FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Kahoot!

[7507/9502] Algoritmos y Programación III
Grupo N8 - Curso 2
Primer cuatrimestre de 2020

Alumnos	Padrón	Email
Escandar, German	105250	gescandar@fi.uba.ar
Zardain, Javier	102521	jzardain@fi.uba.ar
Guerra, Santiago	103607	sguerra@fi.uba.ar
Sapunar, Petri Miroslav	95060	psapunar@fi.uba.ar
De Sousa, Nicolas	94957	ndesousa@fi.uba.ar

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un juego de preguntas y respuestas que califica correctamente lo que responda cada jugador en Java utilizando los conceptos del paradigma de la orientación a objetos vistos hasta ahora en el curso.

2. Supuestos

Se debe tener en cuenta las siguientes premisas para poder utilizar correctamente el programa:

- El programa consiste en la competición entre dos jugadores.
- Cada jugador solo tendrá una oportunidad para contestar correctamente una pregunta que se muestre en pantalla.
- Cada turno finalizará cuando cada jugador conteste la pregunta mostrada en pantalla una vez.
- Cada jugador posee a su disposición: un ”multiplicador x2”, ”un multiplicador x3” y dos opciones de ”exclusividad de puntaje”.
- Para los casos en que se aplique la ”exlusividad de puntaje” para preguntas con puntaje parcial, se considera ganador al jugador que obtenga mayor puntaje, en el caso que ambos obtengan el mismo puntaje no sumarán puntos.
- Se considera la partida como terminada cuando la última pregunta es respondida por ambos jugadores.
- Los jugadores pueden tener puntaje negativo.

3. Diagramas de clase

El programa utiliza el patrón de diseño MVC para organizar las clases que componen toda la aplicación. En los próximos diagramas se puede observar como se realizaron la implementación de las clases relacionadas al modelo:

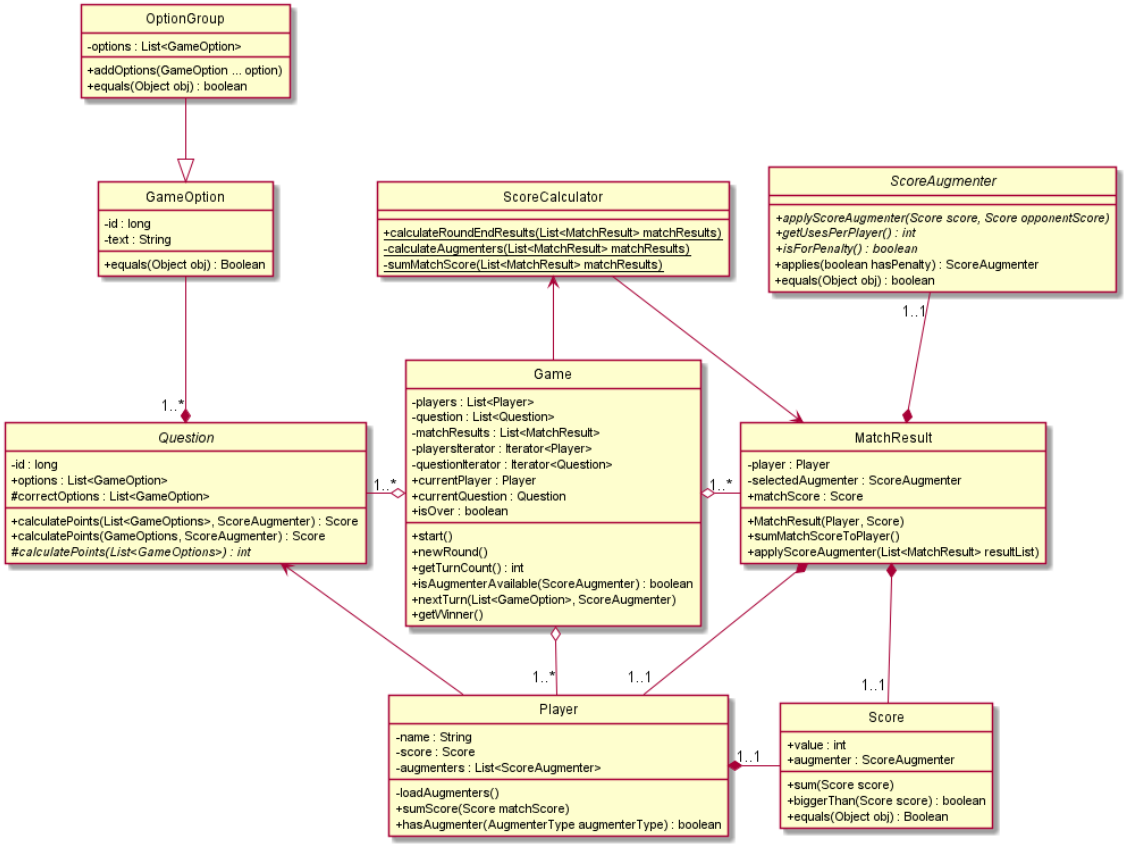

Figura 1: Diagrama de Clases Principal.

El funcionamiento del juego está contenido en la clase Game, que guarda como atributo una lista de jugadores, que se van agregando a través de la interfaz gráfica. Además contiene una lista de preguntas, que se instanciará y llenará parseando un archivo json. Los jugadores están representados por la clase Player y cada uno tendrá un puntaje que serán instancias de la clase Score. Las preguntas serán todas instancias especificas de la clase abstracta Question, pudiendo dividirse en 6 tipos de preguntas distintas (ver diagrama en detalle de la clase Question).

Cada pregunta tendrá opciones entra las cuales el jugador tendrá que elegir la/las correcta/s, esas opciones son todas instancias de la clase GameOption. Para el caso particular de las preguntas GroupChoiceQuestion en el cual deben clasificarse las opciones dentro de grupos, se utiliza la clase OptionGroup para representar dichos grupos.

Una vez que ambas listas estén llenas (preguntas y jugadores), se podrá comenzar el juego con el método start(). Luego, cada vez que un jugador de la lista responda, terminará su turno con el método nextTurn(...), y pasará a responder el próximo jugador de la lista. Esto continuará hasta que todos los jugadores hayan respondido todas las preguntas. Cuando esto suceda, el juego se considera como terminado.

ScoreCalculator y MatchResult son las clases que permiten calcular los puntajes finales de cada jugador al final de un turno. MatchResult es donde se encuentra el resultado obtenido por el

jugador al responder la pregunta y donde se almacena el multiplicador seleccionado por el jugador. A medida que cada jugador realiza su selección, una instancia de la clase MatchResult es añadida a una lista. Luego, cuando termina la ronda, ScoreCalculator recorre la lista de MatchResult aplicando los multiplicadores y sumando finalmente el puntaje final a cada jugador. Se realiza de este modo ya que determinados multiplicadores alteran el resultado de acuerdo a la relación de la respuesta del jugador con la de otros jugadores.

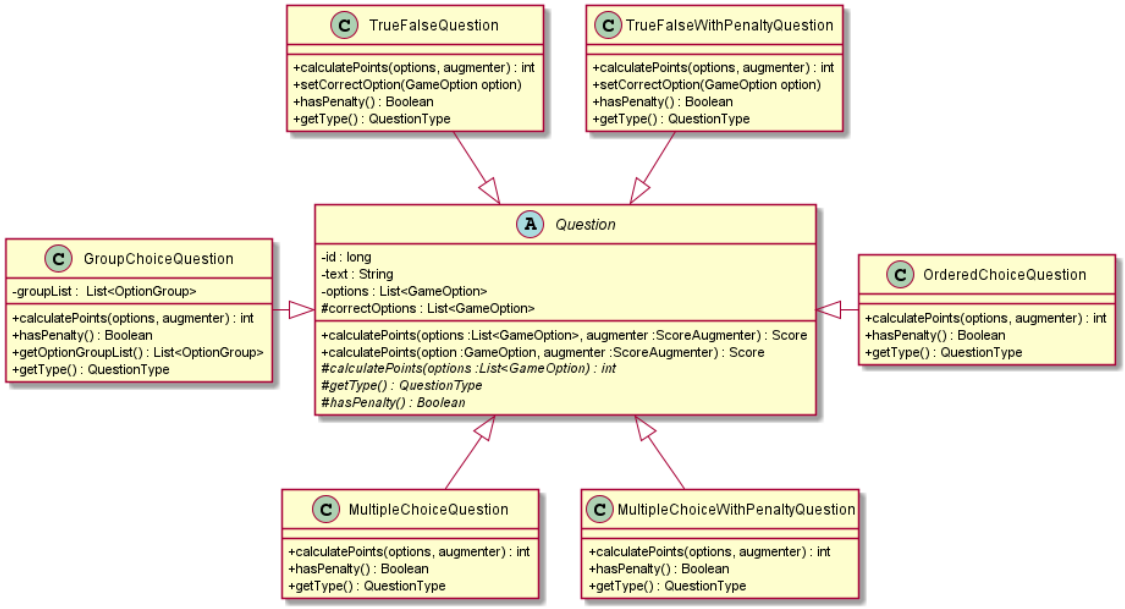

Figura 2: Diagrama en detalle de la clase Question y sus clases hijas.

Las preguntas serán distintas instancias especificas de las clases hijas de Question, siendo ellas TrueFalseQuestion, MultipleChoiceQuestion, OrderedChoiceQuestion, GroupChoiceQuestion, y las preguntas con penalidad, TrueFalseWithPenaltyQuestion y MultipleChoiceWithPenaltyQuestion. Todas esas preguntas implementan métodos específicos de la clase abstracta para responder a distintos escenarios de forma polimórfica.

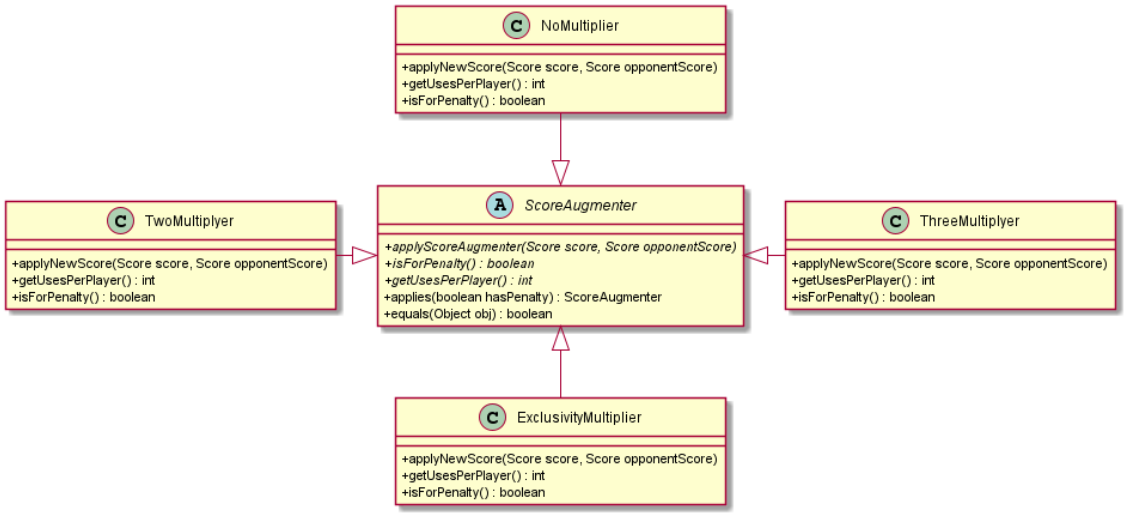

Figura 3: Diagrama en detalle de la clase ScoreAugmenter y sus clases hijas.

De forma similar a lo que sucede con las preguntas, el mismo diseño se aplica para los modificadores de puntaje, siendo estos instancias de las clases hijas de ScoreAugmenter. Estas son NoMultiplier, TwoMultiplier, ThreeMultiplier y ExclusivityMultiplier.

4. Diagramas de estado

El siguiente diagrama de estado muestra como el juego avanza desde que arranca, se cargan a los jugadores, se contestan a las preguntas, muestran el ganador por pantalla y se finaliza el juego.

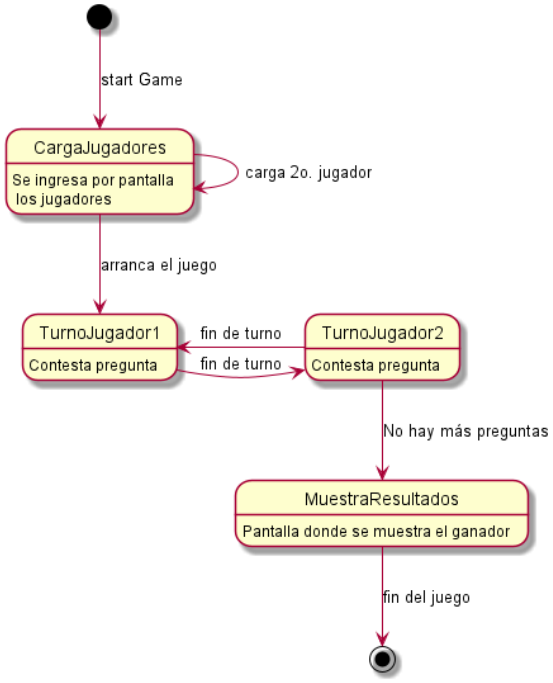

Figura 4: Diagrama de estado para funcionamiento del juego.

5. Diagramas de paquetes

A modo de ilustrar los distintos paquetes y desarrollo del juego se creó el siguiente diagrama de paquetes con algunas clases relevantes para el funcionamiento del programa:

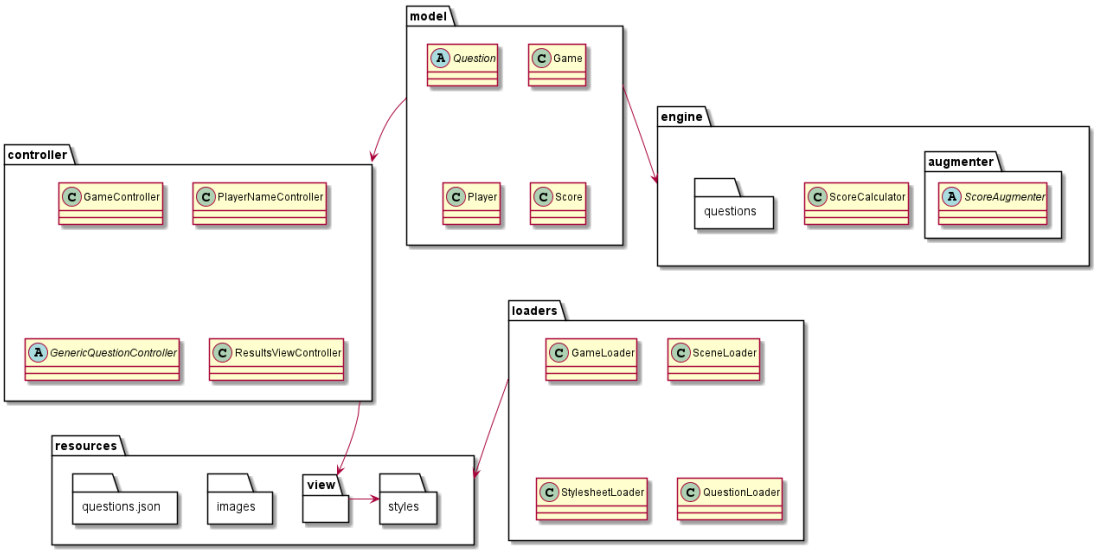

Figura 5: Diagrama de Paquetes

6. Detalles de implementación

6.1. Turnos del juegos y contestar una pregunta

En cada turno los jugadores deben contestar las preguntas que le aparecen en pantalla utilizando su cursor. Las preguntas serán enviadas a la clase Game mediante el método nextTurn, pudiendo ser una opción individual o una lista dependiendo de la pregunta.

En caso de que no se utilice un modificador de puntaje para la pregunta, en el método se invocará otra vez a un método nextTurn pero el cual reciba por parámetro las opciones elegidas y un objeto de la clase NoMultiplier. Si se aplica un modificador de puntaje especifico, este se pasará por parámetro al método nextTurn.

En nextTurn se observa la siguiente implementación:

```
public void nextTurn(List<GameOption> selectedOptions, ScoreAugmenter augmenter){
    Score matchScore = currentQuestion.calculatePoints(selectedOptions, augmenter);

    matchResults.add(new MatchResult(currentPlayer, matchScore));
        if(playersIterator.hasNext()){
            currentPlayer = playersIterator.next();
        }
        else if(!isOver){
            ScoreCalculator.calculateRoundEndResults(matchResults);

            if(questionIterator.hasNext()){
                currentQuestion = questionIterator.next();
                newRound();
            }else {
```

```
                isOver = true;
            }
        }
}
```

Al finalizar el turno del jugador se calcula su puntaje consultando a la pregunta actual si las opciones elegidas son correctas. Como dependiendo del tipo de pregunta se utilizan específicos modificadores de puntaje, el mismo se ingresa por parámetro para conocer si es posible o no su utilización.

El puntaje obtenido por cada jugador al contestar una pregunta se guarda como un objeto Score que en este método se instancia como matchScore del jugador. matchScore tendrá como atributos su puntaje y el modificador de puntaje (augmenter) ingresado previamente en el calculo de la pregunta.

Como se delega en la pregunta si es valido o no el modificador de puntaje, el método calculatePoints lo vuelve a delegar al augmenter recibido al crear una nueva instancia de Score, consultando su validez con el metodo applies:

```
public ScoreAugmenter applies(boolean hasPenalty) {
    if(hasPenalty == isForPenalty()) {
    return this;
    }
    return new NoMultiplier();
}
```

Al ser necesario conocer todos los resultados para finalmente calcular cual es el puntaje obtenido por cada jugador, se utiliza la lista de resultados de partidas (matchResults) y se invoca el método estático calculateRoundEndResults de ScoreCalculator.

La clase ScoreCalculator tiene 3 metodos estáticos, los cuales uno es el método publico invocado y otros son dos metodos privados: Uno para calcular la variación de puntajes considerando los multiplicadores y exclusividades de puntaje, y otro para sumar al puntaje de cada jugador el obtenido en la actual ronda.

El metodo calculateAugmenters recorre la lista de matchResults del turno actual y delega en cada resultado de partida el calculo del modificador de puntaje, enviando como parametro la lista de resultados.

Para la clase MatchResult el metodo applyScoreAugmenter recorre de vuelta el vector de matchResults pero ahora desde cada uno de los resultados, delegando en cada ScoreAugmenter la responsabilidad de modificar el puntaje suyo y de los otros jugadores en partida por el siguiente método:

```
private static void calculateAugmenters(List<MatchResult> matchResults) {
    matchResults.stream().forEach(result -> {
        result.applyScoreAugmenter(matchResults);
    });
}
```

El método applyScoreAugmenter de ScoreAugmenter es entendido por todas las clases hijas (TwoMultiplier, ThreeMultiplier, ExclusivityMultiplier y NoMultiplier), implementándose un comportamiento especifico para cada una pero entendiendo siempre el mismo mensaje.

Finalmente, para modificar el puntaje de cada jugador en partida se invoca el método sumMatchScore en el método calculateRoundEndResults, recorriendo a todos los jugadores y cargando su nuevo puntaje.

7. Excepciones

Exception FileNotFoundException Es lanzada cuando algún recurso estático no pudo ser hallado.

Exception QuestionsNotLoadedException Indica que las preguntas no pudieron ser cargadas desde el origen previsto en la aplicación.

Exception StylesheetLoadingException Indica que una hoja de estilos utilizada por las vistas de JavaFX no pudo ser cargada.

Exception ViewLoadingException Esta excepción es lanzada si ocurre algún problema al querer cargar la vista de JavaFX.

Exception ViewNotFoundException Indica que una vista no pudo ser encontrada en el origen esperado.

Error FatalError Error en tiempo de ejecución que es lanzado si sucede una situación inmanejable que debe interrumpir el flujo de la aplicación, es utilizada si no es posible cargar la pantalla que informa al usuario que ocurrió un error.

8. Diagramas de secuencia

Para observar las funcionalidades del programas se hicieron distintos diagramas de secuencia para analizar como los jugadores interactúan con las preguntas, como se relacionan los metodos del juego, jugador y el puntaje para que se asignen correctamente a quien contestó correctamente.

Inicialmente, se muestra en un diagrama como funciona el programa de manera simplificada:

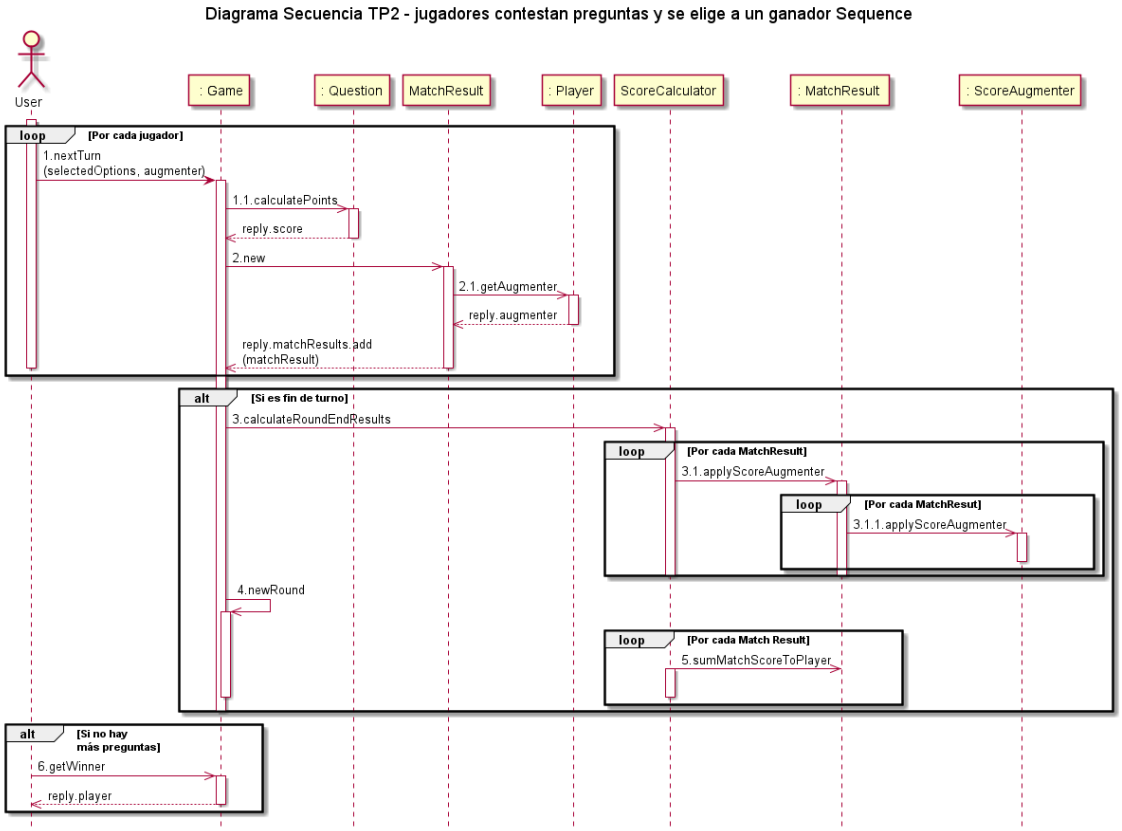

Figura 6: Diagrama de secuencia del juego

En los diagramas siguientes se analizan de forma detallada como interactúan los métodos y clases:

8.1. Jugador contesta pregunta

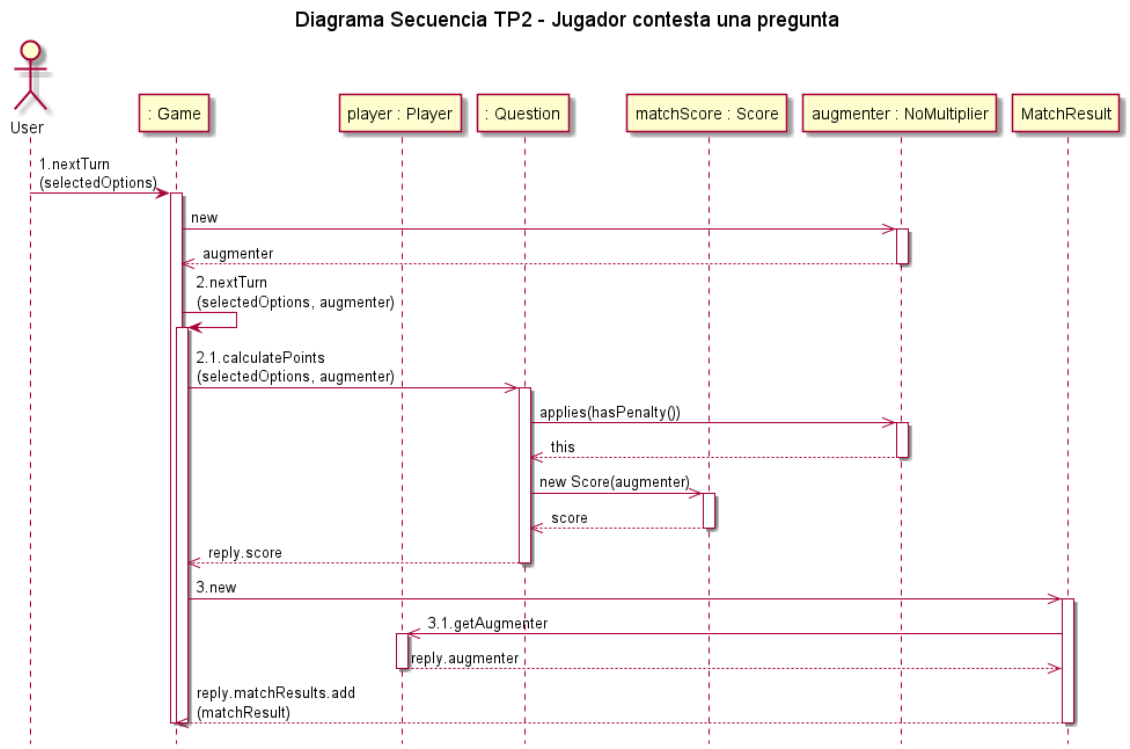

Figura 7: Diagrama de secuencia jugador contesta pregunta.

Cuando un jugador contesta una pregunta sin uso del augmenter se instancia uno de la clase NoMultiplier.

8.2. Jugador contesta pregunta utilizando Augmenter

La siguiente figura describe paso por paso la acción realizada por un jugador cualquiera al responder una pregunta, con una lista de las respuestas seleccionadas y el multiplicador de puntaje elegido.

Como se mencionó anteriormente, primero se calcula el puntaje del jugador teniendo en cuenta solamente sus respuestas. Con ese puntaje se instancia un objeto MatchResult, que se agrega a la lista de resultados de la ronda.

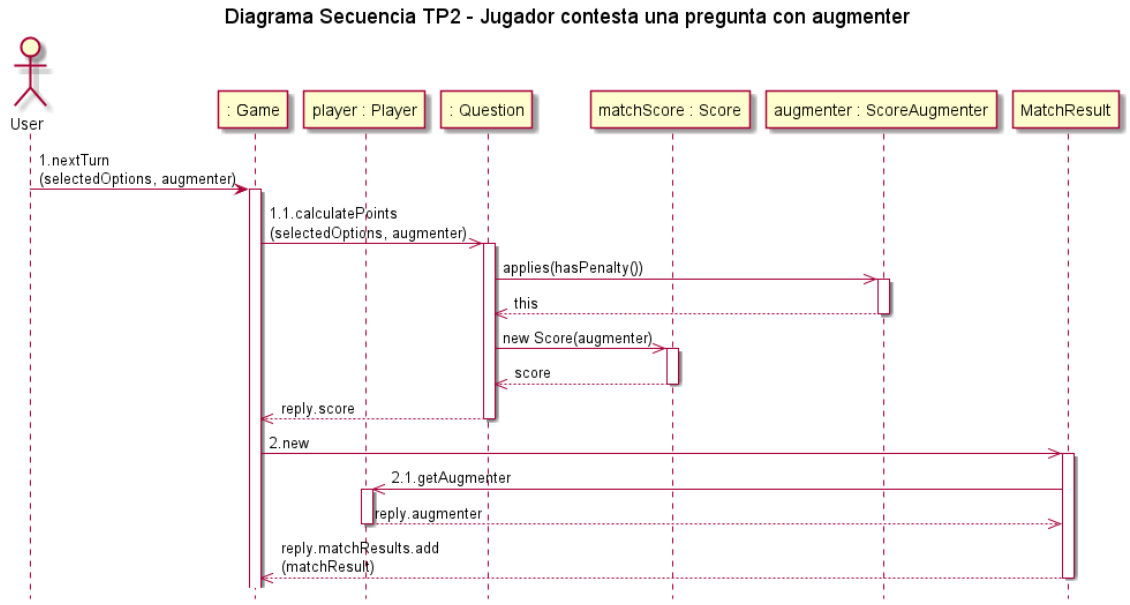

Figura 8: Diagrama de secuencia Jugador contesta una pregunta con augmenter.

8.3. Aplicación de multiplicador de exclusividad

Este último diagrama muestra como se aplica el multiplicador de exclusividad cuando alguno de los jugadores la haya elegido, y evidencia la necesidad de calcular los puntajes finales al final de la ronda, pues dicho puntaje dependerá de los puntajes parciales de los dos jugadores que se esté comparando.

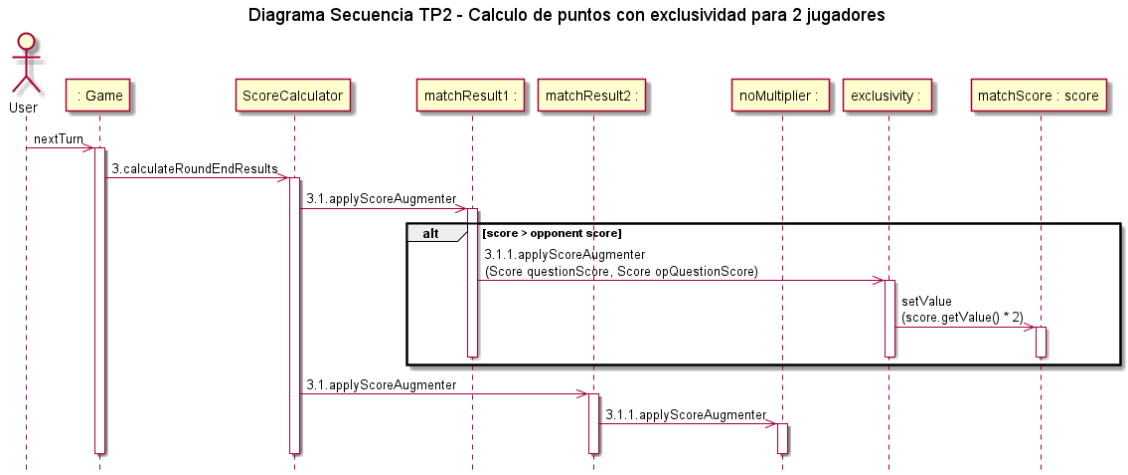

Figura 9: Diagrama de secuencia para exclusividad.