FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2 — Kahoot

[7507/9502] Algoritmos y Programación III
Curso 2
Grupo: N14
Primer cuatrimestre de 2020

Integrante		Padrón		Correo electrónico
Ferrari Coronel, Mateo	—	# 102375	—	mateo.ferrari97@gmail.com
Xifro, Juan Bautista	—	# 101717	—	bauxifro@gmail.com
Rodriguez, Agustín	—	# 101570	—	agrodriguez@fi.uba.ar
Rial, Tadeo Ezequiel	—	# 104677	—	trial@fi.uba.ar
Untrojb, Kevin	—	# 97866	—	oliverk12@hotmail.com

Índice

1. Introducción

El presente informe expone la documentación de la solución del Segundo trabajo práctico de la materia Algoritmos y Programación III. El trabajo tiene como objetivo desarrollar una aplicación de manera grupal aplicando todos los conceptos de la programación orientada a objetos vistos en clase. Se debe utilizar el lenguaje de programación java y trabajar con tecnicas de *TDD* e integración continua.

2. Especificaciones

Se brindaron las especificaciones que debe cumplir la aplicación a desarrollar. La aplicación consiste en un juego por turnos, de dos jugadores conformado de un panel en el cual se mostraran preguntas con múltiples opciones de respuesta. Cada pregunta será mostrada dos veces, una vez para cada jugador (al estar jugando los dos en la misma computadora, será responsabilidad de cada jugador no mirar la pantalla mientras el otro responde). El jugador dispone de un tiempo limitado para responder cada pregunta. Existen varios tipos de preguntas, que asignan puntaje en forma diferenciada a cada jugador dependiendo de cómo responde cada uno. También existen opciones como los multiplicadores y la exclusividad de puntaje que cada jugador puede utilizar para mejorar sus oportunidades de obtener puntos. El objetivo del juego es lograr más puntos que el otro jugador respondiendo correctamente las preguntas.

3. Supuestos

Se consideraron varios supuestos que no están contempladas en especificaciones y se adoptaron para poder realizar el proyecto

- Las preguntas van a ser definidas por nosotros
- Hay empate
- Una vez que el usuario elige una opción, no hay vuelta atrás.
- El puntaje mínimo de cada jugador es cero, no puede ser negativo.
- Si se acaba el tiempo no se otorga puntaje en esa pregunta.
- No se puede elegir primero un multiplicador x2 y después x3 o viceversa.
- Una vez que el usuario activa un multiplicador, no hay vuelta atrás.

4. Planificación

Para poder llevar a cabo este trabajo se especificó una serie de entregas en durante el transcurso de tiempo de implementación del proyecto.

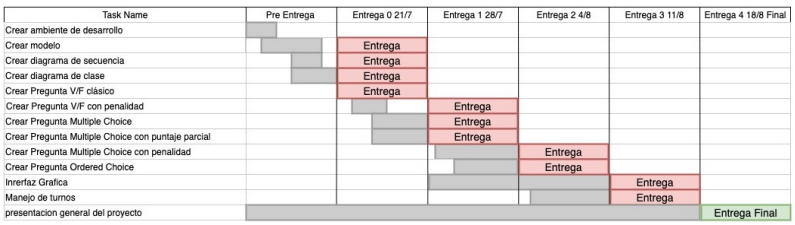

Figura 1: Diagrama de grantt con la planificación general del proyecto dividido en entregas.

en cada etapa se realizo un *roadmap* diferente de que tareas realizar y la forma de continuar el proyecto

5. Diagramas de clase

En esta seccion se presentan los distintos diasgramas de clases utilizados en el diseño de trabajo practico.

Diagrama general de las relaciones que tiene la clase Question.

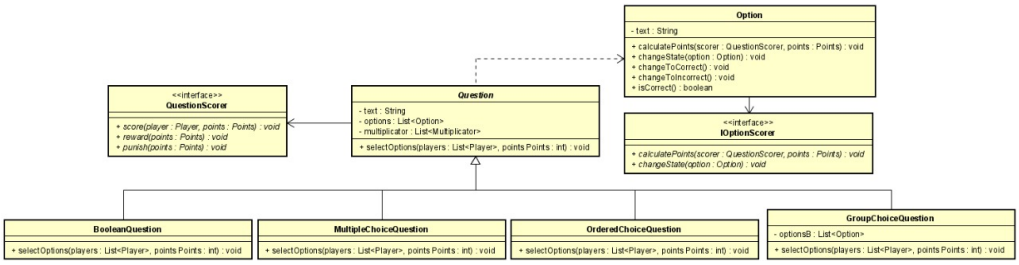

Figura 2: Diagrama de clase de *Question*.

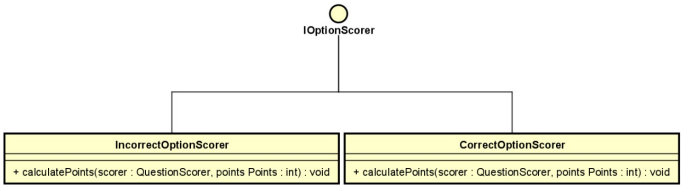

Figura 3: Diagrama de clase de a interfaz *IOptionScorer* .

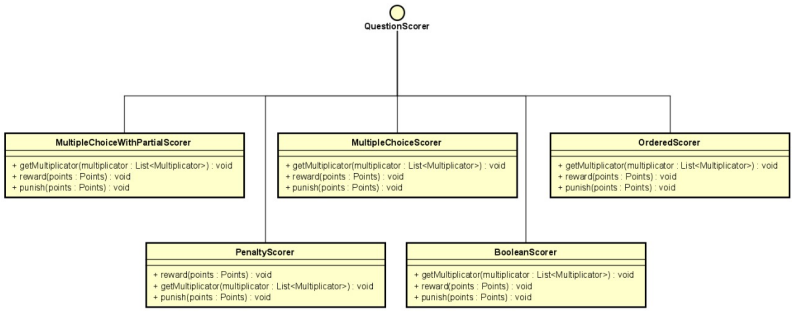

Figura 4: Diagrama de clase del *Question socrer*.

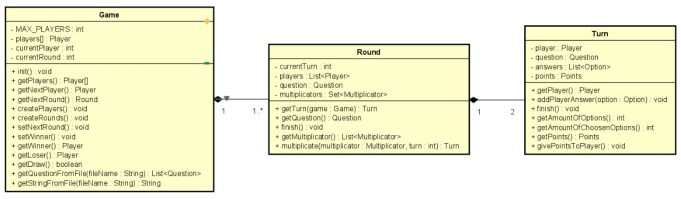

Figura 5: Diagrama de clase de *Game* .

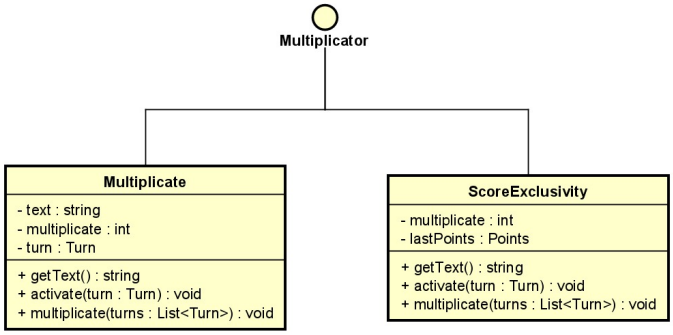

Figura 6: Diagrama de clase de la interfaz *Multiplicators* .

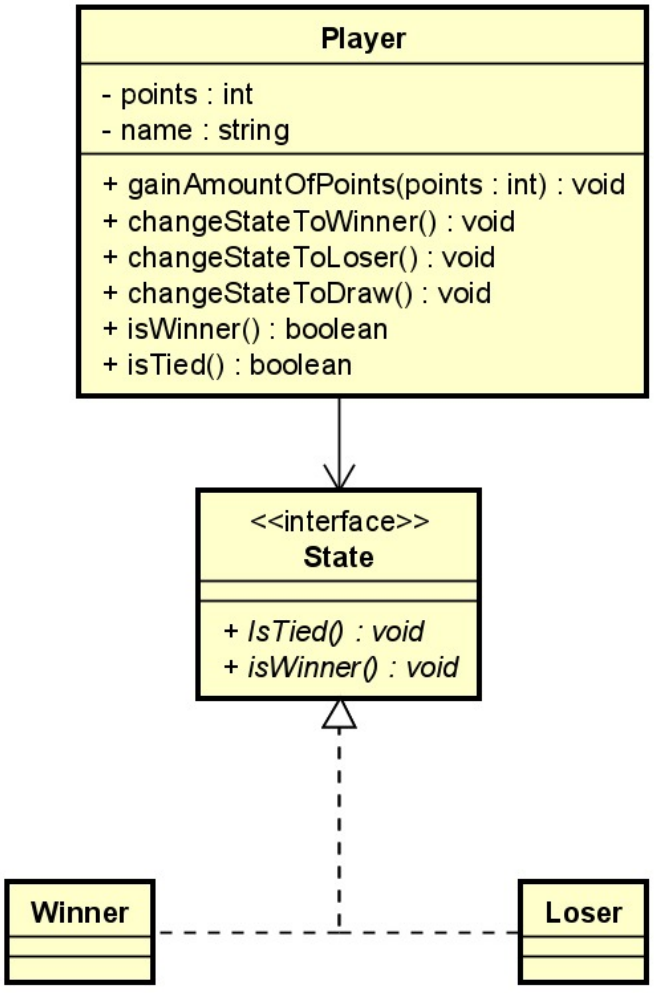

Figura 7: Diagrama de clase de *Player* .

6. Diagramas de secuencia

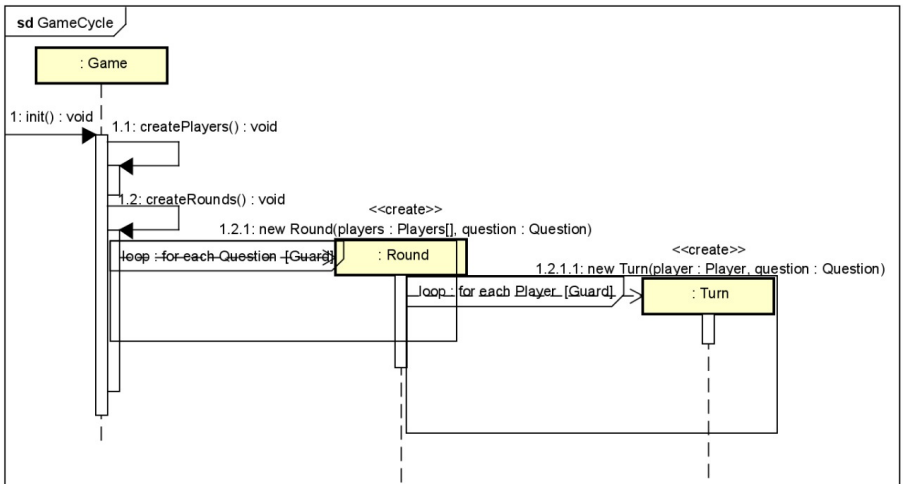

Figura 8: Diagrama de secuencia del comienzo del juego.

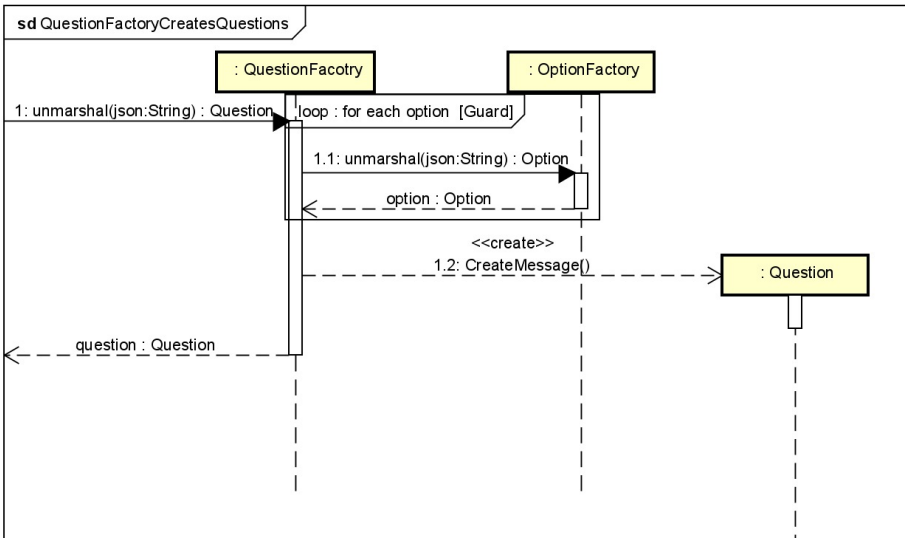

Figura 9: Diagrama de secuencia de como se levantan las preguntas desde un JSON.

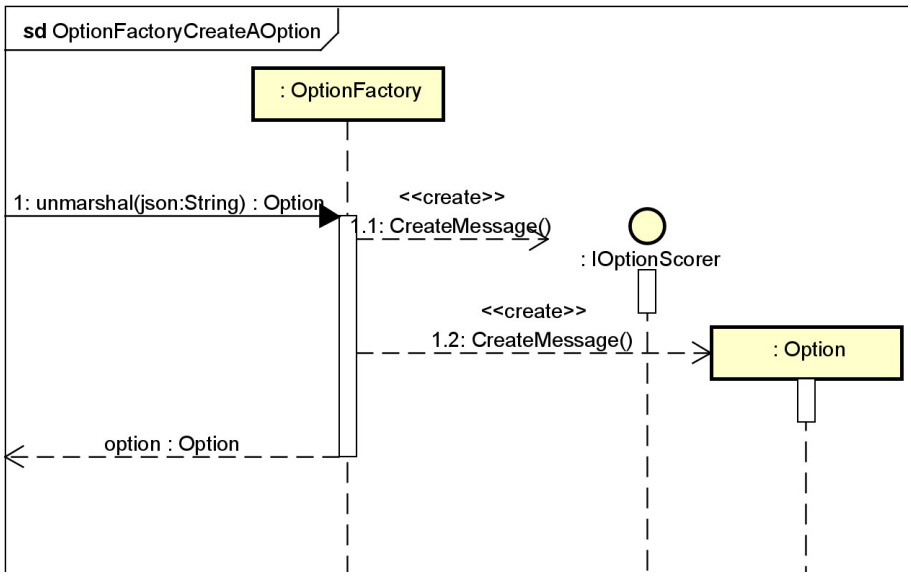

Figura 10: Diagrama de secuencia de como se levantan las opciones desde un JSON.

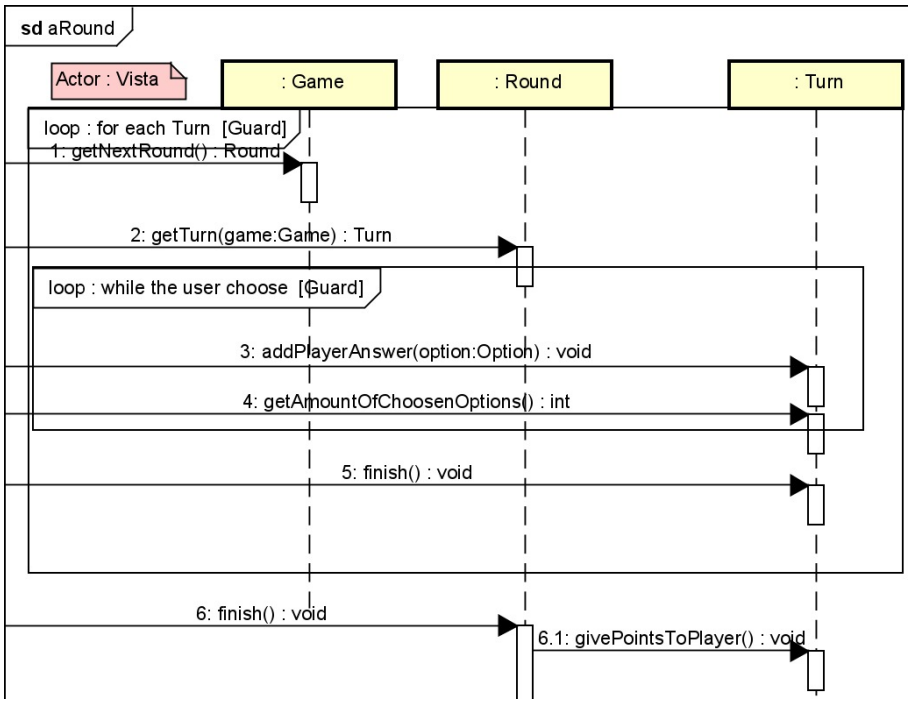

Figura 11: Diagrama de secuencia de como la vista se comunica con el modelo para generar una ronda.

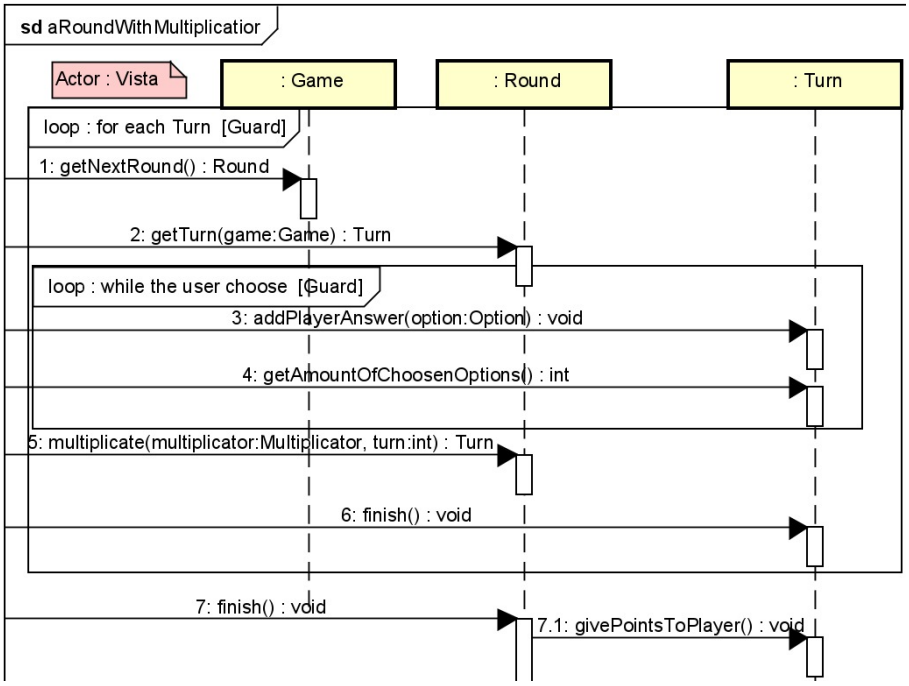

Figura 12: Diagrama de secuencia de como la vista se comunica con el modelo para generar una ronda en la cual se multiplican los puntos.

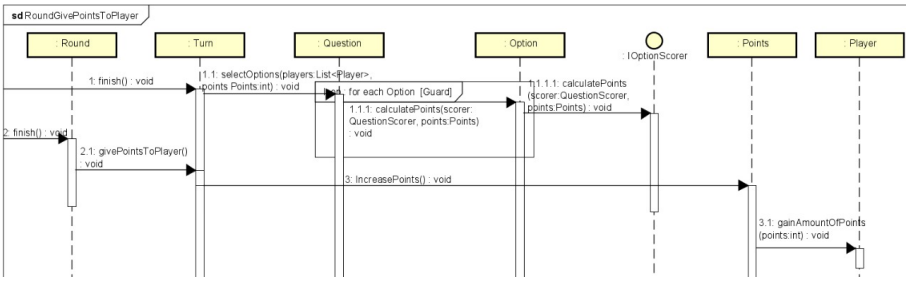

Figura 13: Diagrama de secuencia de como se transfieren los puntos desde una respuesta hasta la clase player.

Para ser mas específicos e indagar un poco mas adentro de lo que pasa al calcular los puntos, a continuación mostramos como calculan los puntos los distintos scorers, dependiendo si la pregunta fue correctamente o incorrectamente respondida.

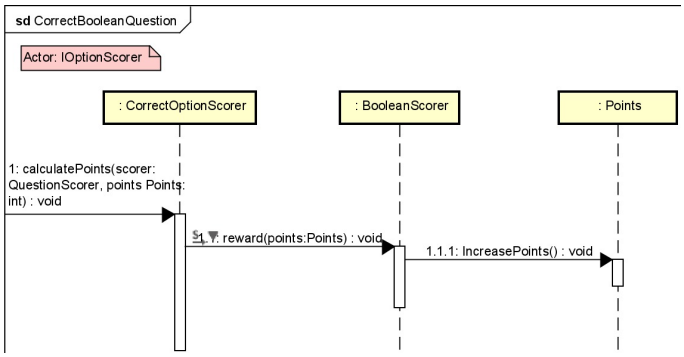

Figura 14: Diagrama de secuencia de una pregunta *Booleana* correcta.

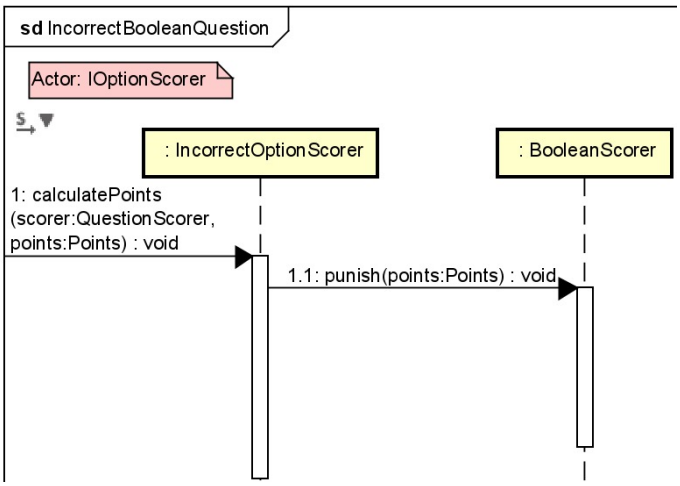

Figura 15: .Diagrama de secuencia de una pregunta *Booleana* incorrecta

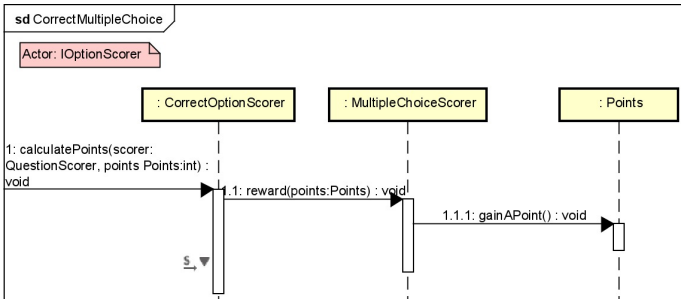

Figura 16: . Diagrama de secuencia de una pregunta *Mutiple Choice* correcta

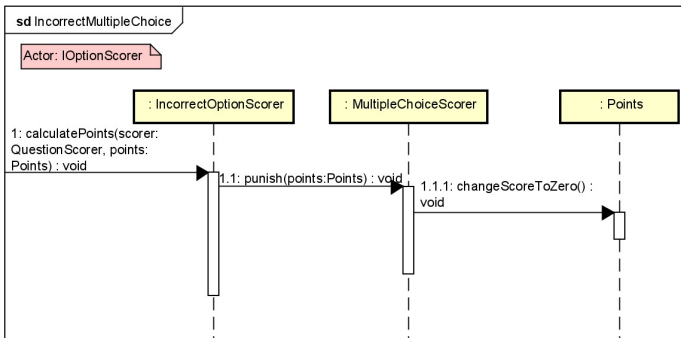

Figura 17: .Diagrama de secuencia de una pregunta *Multiple Choice* incorrecta

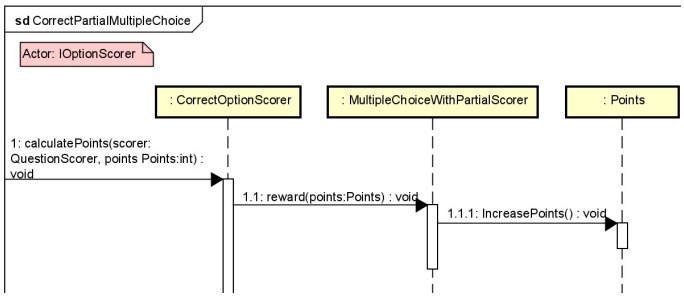

Figura 18: . Diagrama de secuencia de una pregunta *Mutiple Choice* correcta con puntos parcial

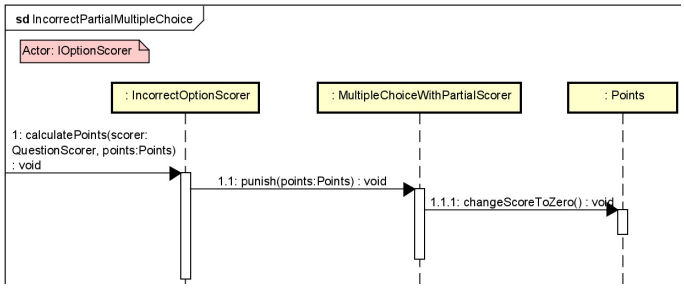

Figura 19: .Diagrama de secuencia de una pregunta *Multiple Choice* incorrecta con puntos parciales

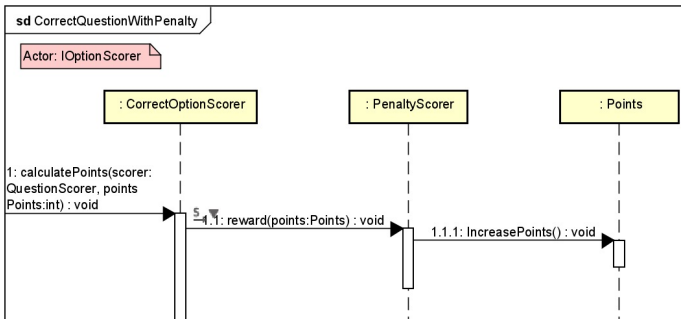

Figura 20: . Diagrama de secuencia de una pregunta *With Penalty* correcta

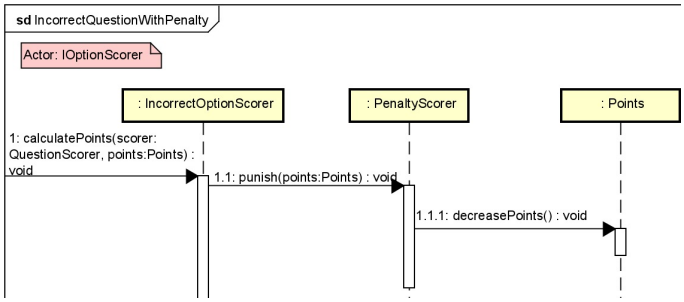

Figura 21: .Diagrama de secuencia de una pregunta *With Penalty* incorrecta

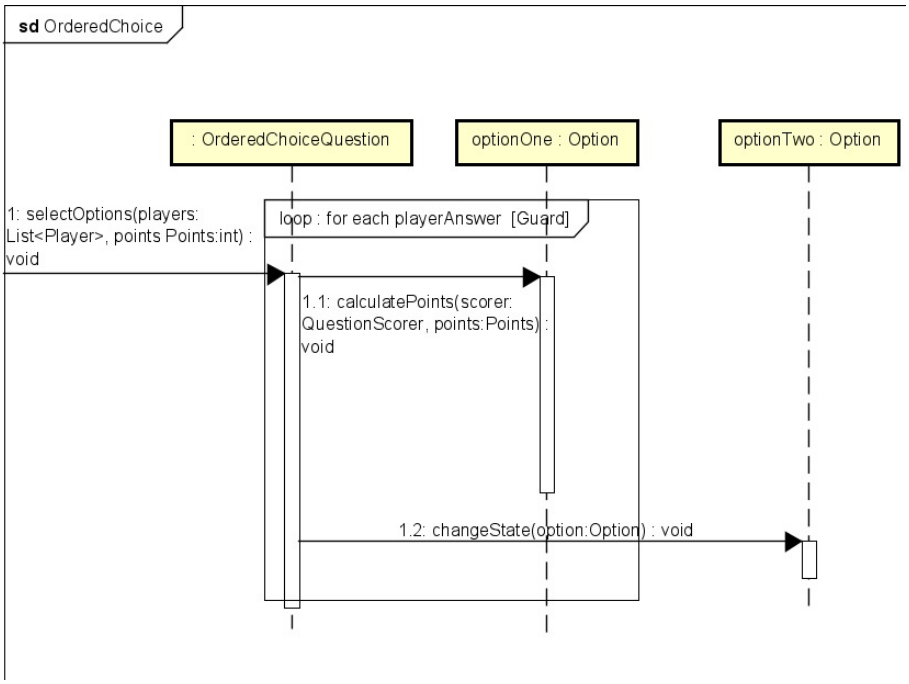

Figura 22: Diagrama de secuencia de una pregunta *Ordered Choice*.

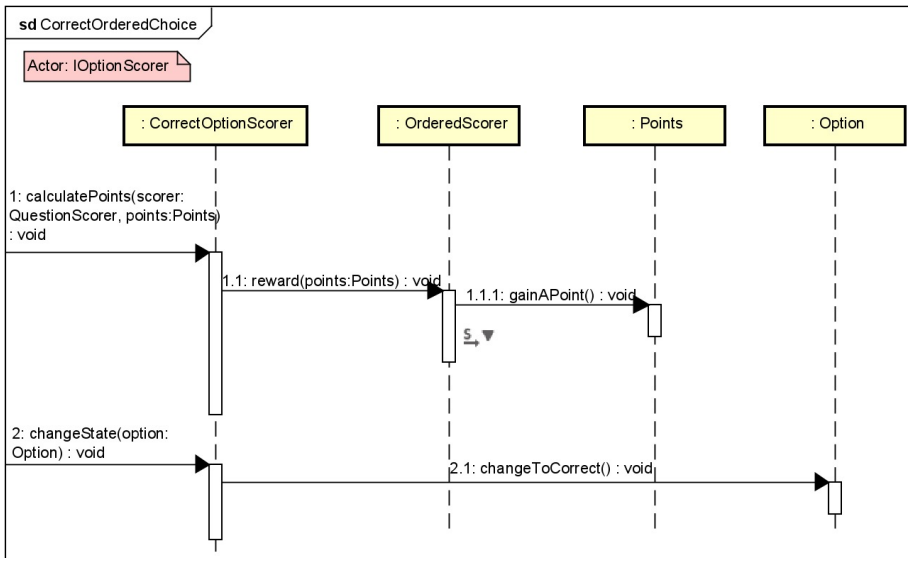

Figura 23: Diagrama de secuencia de una pregunta *Ordered Choice* correcta.

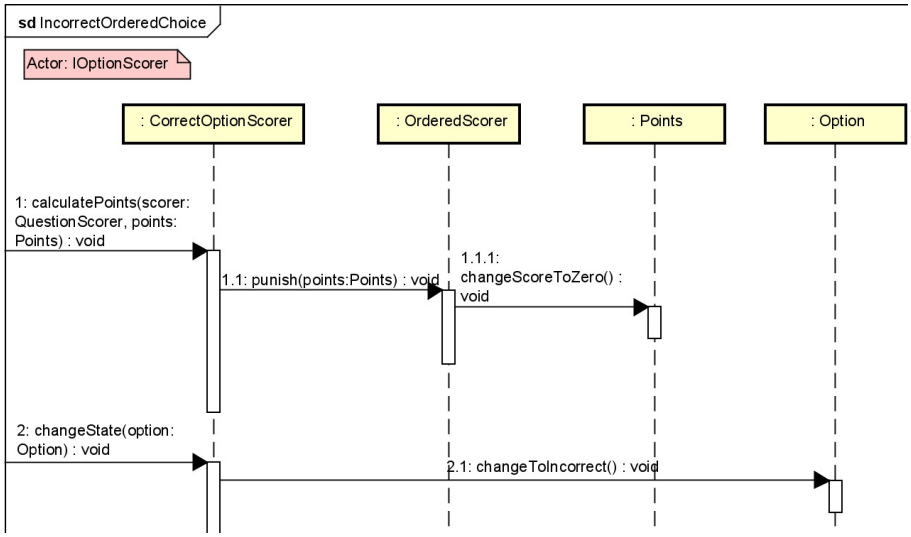

Figura 24: Diagrama de secuencia de una pregunta *Ordered Choice* incorrecta .

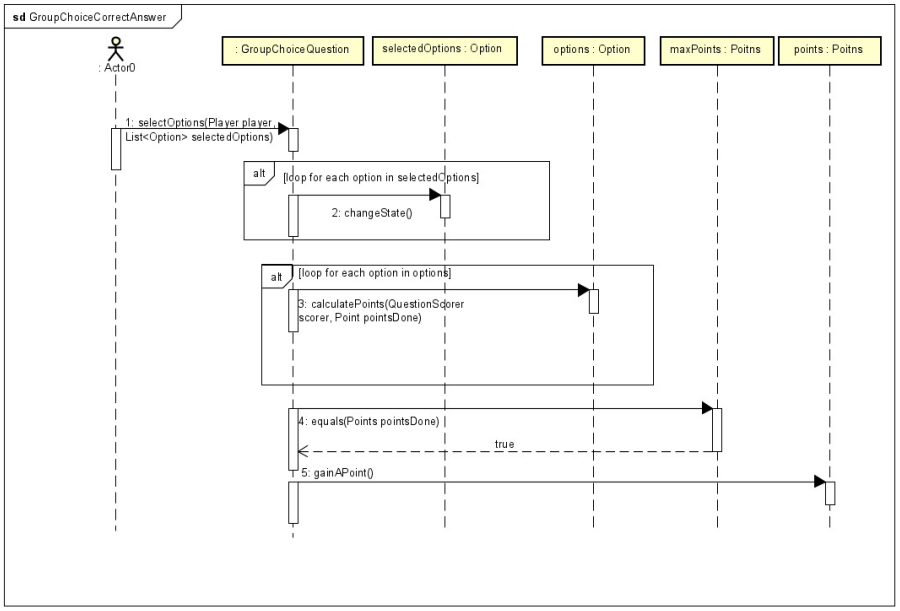

Figura 25: Diagrama de secuencia de una pregunta *Group Choice* correcta.

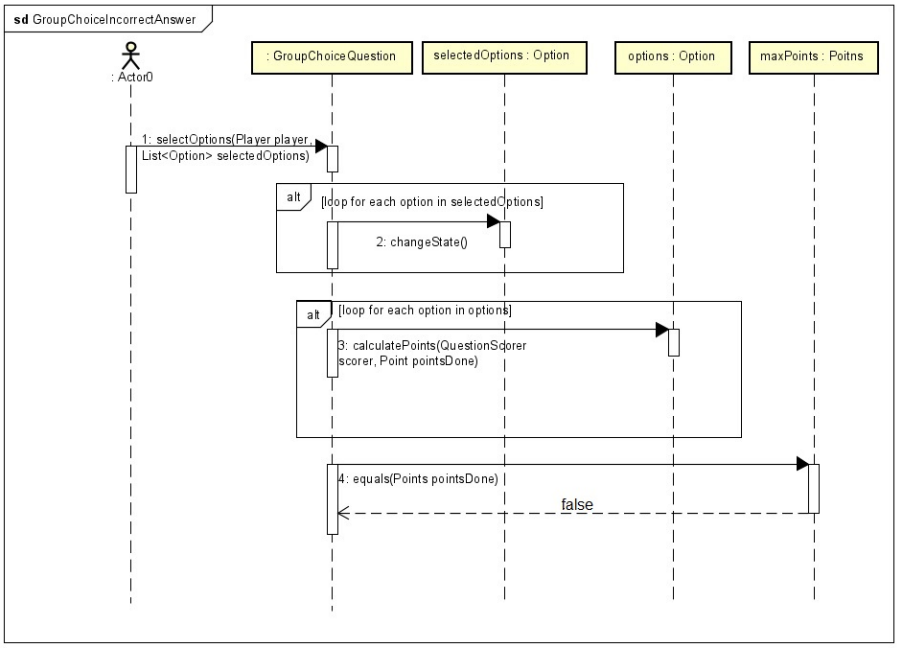

Figura 26: Diagrama de secuencia de una pregunta *Group Choice* incorrecta.

7. Diagramas de Paquetes

Se dividió el proyecto en varios paquetes ordenado por utilidades. EL paquete constantes contiene todas las constantes que se utilizan en el modelo, las excepciones y demás paquetes. Se creó con el propósito de no *hardcodear* las constantes. Dentro de el paquete de *utils* se encuentra las clases encargadas de crear objetos a partir de un formato *json* de entrada. En el paquete *exceptions* contiene las excepciones utilizadas

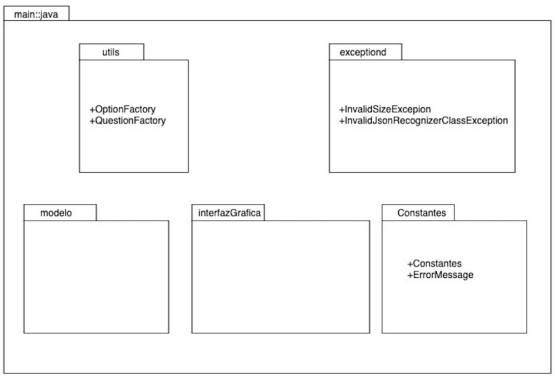

Figura 27: Diagrama de paquetes dentro de main y java.

Tanto dentro de modelo como dentro de interfazGrafica se dividió el proyecto en subpaquetes El paquete de modelo contiene todas las clases que forman parte del diseño del juego *kahoot*

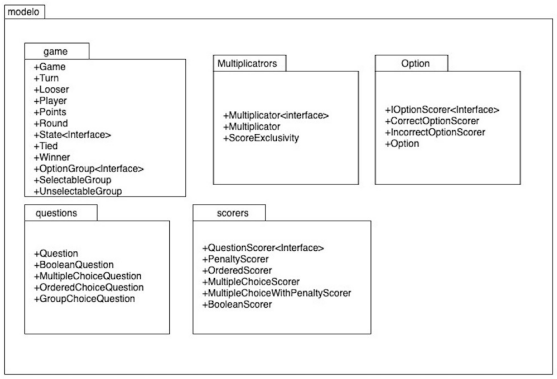

Figura 28: Diagrama de paquetes dentro modelo.

Dentro modelo también se dividió en subpaquetes como se ve el la figura 28. El subpaquete game con las clases correspondientes al manejo del juego, el subpaquete *questions* con la clase abstracta *Question* y las que heredan de la misma. El subpaquete *scorers* es similar a *questions* pero con la interfaz QuestionScorer.

El paquete interfazGrafica contiene todas las clases que forman parte del diseño de la pantalla del juego

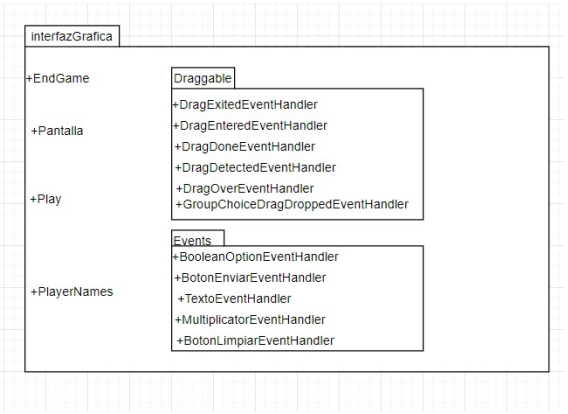

Figura 29: Diagrama de paquetes dentro interfaz Gráfica.

Dentro de interfaz gráfica se dividió entre el subpaquete de los eventos que contiene parte de los *event handlers* y el subpaquete *draggable* .

8. Diagramas de Estado

El juego comienza en el estado inicial donde se ingresan los nombres de cada jugador. Posteriormente se pasa al estado de responder la pregunta, cada jugador tiene su turno y se vuelve al mismo estado con la siguiente pregunta, de no haber mas preguntas se llega al estado final. En estado final se muestra en pantalla los puntos de los jugadores, quien ganó o si empataron.

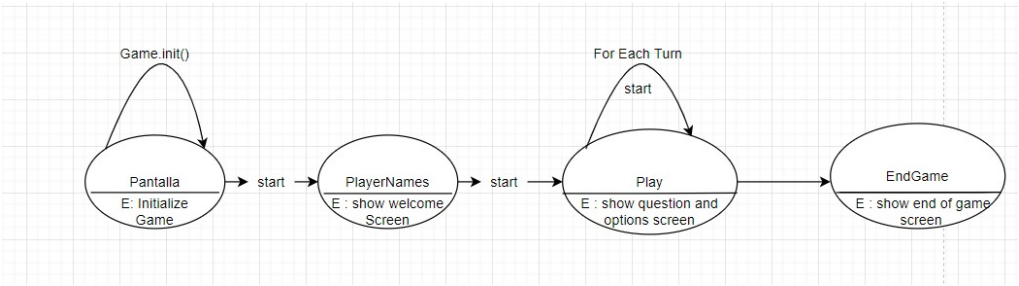

Figura 30: Diagrama de estado del juego.

9. Detalles de implementación

Para llegar a resolver este trabajo se decidió realizar a cada entrega los diagramas tanto de clase como de secuencia y los test que se iban creando y refacotrizando a medida que se agregaba implementaciones en el código (siguiendo TDD). Dentro del código decidimos usar algunos patrones de diseño como *State Pattern, Strategy Pattern y Abstract Factory Pattern.* El Abstract Factory Pattern se utilizó para el manejo del archivo JSON, el cual contiene las preguntas que se utilizan en el juego. El *State Pattern*, se utilizó para poder diferenciar polimorficamente si las opciones posibles de una pregunta son correctas o incorrectas y atribuir puntos adecuadamente. El *Strategy Pattern* fue utilizado de ayuda al crear la manera de atribuir puntos según las distintas preguntas, en nuestro caso llamado *Scorers.* Se utilizo este para desacoplar los *Scorers* de las preguntas y que estos no dependan de ellas. Para la implementación del modelo y la vista, decidimos hacer un modelo de rondas y turnos donde la clase *Game* cumpla la función orquestador delegando responsabilidades a las demás clases y donde éste tuviera una clase *Round* por cada pregunta del

juego y una clase *Turn* por cada jugador (teniendo en cuenta que el proyecto fue pensado para 2 jugadores habrá 2 turnos por cada ronda).

10. Excepciones

- InvalidSizeException: Su finalidad es garantizar que la lista de opciones que reciben las preguntas de MultipleChoice no posean menos de dos opciones o más de cinco. En el caso de GroupChoice no puede tener más de seis. Se lanza en el constructor de las respectivas preguntas y en los métodos unmarshal de la clase QuestionFactory.
- InvalidJsonRecognizerClassException: Se lanza cuando no se puede obtener correctamente el tipo tanto de una Question o un QuestionScorer dentro de la clase QuestionFactory.

11. Conclusiones

A partir de las especificaciones dadas y los supuestos considerados nos propusimos un esquema de trabajo y objetivos a realizar para dar por cumplido con el proyecto Kahoot.

Dentro de los objetivos que nos propusimos al comienzo del trabajo fue lograr un buen diseño, cumplir con las especificaciones, obtener un coverage del código superior al 90 % y cumplir con los supuestos propuestos.

Se logró superar el 90 % del coverage de todo el código y cumplir con casi todas las especificaciones excepto por la regla de tener dos consumibles cada jugador en todo el juego, por falta de tiempo no se pudo agregar al final aunque la idea para poder realizarlo era que la clase Player conozca la cantidad de consumibles que tiene y se resten cada vez que se usa.

Finalmente se pudieron aplicar los conceptos de la programación orientada a objetos aprendidos dentro de la cursada. Se afianzaron los conceptos de los patrones de diseño al tener que implementarlos y se adquirieron conocimientos en el trabajo en equipo, integración continuas y el manejo herramientas como github y trello que hoy toman un rol casi imprescindible dentro del ámbito del desarrollo de software.