

import exciting.game.Game;
import exciting.system.GameSystem;
import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.GameState;
import exciting.util.Level;
import java.awt.CardLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.concurrent.locks.ReentrantLock;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.Timer;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * GameSession manages the state of the game application.
 *
 * @author Alejandro Reyes, Tsz Shing Tsoi
 */
public class GameSession {

    /**
     * Construct a GameSession object, initialize all graphical user
interface
     * (GUI) and game system objects and add listeners, and create the main
     * JFrame object.
     *
     * @postcondition state == GameSessionState.GAME_SESSION_BEGAN
     */
    public GameSession() {
        cardLayout = new CardLayout();
        gameScreen = new GameScreen();
        gameMenu = new GameMenu();
        viewScore = new ViewScore();
        splashScreen = new SplashScreen();

        mainPanel = new JPanel(cardLayout);
        mainPanel.add("SplashScreen", splashScreen.getSplashScreenPanel());
        mainPanel.add("GameMenu", gameMenu.getGameMenuPanel());
        mainPanel.add("GameScreen", gameScreen.getGameScreenPanel());
        mainPanel.add("ViewScore", viewScore.getViewScorePanel());

        mainFrame = new JFrame();
        mainFrame.add(mainPanel);
        mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mainFrame.pack();
        mainFrame.setResizable(false);

        splashScreenTimer = new Timer(3000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                cardLayout.show(mainPanel, "GameMenu");
            }
        });
        splashScreenTimer.setRepeats(false);

        gameSystem = new GameSystem();
        game = null;
        addGameSystemListener();
        addGameMenuListener();
        addGameScreenListener();
        addViewScoreScreenListener();
        setState(GameSessionState.GAME_SESSION_BEGAN);

        stateLock = new ReentrantLock();
    }

    /**
     * Run the Exciting Game application.
     *
     */
    public void run() {
        mainFrame.setVisible(true);
        splashScreenTimer.start();
        gameSystem.initialize();
    }

    // private methods and helper functions:
    /**
     * Set the state of the GameSession object.
     *
     * @param state the state to be set to
     */
    private void setState(GameSessionState state) {
        this.state = state;
    }

    /**
     * Add listener to the GameSystem object to update user name and
difficulty
     * level.
     *
     */
    private void addGameSystemListener() {
        gameSystem.addChangeListener((e) -> {
            stateLock.lock();
            try {
                if (state == GameSessionState.GAME_SESSION_BEGAN
                        || state == GameSessionState.OBTAINED_PLAYER_INFO) {
                    if (gameSystem.isUserDataExist()) {
                        gameMenu.setUserName(gameSystem.getUserName());
                        gameMenu.setDifficultyLevel(gameSystem.getDiff());
                        gameMenu.setEnabledMenu(true);
                        setState(GameSessionState.OBTAINED_PLAYER_INFO);
                    } else {
                        gameMenu.setUserName("");
                        gameMenu.setDifficultyLevel(Level.NOVICE);
                        gameMenu.setEnabledMenu(false);
                        setState(GameSessionState.MAIN_MENU);
                    }
                }
            } finally {
                stateLock.unlock();
            }

        });
    }

    /**
     * Add listener to the Game object to update game screen labels.
     *
     */
    private void addGameListener() {
        game.addChangeListener((e) -> {
            if (state == GameSessionState.GAME_SCREEN) {
                gameScreen.showResults(game.getResults(),
game.getCorrectAnswer());
                gameScreen.updateScore(game.getScore(),
gameScreen.getNumPlayersEnteredGame());
            }
        });
    }

    /**
     * Add listener to the ViewScreen object to return to the game menu.
     *
     */
    private void addViewScoreScreenListener() {
        viewScore.addListener((e) -> {
            stateLock.lock();
            try {
                if (state == GameSessionState.DISPLAY_SCORES
                        || game.getState() == GameState.GAME_ENDED) {
                    setState(GameSessionState.OBTAINED_PLAYER_INFO);
                    cardLayout.show(mainPanel, "GameMenu");
                }
            } finally {
                stateLock.unlock();
            }
        });
    }

    /**
     * Add listeners to the GameScreen object to receive user actions.
     *
     */
    private void addGameScreenListener() {
        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            // submit answer
            game.submitAnswerToJudge(gameSystem.getRealPlayer(), Choice.A);
        }, GameScreen.JItem.A_BUTTON);

        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            // submit answer
            game.submitAnswerToJudge(gameSystem.getRealPlayer(), Choice.B);
        }, GameScreen.JItem.B_BUTTON);

        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            // submit answer
            game.submitAnswerToJudge(gameSystem.getRealPlayer(), Choice.C);
        }, GameScreen.JItem.C_BUTTON);

        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            // submit answer
            game.submitAnswerToJudge(gameSystem.getRealPlayer(), Choice.D);
        }, GameScreen.JItem.D_BUTTON);

        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            stateLock.lock();
            try {
                // end the game
                game.endGame();
                if (game.getState() == GameState.SCORE_HISTORY_UPDATED) {
                    // update score in viewScore screen
                    viewScore.showFinalScores(gameSystem.getScore());
                    game.cleanUp();
                    cardLayout.show(mainPanel, "ViewScore");
                    gameScreen.showNextQuestion(" ");
                    gameScreen.resetNumPlayersEnteredGame();
                } else if (state == GameSessionState.GAME_SCREEN
                        && game.getState() == GameState.INITIAL_GAME_SCREEN)
{ // if game is ended before it started
                    cardLayout.show(mainPanel, "GameMenu");
                    gameScreen.showNextQuestion(" ");
                    gameScreen.resetNumPlayersEnteredGame();
                    setState(GameSessionState.OBTAINED_PLAYER_INFO);
                }
            } finally {
                stateLock.unlock();
            }
        }, GameScreen.JItem.END_GAME_BUTTON);

        gameScreen.addListener((ActionListener) (ActionEvent e) -> {
            if (state == GameSessionState.GAME_SCREEN) {
                String q = game.getNextQuestion();
                if (game.getState() == GameState.QUESTION_EXHAUSTED) {
                    game.endGame();
                    if (game.getState() == GameState.SCORE_HISTORY_UPDATED) {
                        // update score in viewScore screen
                        viewScore.showFinalScores(gameSystem.getScore());
                        cardLayout.show(mainPanel, "ViewScore");
                        gameScreen.resetNumPlayersEnteredGame();
                    }
                } else if (q != null) {
                    gameScreen.showNextQuestion(q);
                }
            }
        }, GameScreen.JItem.NEXT_BUTTON);

        gameScreen.addListener((ChangeListener) (ChangeEvent e) -> {
            stateLock.lock();
            try {
                if (state == GameSessionState.PLAYER_ENTERING) {
                    gameScreen.updateScore(game.getScore(),
gameScreen.getNumPlayersEnteredGame());
                }
                if (gameScreen.getNumPlayersEnteredGame() ==
Constant.MAX_PLAYER_NUM) {
                    gameScreen.showNextQuestion("Welcome! Please click
\"Next\" to begin the Exciting Game!");
                    setState(GameSessionState.GAME_SCREEN);
                }
            } finally {
                stateLock.unlock();
            }
        }, GameScreen.JItem.NUM_PLAYER_CHANGE);
    }

    /**
     * Add listeners to the GameMenu object to receive user actions.
     *
     */
    private void addGameMenuListener() {
        gameMenu.addListener(new KeyAdapter() {
            public void keyReleased(KeyEvent e) {
                stateLock.lock();
                try {
                    // set name of player
                    String name = gameMenu.getUserName();
                    if (name.length() > 0) {
                        gameSystem.setUserName(name);
                        gameMenu.setEnabledMenu(true);
                        setState(GameSessionState.OBTAINED_PLAYER_INFO);
                    }
                } finally {
                    stateLock.unlock();
                }
            }
        }, GameMenu.JItem.NAME_TEXTFIELD);

        gameMenu.addListener((ActionListener) (ActionEvent e) -> {
            // set difficulty level of player in GameSystem
            gameSystem.changeDifficulty(gameMenu.getDifficultyLevel());
        }, GameMenu.JItem.DIFFICULT_LEVEL_CB);

        gameMenu.addListener((ActionListener) (ActionEvent e) -> {
            stateLock.lock();
            try {
                if (state == GameSessionState.OBTAINED_PLAYER_INFO) {
                    game = gameSystem.getNewGame();
                    addGameListener();
                    game.initialize();
                    cardLayout.show(mainPanel, "GameScreen");

gameScreen.showPlayersEnteringGame(game.getPlayersEnteringGame());
                    gameScreen.updateScore(game.getScore(),
gameScreen.getNumPlayersEnteredGame());
                    setState(GameSessionState.PLAYER_ENTERING);
                }
            } finally {
                stateLock.unlock();
            }
        }, GameMenu.JItem.START_BUTTON);

        gameMenu.addListener((ActionListener) (ActionEvent e) -> {
            stateLock.lock();
            try {
                if (state == GameSessionState.OBTAINED_PLAYER_INFO) {
                    // update score in viewScore screen
                    viewScore.showFinalScores(gameSystem.getScore());
                    cardLayout.show(mainPanel, "ViewScore");
                    setState(GameSessionState.DISPLAY_SCORES);
                }
            } finally {
                stateLock.unlock();
            }
        }, GameMenu.JItem.VIEW_SCORES_BUTTON);

        gameMenu.addListener((ActionListener) (ActionEvent e) -> {
            stateLock.lock();
            try {
                if (getResetConfirmation() == JOptionPane.YES_OPTION) {
                    if (state == GameSessionState.OBTAINED_PLAYER_INFO) {
                        gameSystem.resetData();
                    }
                }
            } finally {
                stateLock.unlock();
            }
        }, GameMenu.JItem.RESET_BUTTON);

        gameMenu.addListener((ActionListener) (ActionEvent e) -> {
            stateLock.lock();
            try {
                // save data in gamesystem
                if (state == GameSessionState.OBTAINED_PLAYER_INFO) {
                    gameSystem.saveUserData();
                }
            } finally {
                stateLock.unlock();
            }
            System.exit(0);
        }, GameMenu.JItem.EXIT_BUTTON);

    }

    /**
     * Get confirmation from the user on resetting data.
     *
     * @return the option the user selected
     */
    private int getResetConfirmation() {
        return JOptionPane.showConfirmDialog(
                mainFrame, "All user data and scores will be erased.
Continue?",
                "Reset Confirmation", JOptionPane.YES_NO_OPTION,
JOptionPane.WARNING_MESSAGE);
    }

    /**
     * GameSessionState describes the states of a game session.
     *
     */
    enum GameSessionState {
        // enum constants are public, static and final by default
        GAME_SESSION_BEGAN,
        MAIN_MENU,
        OBTAINED_PLAYER_INFO,
        PLAYER_ENTERING,
        GAME_SCREEN,
        DISPLAY_SCORES,
    }

    // private variables:
    private final ReentrantLock stateLock; // lock to crtical section to
update current state
    private GameSessionState state;
```

```java
    // private variables for GUI
    private final GameMenu gameMenu;
    private final GameScreen gameScreen;
    private final ViewScore viewScore;
    private final SplashScreen splashScreen;
    private CardLayout cardLayout;
    private JPanel mainPanel;
    private final JFrame mainFrame;
    private final Timer splashScreenTimer;

    // private variables for model
    private final GameSystem gameSystem;
    private Game game;
}
```

.\src\main\java\exciting\gui\SplashScreen.java

```java
package exciting.gui;

import javax.swing.GroupLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * Splash Screen class displays a banner to welcome the user to the game.
 *
 * @author Alejandro Reyes
 * @version 1.1
 */
public class SplashScreen {

    /**
     * Constructor of the Splash Screen.
     *
     */
    public SplashScreen() {
        lbWelcomeUser = new JLabel("Welcome to the Exciting Trivia Game.");
        splashScreenPanel = new JPanel();
        GroupLayout layout = new GroupLayout(splashScreenPanel);
        layout.setAutoCreateContainerGaps(true);
        layout.setAutoCreateGaps(true);
        splashScreenPanel.setLayout(layout);
        layout.setHorizontalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                        .addComponent(lbWelcomeUser)
        );
        layout.setVerticalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                        .addComponent(lbWelcomeUser)
        );
    }

    /**
     * Gets the splash screen JPanel.
     *
     * @return returns the JPanel with the welcome message to the user.
     */
    public JPanel getSplashScreenPanel() {
        return splashScreenPanel;
    }

    // instance variables:
    private final JLabel lbWelcomeUser;
    private final JPanel splashScreenPanel;
}
```

.\src\main\java\exciting\gui\ViewScore.java

```
package exciting.gui;

import exciting.util.Constant;
import exciting.util.Pair;
import javax.swing.*;
import java.awt.event.*;
import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

/**
 * ViewScore class contains a JPanel with labels for the history of users
games
 * played. It has 10 labels for a possible list of the last 10 games played.
 *
 * @author Alejandro Reyes
 * @version 1.1
 */
public class ViewScore {

    /**
     * Constructor of the ViewScoree class. Initially all the labels are
empty
     * until the user plays a game.
     *
     * @return return ViewScore object.
     */
    public ViewScore() {
        lbtitle = new JLabel();
        btBackToGameMenu = new JButton();
        pnViewScore = new JPanel();
        lbtitle.setText("Last 10 Games");
        btBackToGameMenu.setText("Back to Game Menu");
        scoreLabels = new ArrayList<>();

        for (int i = 0; i < Constant.MAX_SCORE_HISTORY; i++) {
            scoreLabels.add(new JLabel());
        }

        componentLayout();
    }
    /**
     * Add action listener to the Back to Game Menu button that is on the
     * JPanel.
     *
     */
    public void addListener(ActionListener listener) {
        btBackToGameMenu.addActionListener(listener);
    }

    /**
     * Get the View Score UI.
     *
     * @return returns the JPanel with the score history.
     */
    public JPanel getViewScorePanel() {
        return pnViewScore;
    }

    /**
     * Fills history for the user in the View Scores UI.
     *
     * @param scores is a list of the score history for each player.
     * @precondition scores has real player history first and AI player Jane
     * history last.
     * @precondition scores should have history of three players exactly.
     * @postcondition JPanel will contain labels with the history of games
for
     * the user to see in the proper format.
     */
    public void showFinalScores(List<List<Pair<Date, Integer>>> scores) {
        DateFormat df = DateFormat.getInstance();
        String yourScore = "";
        String tomsScore = "";
        String janesScore = "";
        String dateTime = "";

        for (int i = 0; i < Constant.MAX_SCORE_HISTORY; i++){
            scoreLabels.get(i).setText("");
        }

        if (scores.size() > 0) {
            for (int i = 0; i < Constant.MAX_SCORE_HISTORY; i++) {

                if (i < scores.get(0).size()) {
                    dateTime = df.format((scores.get(0).get(i).k()));
                    yourScore = scores.get(0).get(i).v().toString();
                    tomsScore = scores.get(1).get(i).v().toString();
                    janesScore = scores.get(2).get(i).v().toString();

                    scoreLabels.get(i).setText((i + 1) + ". " + dateTime + "
"
                            + "  Your Score: " + yourScore
                            + "  Tom's Score: " + tomsScore
                            + "  Jane's Score: " + janesScore + "\n");
                }
            }
        }
    }

    /**
     * Inserts all components in the JPanel in the proper location.
     *
     */
    private void componentLayout() {
        GroupLayout layout = new GroupLayout(pnViewScore);
        pnViewScore.setLayout(layout);
        layout.setHorizontalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                                .addGroup(layout.createParallelGroup(GroupLay
out.Alignment.LEADING)
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addContainerGap()
                                                .addGroup(layout.createParall
elGroup(GroupLayout.Alignment.LEADING)
                                                        .addComponent(lbtitle
)
                                                        .addComponent(scoreLa
bels.get(0))
                                                        .addComponent(scoreLa
bels.get(1))
                                                        .addComponent(scoreLa
bels.get(2))
                                                        .addComponent(scoreLa
bels.get(3))
                                                        .addComponent(scoreLa
bels.get(4))
                                                        .addComponent(scoreLa
bels.get(5))
                                                        .addComponent(scoreLa
bels.get(6))
                                                        .addComponent(scoreLa
bels.get(7))
                                                        .addComponent(scoreLa
bels.get(8))
                                                        .addComponent(scoreLa
bels.get(9))
                                                ))
                                        .addGroup(layout.createSequentialGrou
p()
                                                .addGap(132, 132, 132)
                                                .addComponent(btBackToGameMen
u)))
                                .addContainerGap(145, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
                layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                                .addContainerGap()
                                .addComponent(lbtitle)
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
```

```
                                .addComponent(scoreLabels.get(0))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(1))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(2))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(3))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(4))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(5))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(6))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(7))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(8))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED)
                                .addComponent(scoreLabels.get(9))
                                .addPreferredGap(LayoutStyle.ComponentPlaceme
nt.RELATED, 225, Short.MAX_VALUE)
                                .addComponent(btBackToGameMenu)
                                .addContainerGap())
        );
    }

    // instance variables:
    private final JButton btBackToGameMenu;
    private final JLabel lbtitle;
    private final JPanel pnViewScore;
    private final ArrayList<JLabel> scoreLabels;
}
```

.\src\main\java\exciting\system\DataLoader.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

import exciting.game.Player;
import exciting.util.Choice;
import exciting.util.Level;
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Scanner;

/**
 * Data Loader reads questions from a text file and player profiles from a
file
 * @author Leigh Chin
 */
public class DataLoader {
    /**
    * Converts a String to Difficulty Level enumerated type.  Default is
Novice.
    * @invariant The returned type will be valid.
    * @param s String
    * @return Difficulty Level
    */
    public Level ConvertToLevel(String s) {
        Level out = Level.NOVICE;

        switch (s) {
            case "NOVICE" : out = Level.NOVICE;
                            break;
            case "INTERMEDIATE" : out = Level.INTERMEDIATE;
                            break;
            case "ADVANCED" : out = Level.ADVANCED;
                            break;
        }
        return out;
    }

    /**
     * Converts a String to Choice enumerated type. Default is A.
     * @invariant The returned value will be valid.
     * @param s String
     * @return Choice
     */
    public Choice ConvertToChoice(String s) {
        Choice out = Choice.A;

        switch (s) {
            case "A" : out = Choice.A;
                       break;
            case "B" : out = Choice.B;
                       break;
            case "C" : out = Choice.C;
                       break;
            case "D" : out = Choice.D;
                       break;
        }
        return out;
    }

    /**
     * Loads questions from a data file
     * @param ql the QuestionList Object where the questions will be loaded
     * @precondition the question file exists in the expected location
     */
    public void loadQuestions(QuestionList ql) {
        String questionFilename = "questions.txt";

        try {
            File file = new File(questionFilename);
            Scanner sc = new Scanner(file);

            while (sc.hasNextLine()) {
                Question newQ = new Question();

                try {
                    newQ.setQ(sc.nextLine());
                    newQ.setA(sc.nextLine());
                    newQ.setB(sc.nextLine());
                    newQ.setC(sc.nextLine());
                    newQ.setD(sc.nextLine());
                    newQ.setCorrect(ConvertToChoice(sc.nextLine()));
                    newQ.setLevel(ConvertToLevel(sc.nextLine()));
                    ql.addQuestion(newQ);
                }
                catch (NoSuchElementException e) {
                // If the last question is incomplete, it will not
                // be added.  The previous questions are still intact.
                }
            }
        } catch (FileNotFoundException e) {
            // option here to display an error message
            // question list will be empty
        }
    }

    /**
     * Tries to load player data from player.dat
     * If there are errors or the file does not exist, null is returned
     * @return List of Player objects
     * @precondition the file must exist otherwise null is returned
     */
    public List<Player> loadPlayerInfo (){
        String filename = "players.dat";
        Player p = null;
        List<Player> pList = new ArrayList<>();
        boolean error = false;

        try {
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            while (!error) {
                try {
                    p = (Player)in.readObject();
                    pList.add(p);
                } catch (Exception e) {
                    // reached end of file
                    error = true;
                }
            }
            in.close();
            file.close();
        } catch (IOException e) {
            // any IO errors will result in an empty Player list being
returned
        }

        return pList;
    }
}
```

.\src\main\java\exciting\system\DataSaver.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

import exciting.game.Player;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.List;

/**
 * Data Saver writes the player profiles to a data file
 * @author Leigh Chin
 *
 */
public class DataSaver {

     /**
      * saves player data.  errors are caught here.  does not guarantee save
occurred.
      * @param players List of Player objects to save
      */
    public void savePlayers (List<Player> players) {

        try {
            String filename = "players.dat";
            FileOutputStream file = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file);

            for (Player p: players) {
                out.writeObject(p);
            }

            out.close();
            file.close();
        } catch (IOException e) {
            // possibiity here for displaying an error message
        }
    }

     /**
     * Deletes player data.Errors are caught here.
     */
    public void deletePlayers() {

        try {
            String filename = "players.dat";
            File file = new File(filename);
            file.delete();
        } catch (Exception e) {
            // possibiity here for displaying an error message
        }
    }
}
```

.\src\main\java\exciting\system\GameSystem.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

import exciting.game.Game;
import exciting.game.Player;
import exciting.game.RealPlayer;
import exciting.game.SimulatedPlayer;
import static exciting.util.Constant.*;
import exciting.util.Level;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * Game System manages main menu selections
 * @author Leigh Chin
 */
public class GameSystem {

    /**
```

```
     * constructor for GameSystem
     */
    public GameSystem() {
        this.listeners = new ArrayList<>();
        this.qList = new QuestionList();
        this.players = new ArrayList<>();
    }

    /**
     * Loads questions from disk, loads player data if it exists.
     * @postcondition the players list will contain 1 real and 2 simulated
players
     */
    public void initialize() {
        DataLoader dl = new DataLoader();
        dl.loadQuestions(qList);

        // try to load from file, if no file exists, null will be returned
        players = dl.loadPlayerInfo();

        if (players.isEmpty()) {
            // 1 real player and 2 simulated players will be added
            RealPlayer p = new RealPlayer("", Level.NOVICE);
            players.add(p);
            SimulatedPlayer pl = new SimulatedPlayer("Tom");
            players.add(pl);
            pl = new SimulatedPlayer("Jane");
            players.add(pl);
        }

        listeners.forEach((l) -> l.stateChanged(new ChangeEvent(this)));
    }

    /**
     * getNewGAme creates a Game object including the initialized players
list
     * and an iterator through the question list with appropriate difficulty
level
     * @precondition players list must have 3 players
     * @return Game object
     */
    public Game getNewGame() {
        Game g = new Game(qList.getQuestionIterator((q) -> {
            return q.getLevel().equals(getRealPlayer().getDifficulty());
        }), players);
        return g;
    }

    /**
     * getScore assembles a list of each players scores
     * @precondition players must contain players
     * @return List of Lists of scores
     */
    public List<List<Pair<Date, Integer>>> getScore() {
        List<List<Pair<Date, Integer>>> lst = new ArrayList<>();

        for (Player p : players) {
            lst.add(p.getScoreHistory(MAX_SCORE_HISTORY));
        }
        return lst;
    }

    /**
     * Removes all players data files and removes all players from the Game
     * @postcondition player array will be reset to default players
     */
    public void resetData() {
        DataSaver ds = new DataSaver();
        ds.deletePlayers();
        initialize(); // reset player list
        listeners.forEach((l) -> l.stateChanged(new ChangeEvent(this)));
    }

    /**
     * The difficulty level of the real player will be returned.
     * @precondition players list must contain a real player
     * @return Level of real player
     */
    public Level getDiff() {
        return getRealPlayer().getDifficulty();
    }

    /**
     * Changes the difficulty level of the real player.
     * @param lv Level to change to
     * @precondition The GameSystem must have a real player defined
     * @postcondition The real player's level will be changed
     */
    public void changeDifficulty(Level lv) {
        getRealPlayer().setDifficulty(lv);
    }

    /**
     * Finds the real player in the players list Returns null if player list
is
     * empty or no real player identified
     * @precondition players list must contain a real player
     * @return Player object
     */
    public Player getRealPlayer() {
        for (Player p : players) {
            if (p.getClass() == RealPlayer.class) {
                return p;
            }
        }

        return null;
    }

    /**
     * @param name Real player's name
     * @precondition the player list has 1 real player and 2 sims
     * @postcondition the real player's name will be set to name
     */
    public void setUserName(String name) {
        getRealPlayer().setName(name);
    }

    /**
     * returns true if real player established
     * @precondition real player object must exist
     * @return true if real player data is loaded
     */
    public boolean isUserDataExist() {
        return !getRealPlayer().getName().isBlank();
    }

    /**
     * Returns the name of the real player
     * @precondition player list must exist
     * @return String name of real player
     */
    public String getUserName() {
        return getRealPlayer().getName();
    }

    /**
     * Saves players data
     * @precondition the players list must be defined
     */
    public void saveUserData() {
        DataSaver ds = new DataSaver();
        ds.savePlayers(players);
    }

    /**
     * adds changeListener
     * @param listener
     */
    public void addChangeListener(ChangeListener listener) {
        listeners.add(listener);
    }

    private final QuestionList qList;
    private List<Player> players;
    private final ArrayList<ChangeListener> listeners;
}
```

.\src\main\java\exciting\system\Question.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

import exciting.util.Choice;
import exciting.util.Level;

/**
 * Defines Question object and associated methods
 * @author Leigh Chin
 */
public class Question {

    public Question() {
        question = null;
        answerA = null;
        answerB = null;
        answerC = null;
        answerD = null;
        correctAnswer = null;
        level = null;
    }

    /**
     * Sets the question text
     * @param Q
     */
    public void setQ (String Q) {question = Q;}

    /**
     * Sets Option A text
     * @param A
     */
    public void setA (String A) {answerA = A;}

    /**
     * Sets Option B text
     * @param B
     */
    public void setB (String B) {answerB = B;}

    /**
     * Sets Option C text
     * @param C
     */
    public void setC (String C) {answerC = C;}

    /**
     * Sets Option D text
     * @param D
     */
    public void setD (String D) {answerD = D;}

    /**
     * sets the correct answer choice
     * @param ch Choice correct answer
     */
    public void setCorrect (Choice ch) {correctAnswer = ch;}

    /**
     * Set the difficulty level for the question
     * @param lvl difficulty level
     */
    public void setLevel (Level lvl) {level = lvl;}

    /**
     * Constructs the Question string to display onscreen
     * @return String Question with Answer Choices
     */
    public String getQuestionText() {
        return this.question +
               "\n(A) " + this.getA() +
               "\n(B) " + this.getB() +
               "\n(C) " + this.getC() +
               "\n(D) " + this.getD();
    }

    /**
     * Returns question text
     * @return String question text
     */
    public String getQuestion() {
        return this.question;
    }

    /**
     * Returns option A text
     * @return String option A
     */
    public String getA() {
        return this.answerA;
    }

    /**
     * Returns option B text
     * @return String option B
     */
    public String getB() {
        return this.answerB;
    }

    /**
     * Returns option C text
     * @return String option C
     */
    public String getC() {
        return this.answerC;
    }

    /**
     * Returns option D text
     * @return String option D
     */
    public String getD() {
        return this.answerD;
    }

    /**
     * Returns correct answer choice
```

```
     * @return Choice correct answer
     */
    public Choice getCorrectAnswer() {
        return this.correctAnswer;
    }

    /**
     * Returns difficulty level of question
     * @return Level difficulty level of this question
     */
    public Level getLevel() {
        return this.level;
    }

    /**
     * Returns String of question and options
     * @return String of question, options, correct answer, and level
     */
    @Override
    public String toString() {
        String out;

        out = this.question + "\n" +
              this.answerA + "\n" +
              this.answerB + "\n" +
              this.answerC + "\n" +
              this.answerD + "\n" +
              this.correctAnswer + "\n" +
              this.level;

        return out;
    }

    private String question;
    private String answerA;
    private String answerB;
    private String answerC;
    private String answerD;
    private Choice correctAnswer;
    private Level level;
}
```

.\src\main\java\exciting\system\QuestionList.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

import java.util.ArrayList;
import java.util.ListIterator;
```

```
/**
 * Creates List of available questions
 * @author Leigh Chin
 */
public class QuestionList {

    /**
     * creates an iterator based on difficulty level
     * @param qs the question selector to determine if a question should be
used
     * @return an iterator that iterates through the question list
     */
    public ListIterator<Question> getQuestionIterator(QuestionSelector qs) {
        return new
            ListIterator<Question>() {
                public boolean hasNext() {
                    return current < list.size();
                }

                public Question next() {
                    Question q = null;

                    if (hasNext())
                        q = list.get(current++);

                    while ((q != null) && (!qs.isValidQuestion(q))) {
                        if (hasNext())
                            q = list.get(current++);
                        else
                            q = null;
                    }

                    return q;
                }

                public void remove() {
                    throw new UnsupportedOperationException();
                }

                public void reset() {
                    current = 0;
                }

                public void add(Question q) {
                    list.add(q);
                }

                public int previousIndex() {
                    throw new UnsupportedOperationException();
                }

                public int nextIndex() {
                    throw new UnsupportedOperationException();
                }

                public Question previous() {
                    throw new UnsupportedOperationException();
                }

                public boolean hasPrevious() {
                    throw new UnsupportedOperationException();
                }

                public void set(Question q) {
                    throw new UnsupportedOperationException();
                }

                private int current = 0;
            };
    }

    /**
     * Add question to the list
     * @param q Question to add
     */
    public void addQuestion(Question q) {
        list.add(q);
    }

    private ArrayList<Question> list = new ArrayList<>();
}
```

.\src\main\java\exciting\system\QuestionSelector.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.system;

/**
 * QuestionSelector determines if a question should be used for a game.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public interface QuestionSelector {

    /**
     * Check if a question should be selected for a game.
     *
     * @param q the question to be tested against
     * @return true if the question should be used, and false otherwise
     */
    public boolean isValidQuestion(Question q);
}
```

.\src\main\java\exciting\util\Choice.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.util;

/**
 * The options for a Choice, the correct answer to the question or the
 * choice that the player guesses.
 *
 * @author Leigh Chin
 */
public enum Choice {A, B, C, D};
```

.\src\main\java\exciting\util\Constant.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.util;

/**
 * Constant class describes all game parameter constants.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public final class Constant {

    public static final int MAX_SCORE_HISTORY = 10; // maximum number of
score history to be displayed
    public static final int MAX_PLAYER_NUM = 3; // maximum number of players
in a game
    public static final int MAX_NUM_CHOICES = Choice.values().length; //
maximum number of answer choices
    public static final int SCORE_FOR_ONE_ROUND = 1; // score for winner of a
round
    public static final int MAX_TIME_FOR_ONE_ROUND = 20000; // maximum time
in milliseonds to answer one round
    public static final int MIN_TIME_TO_ANSWER = 10000; // minimum time to
answer for simulated player
    public static final int MAX_TIME_TO_ANSWER = 15000; // maximum time to
answer for simulated player
    public static final int MAX_CHANCE = Level.values().length + 1; // affect
the chance the simulated player will return correct answer
}
```

.\src\main\java\exciting\util\GameState.java

```java
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.util;

/**
 * GameState describes the states of a game play.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public enum GameState {

    // enum constants are public, static and final by default
    GAME_BEGAN,
    INITIAL_GAME_SCREEN,
    QUESTION_ASKED,
    CURRENT_SCORE_UPDATED,
    CORRECT_ANSWER_REVEALED,
    SCORE_HISTORY_UPDATED,
    QUESTION_EXHAUSTED,
    GAME_ENDED;
}
```

.\src\main\java\exciting\util\Level.java

```java
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.util;

/**
 * Level describes all valid difficulty levels.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 */
public enum Level {

    // enum constants are public, static and final by default
    NOVICE,
    INTERMEDIATE,
    ADVANCED;
    @Override
    public String toString(){
        return this.name().substring(0, 1).toUpperCase() +
this.name().substring(1).toLowerCase();
    }

}
```

.\src\main\java\exciting\util\Pair.java

```java
package exciting.util;

import java.io.Serializable;

/**
 * Pair contains a key and a value.
 *
 * @author Tsz Shing Tsoi
 * @version 1.1
 * @param <K> a generic element
 * @param <V> a generic element
 */
public class Pair<K, V> implements Serializable, Cloneable {

    /**
     * Construct a Pair object.
     *
     * @param k a key
     * @param v a value
     * @precondition k != null && K implements Serializable && K is immutable
     * @precondition v != null && V implements Serializable && V is immutable
     */
    public Pair(K k, V v) {
        this.k = k;
        this.v = v;
    }

    /**
     * Return the key.
     *
     * @return the key
     */
    public K k() {
        return k;
    }

    /**
     * Return the value.
     *
     * @return the value
     */
    public V v() {
        return v;
    }
```

```java
/**
 * Test whether two objects are equal.
 *
 * @param other another object
 * @return true if two objects are equal in key and value, and false
 * otherwise
 */
@Override
public boolean equals(Object other) {
    if (this == other) {
        return true;
    }
    if (other == null) {
        return false;
    }
    if (getClass() != other.getClass()) {
        return false;
    } else {
        return k.equals(((Pair) other).k) && v.equals(((Pair) other).v);
    }
}

/**
 * Return the hashcode of the Pair object.
 *
 * @return the hashcode of the Pair object
 */
@Override
public int hashCode() {
    return 7 * k.hashCode() + 13 * v.hashCode();
}

/**
 * Convert the Pair object to a string representation.
 *
 * @return the string representation of the Pair object
 */
@Override
public String toString() {
    return getClass().getName()
            + "[key=" + k + ",value=" + v + "]";
}

/**
 * Clone the Pair object (shallow copy).
 *
 * @return a shallow clone of the Pair object
 * @throws CloneNotSupportedException if clone is not supported
 */
@Override
public Object clone() throws CloneNotSupportedException {
    return super.clone();
}

// instance variables
private final K k;
private final V v;
```

```
}
```

Junit Testing Source Codes

.\src\test\java\exciting\game\GameTest.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.system.Question;
import exciting.system.QuestionList;
import exciting.util.Choice;
import exciting.util.GameState;
import exciting.util.Level;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * GameTest tests the Game class.
 *
 * @author ShingShing
 */
public class GameTest {

    public GameTest() {
    }

    private Game game;
    private Player p1 = new SimulatedPlayer("Jane");
    private Player p2 = new RealPlayer("Tom", Level.INTERMEDIATE);
    private Player p3 = new RealPlayer("John", Level.ADVANCED);
    private Question q;
    private QuestionList qList;

    @Before
    public void setUp() {
        // setup question list
        q = new Question();
        q.setQ("This is a question.");
        q.setA("Choice a");
        q.setB("Choice b");
        q.setC("Choice c");
        q.setD("Choice d");
        q.setCorrect(Choice.A);
        q.setLevel(Level.INTERMEDIATE);
        qList = new QuestionList();
        qList.addQuestion(q);

        // setup player list
        p1 = new SimulatedPlayer("Jane");
        p2 = new RealPlayer("Tom", Level.INTERMEDIATE);
        p3 = new RealPlayer("John", Level.ADVANCED);
        Player[] arr = {p1, p2, p3};
        List<Player> pList = new ArrayList<>(Arrays.asList(arr));

        // setup game
        game = new Game(qList.getQuestionIterator((q) -> {
            return q.getLevel().equals(Level.INTERMEDIATE);
        }), pList);

    }

    /**
     * Test of initialize method, of class Game.
     */
    @Test

    public void testInitialize() {
        System.out.println("Test initialize");
        assertTrue(game.getState() == GameState.GAME_BEGAN);
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);
        assertTrue(p1.getDifficulty() == p2.getDifficulty());
    }

    /**
     * Test of getNextQuestion method, of class Game.
     */
    @Test
    public void testGetNextQuestion() {
        System.out.println("Test getNextQuestion");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        String q2 = game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);
        assertTrue(q2.equals(q.getQuestionText()));

        // get next question again
        q2 = game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);
        assertTrue(q2 == null);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        game.getResults();
        game.getCorrectAnswer();
        // get question in the next round
        q2 = game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_EXHAUSTED);
        assertTrue(q2 == null);
    }

    /**
     * Test of submitAnswerToJudge method, of class Game.
     */
    @Test
    public void testSubmitAnswerToJudge() {
        System.out.println("Test submitAnswerToJudge");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // no effect
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);

        // get results and compare
        List<Pair<String, Choice>> list1 = new ArrayList<>();
        list1.add(new Pair<String, Choice>("Jane", Choice.A));
        list1.add(new Pair<String, Choice>("Tom", Choice.B));
        list1.add(new Pair<String, Choice>("John", Choice.C));
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);
        assertTrue(game.getResults().equals(list1));
    }

    /**
     * Test of getResults method, of class Game.
     */
    @Test
    public void testGetResults() {
        System.out.println("Test getResults");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // no effect
        assertTrue(game.getResults() == null);
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);
        // no effect
        assertTrue(game.getResults() == null);
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);

        // get results and compare
        List<Pair<String, Choice>> list1 = new ArrayList<>();
        list1.add(new Pair<String, Choice>("Jane", Choice.A));
        list1.add(new Pair<String, Choice>("Tom", Choice.B));
        list1.add(new Pair<String, Choice>("John", Choice.C));
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);
        assertTrue(game.getResults().equals(list1));
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);
    }

    /**
     * Test of getCorrectAnswer method, of class Game.
     */
    @Test
    public void testGetCorrectAnswer() {
        System.out.println("Test getCorrectAnswer");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // no effect
        assertTrue(game.getCorrectAnswer() == null);
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // no effect
        assertTrue(game.getCorrectAnswer() == null);
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        // get correct answer and compare
        assertTrue(game.getCorrectAnswer() == Choice.A);
        assertTrue(game.getState() == GameState.CORRECT_ANSWER_REVEALED);
    }

    /**
     * Test of endGame method, of class Game.
     */
    @Test
    public void testEndGame() {
        System.out.println("Test endGame");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // no effect
        game.endGame();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // no effect
        game.endGame();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        // no effect
        game.endGame();
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        // get correct answer
        assertTrue(game.getCorrectAnswer() == Choice.A);
        assertTrue(game.getState() == GameState.CORRECT_ANSWER_REVEALED);

        // no effect
        game.endGame();
        assertTrue(game.getState() == GameState.SCORE_HISTORY_UPDATED);
    }

    /**
     * Test of cleanUp method, of class Game.
     */
    @Test
    public void testCleanUp() {
        System.out.println("Test cleanUp");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // no effect
        game.cleanUp();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // no effect
        game.cleanUp();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        // no effect
        game.cleanUp();
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        // get correct answer
        assertTrue(game.getCorrectAnswer() == Choice.A);
        assertTrue(game.getState() == GameState.CORRECT_ANSWER_REVEALED);

        // no effect
        game.cleanUp();
        assertTrue(game.getState() == GameState.CORRECT_ANSWER_REVEALED);

        // end the game
        game.endGame();
        assertTrue(game.getState() == GameState.SCORE_HISTORY_UPDATED);

        // clearn up
        game.cleanUp();
        assertTrue(game.getState() == GameState.GAME_ENDED);
    }

    /**
     * Test of getScore method, of class Game.
     */
    @Test
    public void testGetScore() {
        System.out.println("Test getScore");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        Integer[] arr = {0, 0, 0};
        List<Integer> list1 = new ArrayList<>(Arrays.asList(arr));

        // get scores
        assertTrue(game.getScore().equals(list1));

        // get next question
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);

        // get scores
        assertTrue(game.getScore().equals(list1));

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);
        assertTrue(game.getState() == GameState.CURRENT_SCORE_UPDATED);

        arr[0] = 1;
        list1 = new ArrayList<>(Arrays.asList(arr));

        // get scores
        assertTrue(game.getScore().equals(list1));
    }

    /**
     * Test of getPlayersEnteringGame method, of class Game.
     */
    @Test
    public void testGetPlayersEnteringGame() {
        System.out.println("Test getPlayersEnteringGame");
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);

        String[] arr = {"Jane", "Tom", "John"};
        List<String> list1 = new ArrayList<>(Arrays.asList(arr));

        assertTrue(game.getPlayersEnteringGame().equals(list1));
    }

    /**
     * Test of addChangeListener method, of class Game.
     */
    @Test
    public void testAddChangeListener() {
        System.out.println("Test addChangeListener");
        final String winnerMessage = "Jane is the winner!";
        final String[] str = {""};
        game.addChangeListener((e) -> {
            str[0] = winnerMessage;
        });

        game.initialize();
        game.getNextQuestion();

        // submit answers and end the current round
        game.submitAnswerToJudge(p1, Choice.A);
        game.submitAnswerToJudge(p2, Choice.B);
        game.submitAnswerToJudge(p3, Choice.C);

        assertTrue(str[0].equals(winnerMessage));
    }

    /**
     * Test of getState method, of class Game.
     */
    @Test
    public void testGetState() {
        System.out.println("Test getState");
        assertTrue(game.getState() == GameState.GAME_BEGAN);
        game.initialize();
        assertTrue(game.getState() == GameState.INITIAL_GAME_SCREEN);
        game.getNextQuestion();
        assertTrue(game.getState() == GameState.QUESTION_ASKED);
    }
}
```

.\src\test\java\exciting\game\JudgeTest.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.util.Choice;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.List;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * JudgeTest tests the Judge class.
 *
 * @author Tsz Shing Tsoi
 */
public class JudgeTest {

    public JudgeTest() {
    }

    private Judge judge;

    @Before
    public void setUp() {
        judge = new Judge();
    }

    /**
     * Test of setCorrectAnswer method, of class Judge.
     */
    @Test
    public void testSetCorrectAnswer() {
        System.out.println("Test setCorrectAnswer");
        judge.setCorrectAnswer(Choice.A);
        assertTrue(judge.getCorrectAnswer() == Choice.A);
    }

    /**
     * Test of getCorrectAnswer method, of class Judge.
     */
    @Test
    public void testGetCorrectAnswer() {
        System.out.println("Test getCorrectAnswer");
        assertTrue(judge.getCorrectAnswer() == null);
        judge.setCorrectAnswer(Choice.B);
        assertTrue(judge.getCorrectAnswer() == Choice.B);
    }
    /**
     * Test of setPlayerAnswer method, of class Judge.
     */
    @Test
    public void testSetPlayerAnswer() {
        System.out.println("Test setPlayerAnswer");
        Player p1 = new RealPlayer("Tom");
        Pair<Player, Choice> pair1 = new Pair<>(p1, Choice.A);
        judge.setPlayerAnswer(p1, Choice.A);
        Pair<Player, Choice> pair2 = judge.getPlayerAnswer().get(0);
        assertTrue(pair1.equals(pair2));
    }

    /**
     * Test of getPlayerAnswer method, of class Judge.
     */
    @Test
    public void testGetPlayerAnswer() {
        System.out.println("Test getPlayerAnswer");
        Player p1 = new RealPlayer("Tom");
        Player p2 = new SimulatedPlayer("Jane");
        Player p3 = new RealPlayer("John");

        List<Pair<Player, Choice>> list1 = new ArrayList<>();
        list1.add(new Pair<>(p1, Choice.A));
        list1.add(new Pair<>(p2, Choice.B));
        list1.add(new Pair<>(p3, Choice.C));

        judge.setPlayerAnswer(p1, Choice.A);
        judge.setPlayerAnswer(p2, Choice.B);
        judge.setPlayerAnswer(p3, Choice.C);
        List<Pair<Player, Choice>> list2 = judge.getPlayerAnswer();
        assertTrue(list1.equals(list2));
    }

    /**
     * Test of reset method, of class Judge.
     */
    @Test
    public void testReset() {
        System.out.println("Test reset");
        // initial settings
        assertTrue(judge.getCorrectAnswer() == null);
        assertTrue(judge.getPlayerAnswer().equals(new ArrayList<Pair<Player,
Choice>>()));
        assertTrue(judge.getWinner() == null);

        // set some parameters
        Player p1 = new RealPlayer("Tom");
        List<Pair<Player, Choice>> list1 = new ArrayList<>();
        list1.add(new Pair<>(p1, Choice.A));
        judge.setCorrectAnswer(Choice.A);
        judge.setPlayerAnswer(p1, Choice.A);
        assertTrue(judge.getCorrectAnswer() == Choice.A);
        assertTrue(judge.getPlayerAnswer().equals(list1));
        assertTrue(judge.getWinner() == p1);
        // reset
        judge.reset();
        assertTrue(judge.getCorrectAnswer() == null);
        assertTrue(judge.getPlayerAnswer().equals(new ArrayList<Pair<Player,
Choice>>()));
        assertTrue(judge.getWinner() == null);
    }

    /**
     * Test of getWinner method, of class Judge.
     */
    @Test
    public void testGetWinner() {
        System.out.println("Test getWinner");
        Player p1 = new RealPlayer("Tom");
        judge.setCorrectAnswer(Choice.A);
        judge.setPlayerAnswer(p1, Choice.A);
        assertTrue(judge.getWinner() == p1);
    }

    /**
     * Test of setChangeListener method, of class Judge.
     */
    @Test
    public void testSetChangeListener() {
        System.out.println("Test getWinner");
        final String winnerMessage = "Jane is the winner!";
        final String[] str = {""};
        judge.setChangeListener((e) -> {
            str[0] = winnerMessage;
        });

        judge.setCorrectAnswer(Choice.B);
        Player p1 = new RealPlayer("Tom");
        Player p2 = new SimulatedPlayer("Jane");
        Player p3 = new RealPlayer("John");
        judge.setPlayerAnswer(p1, Choice.A);
        judge.setPlayerAnswer(p2, Choice.B);
        judge.setPlayerAnswer(p3, Choice.C);

        assertTrue(str[0].equals(winnerMessage));
    }

}
```

.\src\test\java\exciting\game\RealPlayerTest.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;
```

```java
import exciting.util.Level;
import exciting.util.Pair;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * RealPlayerTest tests the RealPlayer class.
 *
 * @author Tsz Shing Tsoi
 */
public class RealPlayerTest {

    public RealPlayerTest() {
    }

    private Player p1;

    @Before
    public void setUp() {
        p1 = new RealPlayer("Tom");
    }

    /**
     * Test the setName() method.
     *
     */
    @Test
    public void testSetName() {
        System.out.println("Test setName");
        p1.setName("Jane");
        assertTrue(p1.getName().equals("Jane"));
    }

    /**
     * Test the getName() method.
     *
     */
    @Test
    public void testGetName() {
        System.out.println("Test getName");
        assertTrue(p1.getName().equals("Tom"));
    }

    /**
     * Test the getCurrentScore() method.
     *
     */
    @Test
    public void testGetCurrentScore() {
        System.out.println("Test getCurrentScore");
        assertTrue(p1.getCurrentScore() == 0);
        p1.incrementScore(1);
        assertTrue(p1.getCurrentScore() == 1);
    }

    /**
     * Test the incrementScore() method.
     *
     */
    @Test
    public void testIncrementScore() {
        System.out.println("Test incrementScore");
        p1.incrementScore(1);
        assertTrue(p1.getCurrentScore() == 1);
        p1.incrementScore(-1);
        assertTrue(p1.getCurrentScore() == 0);
    }

    /**
     * Test the resetCurrentScore() method.
     *
     */
    @Test
    public void testResetCurrentScore() {
        System.out.println("Test resetCurrentScore");
        p1.incrementScore(1);
        assertTrue(p1.getCurrentScore() == 1);
        p1.resetCurrentScore();
        assertTrue(p1.getCurrentScore() == 0);
    }

    /**
     * Test the updateScoreHistory() method.
     *
     */
    @Test
    public void testUpdateScoreHistory() {
        System.out.println("Test updateScoreHistory");
        Pair<Date, Integer> pair1 = new Pair<>(new Date(), 0); // #1
        p1.updateScoreHistory(); // #2
        Pair<Date, Integer> pair2 = p1.getScoreHistory(1).get(0);
        assertTrue(pair1.equals(pair2)); // assume execution time between
statements #1 and #2 are negligible
    }

    /**
     * Test the getScoreHistory() method.
     *
     */
    @Test
    public void testGetScoreHistory() {
        System.out.println("Test getScoreHistory");
        Pair<Date, Integer> pair1 = new Pair<>(new Date(), 0); // #1
        p1.updateScoreHistory(); // #2
        p1.incrementScore(1);
        Pair<Date, Integer> pair2 = new Pair<>(new Date(), 1); // #1
        p1.updateScoreHistory(); // #2

        List<Pair<Date, Integer>> list1 = new ArrayList<>();
        list1.add(pair1);
        list1.add(pair2);
        List<Pair<Date, Integer>> list2 = p1.getScoreHistory(2);
        assertTrue(list1.equals(list2)); // assume execution time between
statements #1 and #2 are negligible
    }

    /**
     * Test the setDifficulty() method.
     *
     */
    @Test
    public void testSetDifficulty() {
        System.out.println("Test setDifficulty");
        p1.setDifficulty(Level.ADVANCED);
        assertTrue(p1.getDifficulty() == Level.ADVANCED);
    }

    /**
     * Tests the getDifficulty() method.
     *
     */
    @Test
    public void testGetDifficulty() {
        System.out.println("Test getDifficulty");
        assertTrue(p1.getDifficulty() == Level.NOVICE);
    }
}
```

.\src\test\java\exciting\game\SimulatedPlayerTest.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package exciting.game;

import exciting.util.Choice;
import exciting.util.Constant;
import exciting.util.Level;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 * SimulatedPlayerTest tests the SimulatedPlayer class.
 *
 * @author Tsz Shing Tsoi
 */
public class SimulatedPlayerTest {

    public SimulatedPlayerTest() {
    }
    private SimulatedPlayer p1;

    @Before
    public void setUp() {
        p1 = new SimulatedPlayer("Tom");
    }

    /**
     * Test of setCorrectAnswer method, of class SimulatedPlayer.
     */
    @Test
    public void testSetCorrectAnswer() {
        System.out.println("Test setCorrectAnswer");
        System.out.println("  Skipped due to the lack"
                + " of accessor for correctAnswer. See \"Test getAnswer\"");
    }

    /**
     * Test of getAnswer method, of class SimulatedPlayer.
     */
    @Test
    public void testGetAnswer() {
        System.out.println("Test getAnswer");
        Choice ans = Choice.A;
        p1.setCorrectAnswer(ans);

        // test if the number of correct answers matches the expected number
        for (Level lv : Level.values()) {
            p1.setDifficulty(lv);
            int numRepetitions = 10000;
            int numCorrectAnswers = 0;
            double chanceOfCorrect = ((double) (lv.ordinal() + 1)) /
Constant.MAX_CHANCE;
            double expectedPercentCorrect = chanceOfCorrect + (1 -
chanceOfCorrect) * 1.0 / Constant.MAX_NUM_CHOICES;
            double delta = 0.05;
            for (int i = 0; i < numRepetitions; i++) {
                if (p1.getAnswer() == ans) {
                    numCorrectAnswers++;
                }
            }
            assertTrue(Math.abs(expectedPercentCorrect
                    - ((double) numCorrectAnswers) / numRepetitions)
                    <= delta);
        }
    }

}
```

.\src\test\java\exciting\system\DataLoaderTest.java

```
package exciting.system;

/*
```

```
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import exciting.game.Player;
import exciting.game.RealPlayer;
import exciting.game.SimulatedPlayer;
import exciting.util.Choice;
import exciting.util.Level;
import java.util.ArrayList;
import java.util.List;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Leigh Chin
 */
public class DataLoaderTest {

    public DataLoaderTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Test
    public void testConvertToLevel() {
        System.out.println("ConvertToLevel");
        DataLoader instance = new DataLoader();

        String s = "NOVICE";
        Level expResult = Level.NOVICE;
        Level result = instance.ConvertToLevel(s);
        assertEquals(expResult, result);

        s = "INTERMEDIATE";
        expResult = Level.INTERMEDIATE;
        result = instance.ConvertToLevel(s);
        assertEquals(expResult, result);

        s = "ADVANCED";
        expResult = Level.ADVANCED;
        result = instance.ConvertToLevel(s);
        assertEquals(expResult, result);
    }

    @Test
    public void testConvertToChoice() {
        System.out.println("ConvertToChoice");
        DataLoader instance = new DataLoader();

        String s = "A";
        Choice expResult = Choice.A;
        Choice result = instance.ConvertToChoice(s);
        assertEquals(expResult, result);

        s = "B";
        expResult = Choice.B;
        result = instance.ConvertToChoice(s);
        assertEquals(expResult, result);

        s = "C";
        expResult = Choice.C;
        result = instance.ConvertToChoice(s);
        assertEquals(expResult, result);

        s = "D";
        expResult = Choice.D;
        result = instance.ConvertToChoice(s);
        assertEquals(expResult, result);
    }

    @Test
    public void testLoadQuestions() {
        System.out.println("loadQuestions");
        DataLoader instance = new DataLoader();
        QuestionList ql = new QuestionList();
        instance.loadQuestions(ql);
        String result = "";
        if (ql != null) {
            result = ql.getQuestionIterator((q2) -> {
                return q2.getLevel().equals(Level.NOVICE);
            }).next().toString();
        }
        String expResult = "Which email service is owned by Microsoft?\n" +
"Hotmail\n"
                + "GMail\n" + "AOL Mail\n" + "Yahoo! Mail\n" + "A\n" +
"Novice";
        assertEquals(expResult, result);
    }

    @Test
    public void testLoadPlayerInfo() {
        System.out.println("loadPlayerInfo");
        DataLoader instance = new DataLoader();
        DataSaver ds = new DataSaver();
        List<Player> expResult = new ArrayList<>();

        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        expResult.add(p);
        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        expResult.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        expResult.add(sim);
        ds.savePlayers(expResult);

        List<Player> result = instance.loadPlayerInfo();
        assertEquals(expResult.get(0).getName(), result.get(0).getName());
        assertEquals(expResult.get(0).getDifficulty(),
result.get(0).getDifficulty());
    }
}
```

.\src\test\java\exciting\system\DataSaverTest.java

```
package exciting.system;

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


import exciting.game.Player;
import exciting.game.RealPlayer;
import exciting.game.SimulatedPlayer;
import exciting.system.DataSaver;
import exciting.system.DataLoader;
import exciting.util.Level;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Leigh Chin
 */
public class DataSaverTest {

    public DataSaverTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Test
    public void testSavePlayers() {
        System.out.println("savePlayers");
        DataSaver ds = new DataSaver();
        DataLoader dl = new DataLoader();
        List<Player> players = new ArrayList<>();
        List<Player> instance = new ArrayList<>();

        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance = dl.loadPlayerInfo();

        assertEquals(players.get(0).getName(), instance.get(0).getName());
        assertEquals(players.get(0).getDifficulty(),
instance.get(0).getDifficulty());
    }

    @Test
    public void testDeletePlayers() {
        System.out.println("deletePlayers");
        DataSaver instance = new DataSaver();
        String filename = "players.dat";
        List<Player> players = new ArrayList<>();

        File file = new File(filename);
        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        instance.savePlayers(players);

        boolean Before = file.exists();
        instance.deletePlayers();
        boolean After = file.exists();
        assertEquals(!Before,After);
    }
}
```

.\src\test\java\exciting\system\GamesystemTest.java

```
package exciting.system;

/*
 * To change this license header, choose License Headers in Project 
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


import exciting.game.Player;
import exciting.game.RealPlayer;
import exciting.game.SimulatedPlayer;
import exciting.util.Level;
import exciting.util.Pair;
import java.io.File;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
```

```
 *
 * @author Leigh Chin
 */
public class GameSystemTest {

    public GameSystemTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Test
    public void testInitialize() {
        DataSaver ds = new DataSaver();
        System.out.println("initialize");
        GameSystem instance = new GameSystem();
        List<Player> players = new ArrayList<>();

        // for testing, do savedata first which will guarantee file exists
        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance.initialize();

        String expResult = "Leigh";
        String result = instance.getUserName();
        assertEquals(expResult, result);

        ds.deletePlayers();
        instance.initialize();
        expResult = "";
        result = instance.getUserName();
        assertEquals(expResult, result);
```

```
}

@Test
public void testGetRealPlayer() {
    System.out.println("getRealPlayer");
    GameSystem instance = new GameSystem();
    DataSaver ds = new DataSaver();
    List<Player> players = new ArrayList<>();

    // save first to make sure the file exists
    // if file did not exist, null would be returned
    RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
    p.incrementScore(10);
    p.updateScoreHistory();
    p.incrementScore(15);
    p.updateScoreHistory();
    players.add(p);

    SimulatedPlayer sim = new SimulatedPlayer("Tom");
    sim.setDifficulty(Level.NOVICE);
    sim.incrementScore(5);
    sim.updateScoreHistory();
    players.add(sim);

    sim = new SimulatedPlayer("Jane");
    sim.setDifficulty(Level.INTERMEDIATE);
    sim.incrementScore(7);
    sim.updateScoreHistory();
    sim.incrementScore(4);
    sim.updateScoreHistory();
    players.add(sim);

    ds.savePlayers(players);
    instance.initialize();

    // if data was loaded the real player will be Leigh,
    // if data was reset first, real player will be ""
    assertEquals("Leigh", instance.getRealPlayer().getName());

    // if this is the first time the app is run, the user will have
    // to enter the player's name next
    ds.deletePlayers();
    instance.initialize();
    assertEquals("", instance.getRealPlayer().getName());
}

@Test
public void testGetScore() {
    System.out.println("getScore");
    GameSystem instance = new GameSystem();
    List<Player> players = new ArrayList<>();
    Integer expResult = 10;
    DataSaver ds = new DataSaver();

    // for testing, data must exist
    RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
    p.incrementScore(10);
```

```
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance.initialize();

        List<List<Pair<Date, Integer>>> result = instance.getScore();

        assertEquals(expResult, result.get(0).get(0).v());
    }

    @Test
    public void testResetData() {
        System.out.println("resetData");
        GameSystem instance = new GameSystem();

        String filename = "players.dat";
        File file = new File(filename);

        instance.resetData();
        boolean After = file.exists();
        assertEquals(false, After);
    }

    @Test
    public void testGetDiff() {
        System.out.println("getDiff");
        GameSystem instance = new GameSystem();
        Level expResult = Level.ADVANCED;
        List<Player> players = new ArrayList<>();
        DataSaver ds = new DataSaver();

        // for testing, data must exist
        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance.initialize();
        Level result = instance.getDiff();
        assertEquals(expResult, result);

        ds.deletePlayers();
        instance.initialize();
        expResult = Level.NOVICE;
        result = instance.getDiff();
        assertEquals(expResult, result);
    }

    @Test
    public void testChangeDifficulty() {
        System.out.println("changeDifficulty");
        Level lv = Level.INTERMEDIATE;
        GameSystem instance = new GameSystem();

        // for testing real player must be defined
        testSaveUserData();
        instance.initialize();

        instance.changeDifficulty(lv);
        assertEquals(lv, instance.getDiff());
    }

    @Test
    public void testGetUserName() {
        System.out.println("getUserName");
        GameSystem instance = new GameSystem();
        List<Player> players = new ArrayList<>();
        DataSaver ds = new DataSaver();


        // for testing, do saveuserdata first which will guarantee file
exists
        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance.initialize();

        String expResult = "Leigh";
        String result = instance.getUserName();
        assertEquals(expResult, result);
    }

    @Test
    public void testIsUserDataExist() {
        System.out.println("isUserDataExist");
        GameSystem instance = new GameSystem();
        List<Player> players = new ArrayList<>();
        DataSaver ds = new DataSaver();

        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        ds.savePlayers(players);
        instance.initialize();

        boolean expResult = true;
        boolean result = instance.isUserDataExist();
        assertEquals(expResult, result);
    }
    @Test
    public void testSetUserName() {
        System.out.println("setUserName");
        GameSystem instance = new GameSystem();
        testSaveUserData();
        instance.initialize();
        String name = "Mary";
        instance.setUserName(name);
        assertEquals(name, instance.getRealPlayer().getName());
    }

    @Test
    public void testSaveUserData() {
        System.out.println("saveUserData");
        GameSystem instance = new GameSystem();
        List<Player> players = new ArrayList<>();

        RealPlayer p = new RealPlayer("Leigh", Level.ADVANCED);
        p.incrementScore(10);
        p.updateScoreHistory();
        p.incrementScore(15);
        p.updateScoreHistory();
        players.add(p);

        SimulatedPlayer sim = new SimulatedPlayer("Tom");
        sim.setDifficulty(Level.NOVICE);
        sim.incrementScore(5);
        sim.updateScoreHistory();
        players.add(sim);

        sim = new SimulatedPlayer("Jane");
        sim.setDifficulty(Level.INTERMEDIATE);
        sim.incrementScore(7);
        sim.updateScoreHistory();
        sim.incrementScore(4);
        sim.updateScoreHistory();
        players.add(sim);

        instance.initialize();
        instance.saveUserData();

        String filename = "players.dat";
        File file = new File(filename);

        boolean exists = file.exists();

        assertEquals(true, exists);
    }

}
```

.\src\test\java\exciting\system\QuestionListTest.java

```
package exciting.system;
```

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import exciting.util.Choice;
import exciting.util.Level;
import java.util.ListIterator;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Leigh Chin
 */
public class QuestionListTest {

    public QuestionListTest() {
    }

    @Test
    public void testGetQuestionIterator() {
        System.out.println("getQuestionIterator");
        Level lvl = Level.NOVICE;
        QuestionList instance = new QuestionList();
        Question q = new Question();
        q.setQ("What is 2 + 2?");
        q.setA("2");
        q.setB("3");
        q.setC("4");
        q.setD("5");
        q.setCorrect(Choice.C);
        q.setLevel(Level.NOVICE);
        instance.addQuestion(q);
        String expResult = q.toString();
        ListIterator<Question> result = instance.getQuestionIterator((q2) ->
{
            return q2.getLevel().equals(lvl);
        });
        assertEquals(expResult, result.next().toString());
    }

    @Test
    public void testAddQuestion() {
        System.out.println("addQuestion");
        Question q = new Question();
        q.setQ("What is 2 + 2?");
        q.setA("2");
        q.setB("3");
        q.setC("4");
        q.setD("5");
        q.setCorrect(Choice.C);
        q.setLevel(Level.NOVICE);
        QuestionList instance = new QuestionList();
        instance.addQuestion(q);
        assertEquals(q.toString(),
                instance.getQuestionIterator((q2) -> {
                    return q2.getLevel().equals(Level.NOVICE);
                }).next().toString());
    }

}
```

.\src\test\java\exciting\system\QuestionTest.java

```
package exciting.system;

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */


import exciting.util.Choice;
import exciting.util.Level;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Leigh Chin
 */
public class QuestionTest {

    public QuestionTest() {
    }

    /**
     * Test of setQ method, of class Question.
     */
    @Test
    public void testSetQ() {
        System.out.println("setQ");
        String Q = "This is a test.";
        Question instance = new Question();
        instance.setQ(Q);
        assertEquals(Q, instance.getQuestion());
        // TODO review the generated test code and remove the default call to
fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of setA method, of class Question.
     */
    @Test
    public void testSetA() {
        System.out.println("setA");
        String A = "This is Option A.";
        Question instance = new Question();
        instance.setA(A);
        assertEquals(A, instance.getA());
        // TODO review the generated test code and remove the default call to
fail.
        //fail("The test case is a prototype.");
    }

    /**
     * Test of setB method, of class Question.
     */
    @Test
    public void testSetB() {
        System.out.println("setB");
        String B = "This is Option B.";
        Question instance = new Question();
        instance.setB(B);
        assertEquals(B, instance.getB());
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of setC method, of class Question.
     */
    @Test
    public void testSetC() {
        System.out.println("setC");
        String C = "This is Option C.";
        Question instance = new Question();
        instance.setC(C);
        assertEquals(C, instance.getC());
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of setD method, of class Question.
     */
    @Test
    public void testSetD() {
        System.out.println("setD");
        String D = "This is option D.";
        Question instance = new Question();
        instance.setD(D);
        assertEquals(D, instance.getD());
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of setCorrect method, of class Question.
     */
    @Test
    public void testSetCorrect() {
        System.out.println("setCorrect");
        Choice ch = Choice.B;
        Question instance = new Question();
        instance.setCorrect(ch);
        assertEquals(ch, instance.getCorrectAnswer());
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of setLevel method, of class Question.
     */
    @Test
    public void testSetLevel() {
        System.out.println("setLevel");
        Level lvl = Level.ADVANCED;
        Question instance = new Question();
        instance.setLevel(lvl);
        assertEquals(lvl, instance.getLevel());
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getQuestionText method, of class Question.
     */
    @Test
    public void testGetQuestionText() {
        System.out.println("getQuestionText");
        Question instance = new Question();
        instance.setQ("What is 1 + 1?");
        instance.setA("5");
        instance.setB("4");
        instance.setC("3");
        instance.setD("2");
        String expResult = "What is 1 + 1?\n(A) 5\n(B) 4\n(C) 3\n(D) 2";
        String result = instance.getQuestionText();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getQuestion method, of class Question.
     */
    @Test
    public void testGetQuestion() {
        System.out.println("getQuestion");
        Question instance = new Question();
        instance.setQ("What color is the sky?");
        String expResult = "What color is the sky?";
        String result = instance.getQuestion();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getA method, of class Question.
     */
    @Test
    public void testGetA() {
        System.out.println("getA");
        Question instance = new Question();
        instance.setA("red");
        String expResult = "red";
        String result = instance.getA();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getB method, of class Question.
     */
    @Test
    public void testGetB() {
        System.out.println("getB");
        Question instance = new Question();
        instance.setB("blue");
        String expResult = "blue";
        String result = instance.getB();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getC method, of class Question.
     */
    @Test
    public void testGetC() {
        System.out.println("getC");
        Question instance = new Question();
        instance.setC("green");
        String expResult = "green";
        String result = instance.getC();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getD method, of class Question.
     */
    @Test
    public void testGetD() {
        System.out.println("getD");
        Question instance = new Question();
        instance.setD("yellow");
        String expResult = "yellow";
        String result = instance.getD();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getCorrectAnswer method, of class Question.
     */
    @Test
    public void testGetCorrectAnswer() {
        System.out.println("getCorrectAnswer");
        Question instance = new Question();
        instance.setCorrect(Choice.C);
        Choice expResult = Choice.C;
        Choice result = instance.getCorrectAnswer();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of getLevel method, of class Question.
     */
    @Test
    public void testGetLevel() {
        System.out.println("getLevel");
        Question instance = new Question();
        instance.setLevel(Level.INTERMEDIATE);
        Level expResult = Level.INTERMEDIATE;
        Level result = instance.getLevel();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

    /**
     * Test of toString method, of class Question.
     */
    @Test
    public void testToString() {
        System.out.println("toString");
        Question instance = new Question();
        instance.setQ("What is 1 + 1?");
        instance.setA("5");
        instance.setB("4");
        instance.setC("3");
        instance.setD("2");
        instance.setCorrect(Choice.D);
        instance.setLevel(Level.NOVICE);
        String expResult = "What is 1 + 1?\n5\n4\n3\n2\nD\nNovice";
        String result = instance.toString();
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to
fail.
        // fail("The test case is a prototype.");
    }

}
```