Segunda entrega proyecto final:

Camilo Escobar Arteaga - A00358632
Bryan A. Guapacha F. - A00359388

1.

a. **Requerimientos funcionales:**

RF1. Inicialización del juego, Permite al usuario jugar una partida, logrando cargar los datos necesarios y crear una nueva partida con respectiva estructura del torneo haciendo referencia a las etapas del torneo (octavos, cuartos y final), Resultado esperado poder iniciar el torneo correctamente.

RF2. Seleccionar el equipo y el uniforme del jugador, teniendo en cuenta que primero se escoge el equipo y después el uniforme, Resultado esperado es que el jugador pueda seleccionar el equipo y uniforme deseado y poder así seguir con la secuencia del juego correctamente.

RF3. Generar los puntos del juego**,** Permite guardar los puntos del jugador teniendo en cuenta en donde quedó del torneo, con estos puntos el jugador tenga la oportunidad de competir para quedar en los mejores 10 jugadores del juego, finalmente se espera que se guarde correctamente en el juego el jugador con sus respectivos puntos.

RF4. Ordenamiento por nombres de los récords, este método permite comparar los jugadores ordenándolos en orden ascendente con respecto al abecedario. Resultado esperado los 10 mejores ordenados correctamente.

RF5. Ordenamiento por puntuación de los récords, este método permite comparar los jugadores ordenándolos en orden ascendente. Resultado esperado los 10 mejores ordenados correctamente.

RF6. Ordenamiento por nombres de los equipos, este método permite comparar los equipos ordenándolos en orden ascendente con respecto al abecedario. Resultado esperado es que cuando el jugador vaya a escoger los equipos estos están ordenados correctamente.

RF7. Ordenamiento por nombres de los uniformes, este método permite comparar los uniformes ordenándolos en orden ascendente con respecto al abecedario. Resultado esperado es que cuando el jugador vaya a escoger los uniformes estos están ordenados correctamente.

RF8. Generar resultados de los partidos de los demás oponentes, Permite generar aleatoriamente los resultados de los partidos de los oponentes del torneo y así determinar el oponente del jugador dependiendo si gano o perdió el partido jugado o a punto de jugar. Resultado esperado es tener correctamente generada la estructura de los resultados del torneo, donde se continuará con los equipos que pasaron la ronda.

RF9. Generar el oponente, este método permite al comienzo del torneo generar el oponente del jugador. Resultado esperado es que el jugador pueda tener un oponente y que el oponente

no se genere en otros partidos, cumpliendo así correctamente el objetivo de general el oponente.

RF10. Generar movimiento del Balón, permite generar que rebote tanto en los límites de la pantalla como en el jugador y el oponente, Resultado esperado es que el balón rebote correctamente respondiendo a la dirección de los rebotes y respetando los límites de la pantalla.

RF11. Generador de tiempo del partido, es llevar el tiempo del partido para poder darle un límite y poder que el jugador tenga en cuenta el tiempo para poder así hacer sus estrategias de juego. Resultado esperado que el cronómetro del juego funcione correctamente y que en su debido limite termine el partido para definir el ganador y el perdedor.

RF12. Permite llevar el marcador del partido, es llevar la cuenta de los goles marcados por los dos oponentes y así llevando el marcador donde uno pueda ver quien va ganando o perdiendo, Resultado esperado es que el marcador lleve correctamente la cuenta de los goles de cada equipo.

RF13. Generar movimiento del jugador, Permite que el jugador tenga un movimiento vertical en la pantalla permitiendo que el balón no entre en su arco y poder hacerla rebotar para que efectúe un golazo, Resultado esperado es que el usuario pulse las flechas haciendo que el jugador pueda tener movimiento en la cancha, teniendo en cuenta que tenga coherencia con la flecha que hundió el usuario y que cuando el jugador toque el balón que este rebote correctamente hacia el otro lado de la cancha.

RF14. Generar movimiento del jugador oponente, el oponente que es la "computadora" (el equipo a enfrentar) tenga un movimiento vertical logrando detener ocasionalmente las pelotas para que no le hagan gol haciendo real una simulación de un contrincante bueno para el jugador, Resultado esperado es que el oponente tenga movimientos verticales y que cuando toque el balón este rebote hacia el otro lado de la cancha.

RF15. Generador de lista de puntajes, donde se sacan los 10 mejores jugadores hasta el momento en el juego, Resultado esperado es que la lista tenga coherencia con los puntajes de forma ascendente y que si sean los mejores 10, hay que tener en cuenta que si no se han jugado los 10 jugadores el método solo imprimirá los que estén hasta el momento, finalmente imprimiéndolos con un formato determinado que es primero el número de la posición en la lista, después el nombre y por último el puntaje del jugador.

RF16. Salir del juego, este método permite salir del juego, Resultado esperado que salga del juego correctamente.

RF17. Persistencia de los datos, este método permite guardar el puntaje de los jugadores mediante serialización, Resultado esperado es que los datos de antes de cerrar la aplicación se conservan a la hora de volver a abrir la aplicación.

RF18. Generar siguiente partido, Determinar si el jugador puede seguir jugando o no sigue en el torneo, entonces solo si gana genera el siguiente partido, Resultado esperado que el jugador gane el partido y pueda continuar tu torneo correctamente.

RF19. Impresión de árbol mostrando la etapa del torneo, este método permite darle al usuario una visualización del torneo donde imprime el árbol de la secuencia del torneo y sus etapas, Resultado esperado es que el árbol pueda imprimirse correctamente y pueda tener una buena coherencia con el torneo y los equipos que ganan y pasan a la siguiente etapa.

RF20. Cargar los archivos de texto, este método permite cargar los nombres de los equipos a el programa, Resultado esperado es que cuando el jugador vaya a escoger un equipo si hayan cargado correctamente.

b. **Requerimientos no funcionales:**

Req-1. Usabilidad de la aplicación. La aplicación deber ser intuitiva al uso y las tareas deben estar correctamente explicadas, siendo fácil de usa a simple vista.

Req-2. La plataforma no puede ser accedida directamente, sino a través de una interfaz diseñada para estos propósitos.

Req-3. Las excepciones deben ser reportadas por la propia aplicación en la medida de las posibilidades y no por el sistema operativo. Los mensajes del sistema deben estar en el idioma apropiado.

Req-4. El sistema deberá responder en el mínimo tiempo posible ante las solicitudes de los usuarios y el procesamiento de la información. La eficiencia de la aplicación estará determinada en gran medida por el aprovechamiento de los recursos que se disponen en el modelo.

Req-5. El sistema debe ser tolerante a los fallos inesperados por parte del sistema y los posibles errores de los diferentes usuarios.

Req-6. La aplicación debe poderse ejecutar en diferentes entornos, como Windows, Linux, MacOS, etc.

Req-7. La aplicación debe estar en la capacidad de almacenar los datos suministrados por el usuario, aun cuando la aplicación se haya cerrado y vuelva a ser iniciada tiempo después.

Req-8. Inicializar la aplicación no debería de tardar más de 1 minuto.

Req-9. Todas las operaciones búsqueda, selección, entre otras, deben realizadas por el software.

c. **d. f. Diagrama de clases:**
   por cuestiones de visibilidad decisimos colocarlo en un documento aparte.

e. **Diseño de pruebas unitarias.**

Objetivo: La razón para realizar esta prueba, es verificar que el método "mover" de la clase Ball, permita el recorrido del balón dentro de la cancha sin cruzar los parámetros de la ventana o bordes, etc.

Nombre	**Clase**	**Escenario**
Setup1	Ball	Un objeto de la clase Ball con vecX = 800 y vecY = 400
Setup2	Ball	Un objeto de la case Ball con vecX = -200 y vecY = -80

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Ball	Mover	Setup1	X = 800 y Y= 400	X = -800 y Y = -400
Ball	Mover	Setup2	X = -200 y Y = -80	X = 200 y Y = 80

Objetivo: La razón para realizar esta prueba, es verificar que el método "time" de la case Clock, esté haciendo el incremento en segundo de uno en uno. Ejemplo: 1seg, 2seg, 3seg, etc.

Nombre	**Clase**	**Escenario**
Setup1	ClockTest	Un objeto de la clase Clock con seconds = 0
Setup2	ClockTest	Un objeto de la clase Clock con seconds = 5
Setup3	ClockTest	Un objeto de la clase Clock con seconds = -8

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Clock	time	Setup1	0	1
Clock	time	Setup2	5	6
Clock	time	Setup3	-8	-7

Objetivo: La razón para realizar esta prueba, es verificar que el método "addTeam" de la clase Team se encuentre agregando correctamente los equipos al arrayList de equipos o teams.

Nombre	**Clase**	**Escenario**
Setup1	TeamTest	Un objeto de la clase Team con name = Colombia

Setup2	TeamTest	Un objeto de la clase Team con name = Argentina

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Team	addTeam	Setup1	Nuevo Objeto de tipo Team	TeamColombia se ha agregado correctamente
Team	addTeam	Setup2	Nuevo objeto de tipo Team	TeamArgenita se agrega al final, después del ultimo team agregado.

Objetivo: La razón de esta prueba, es verificar que el método "searchTeam" de la clase Team, se encuentre realizando de manera efectiva la búsqueda del equipo deseado que ingresan los usuarios al jugar.

Nombre	**Clase**	**Escenario**
Setup1	TeamTest	Un objeto de la clase Team con name = Brasil
Setup2	TeamTest	Un objeto de la clase Team con name = Peru

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Team	searchTeam	Setup1	El Objeto de tipo team. (El array esta vacio).	"El equipo buscado no existe"
Team	searchTeam	Setup2	El objeto de tipo team. (El array tiene objetos del mismo tipo y el parámetro buscado se encuentra ahi)	Se retorna el objeto buscado.

Objetivo: La razón de esta prueba, es verificar que el método "addUniform" de la clase Team, se encuentre agregando exitosamente los uniformes de los equipos al arrayList de Uniformes.

Nombre	**Clase**	**Escenario**
Setup1	TeamTest	firstUniform = null
Setup2	TeamTest	firstUniform ≠ null

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**

Team	addUniform	Setup1	colombiaUniform.png	El uniforme se agrega correctamente
Team	addUniform	Setup2	chileUniform.png	El uniforme se agrega correctamente

---

Objetivo: La razón de esta prueba, es verificar que el método "searchUniform" de la clase Team, funcione correctamente a la hora de realizar la selección de si se desea jugar con uniforme local o visitante, se este seleccionando el uniforme correcto.

Nombre	**Clase**	**Escenario**
Setup1	TeamTest	Un objeto de la clase Team con img = caliUniform.ong
Setup2	TeaamTest	Un objeto de la clase Team con img = colombiaUniform.png

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Team	searchUniform	Setup1	"caliUniform.png"	Este uniforme no existe en el sistema
Team	searchUniform	Setup2	"mexicoUniform.png"	Este uniforme fue encontrado

---

Objetivo: La razón de esta prueba, es verificar que el método "resultadoPartidos" de la clase Position, realice exitosamente la asignación del equipo que debe ganar el partido entre cada partido.

Nombre	**Clase**	**Escenario**
Setup1	PositionTest	Un objeto de tipo Team donde el ganador del partido es team1
Setup2	PositionTest	Un objetito de tipo Team donde el ganador del partido es team2

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Position	resultadoPartidos	Setup1	numGanador = 1	El ganador es team1
Position	resultadoPartidos	Setup2	numGanador ≠ 1	El ganador es team2

Objetivo: La razón de esta prueba, es verificar que el método "addPosition" de la clase Position, se encuentre asignando el lugar adecuado a los equipos dentro del árbol binario.

Nombre	Clase	Escenario
Setup1	PositionTest	Las posiciones del árbol están vacías
Setup2	PositionTest	Por lo menos la primera posición ya está definida

Clase	Método	Escenario	Valores de entrada	Resultado
Position	addPosition	Setup1	left = null y right = null	Se añade correctamente team1 y team2
Position	addPosition	Setup2	Left ≠ null y right ≠ null	Añade dependiendo del lado left o right

---

Objetivo: La razon de esta prueba, es verificar que el metodo "ganadorPartido" de la clase Match, realice efectivamente la asignacion de goles que marca cada equipo durante el partido que se esta jugando .

Nombre	Clase	Escenario
Setup1	MatchTest	golesTeam1 > golesTeam2
Setup2	MatchTest	golesTeam2 < golesTeam1

Clase	Método	Escenario	Valores de entrada	Resultado
Match	ganadorPartido	Setup1	3 > 1	El ganador del partido es el team1
Match	ganadorPartido	Setup2	1 < 2	El ganador del partido es el team2

Objetivo: La razón de esta prueba, es verificar que el método "moverBall" de la clase Match, tenga claras las áreas donde el programa reconoce que ha anotado un gol dentro del partido.

Nombre	Clase	Escenario
Setup1	MatchTest	Un objeto de la clase Match donde la posición en x de ball es = 530
Setup2	MatchTest	Un objeto de la clase Match donde la posición en x de ball es = -80

Clase	Método	Escenario	Valores de entrada	Resultado
Match	moverBall	Setup1	X = 530	El equipo 1 ha anotado un gol

Match	moverBall	Setup2	X = -80	El equipo 2 ha anotado un gol

Objetivo: La razón de esta prueba, es verificar que el método "organizarEquipos" de la clase Game, ordene de manera idónea los equipos que entran por un criterio de orden alfabetico.

Nombre	**Clase**	**Escenario**
Setup1	GameTest	Equipo1 = "Colombia", Equipo2 = "Peru", Equipo3 = "Argentina"
Setup2	GameTest	Equipo1 = "Usa", Equipo2 = "Mexico", Equipo3 = "España"

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Game	organizarEquipos	Setup1	Colombia, Peru, Argentina	1-Argentina, 2-Colombia, 3-Peru.
Game	organizarEquipos	Setup2	Usa, Mexico, España	1-España, 2-Mexico, 3-Usa

---

Objetivo: La razón de esta prueba, es verificar que el método "loadTeam" de la clase Game, se encuentre cargando de manera efectiva los archivos que me permiten conocer los equipo que están disponibles para elegir antes de jugar.

Nombre	**Clase**	**Escenario**
Setup1	GameTest	El archivo de texto no existe
Setup2	GameTest	El archivo de texto no tiene el mismo nombre suministrado
Setup3	GameTest	El archivo de texto existe y concuerda su nombre

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Game	loadTeam	Setup1	null	No se encuentra el archivo a cargar
Game	loadTeam	Setup2	"teams#%.txt"	No se reconoce el archivo que debe ser leido
Game	loadTeam	Setup3	"Teams.txt"	Los datos se han cargado exitosamente

Objetivo: La razón de esta prueba, es verificar que el método “serializableScore” de la clase Game, se encuentre guardando de manera efectiva los datos de los usuarios al terminar cada copa jugada. (nombre y puntos obtenidos durante el campeonato)

Nombre	**Clase**	**Escenario**
Setup1	GameTest	La dirección suministrada se creará en los archivos de la aplicación
Setup2	GameTest	La dirección suministrada no existe

Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
Game	serializableScore	Setup1	“.\\data\\Scores.txt”	El archivo se creó correctamente
Game	serializableScore	Setup2	“.\\address\\Scores.txt”	El folder suministrado en la dirección, no existe.