El tren del circo

Autores:

Bayón Benegas, Marc

Cuenca Gómez, Emilio

Daza Pastrana, Francisco José

Espinaco Villalba, Francisco Javier

Tristancho Reyes, Álvaro

Viñas Sandiez, Antonio Jesús

Convocatoria:

Febrero 2011

Índice:

1.URL del video

http://www.youtube.com/watch?v=Nyde3nEXl7s

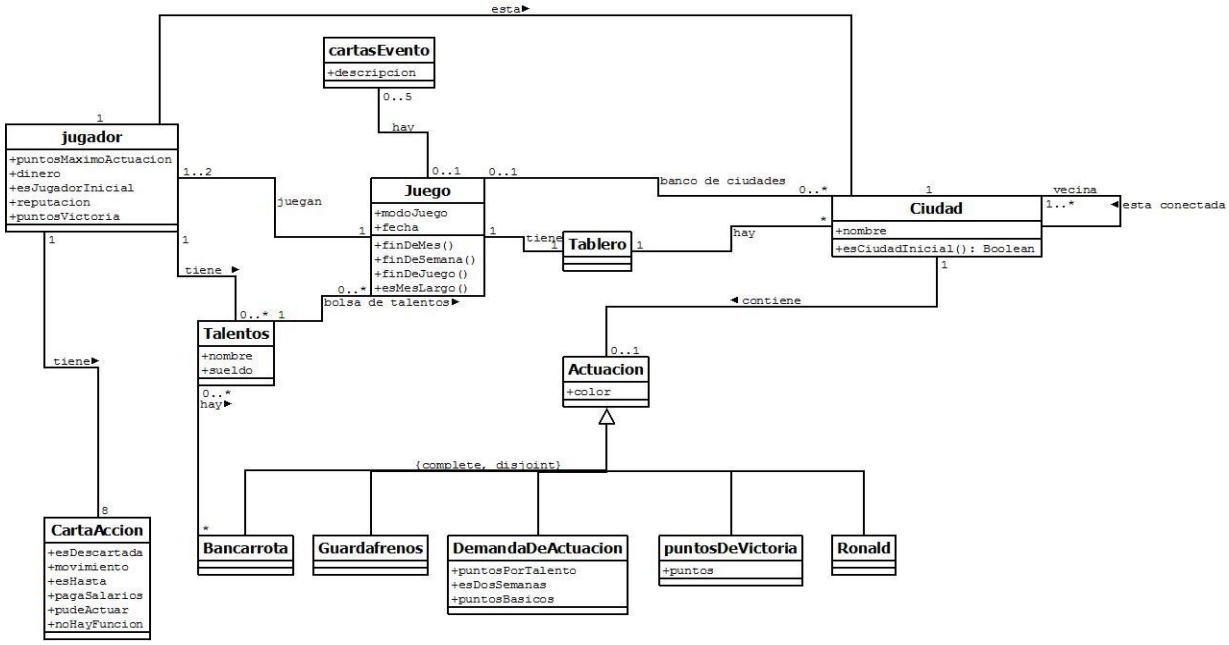
2. Planificación temporal

- Día 4 de octubre de 2010 a las 8:35, coordinación
- Día 5 de octubre de 2010 a las 8:35, coordinación
- Día 7 de octubre de 2010 a las 9:30, entrega iteración 1
- Día 7 de octubre de 2010 a las 11:45, coordinación
- Día 13 de octubre de 2010 a las 8:30, coordinación
- Día 18 de octubre de 2010 a las 9:00, entrega iteración 2
- Día 11 de noviembre de 2010 a las 21:00, entrega iteración 3
- Día 9 de diciembre de 2010 a las 23:00, entrega iteración 4
- Día 23 de diciembre de 2010 a las 11:40, entrega iteración 5
- Día 14 de enero de 2011 a las 14:10, entrega iteración 6
- Día 19 de enero de 2011 a las :,entrega final

3. Seguimiento

Miembros\Puntos por iteración	*1*	*2*	*3*	*4*	*5*	*6*	*7*	***TOTAL***
Bayón Benegas, Marc	5	5	5	5	5	5	5	35
Cuenca Gómez, Emilio	5	5	5	5	5	5	5	35
Daza Pastrana, Francisco José	5	5	5	5	5	5	5	35
Espinaco Villalba, Francisco Javier	5	5	5	5	5	5	5	35
Tristancho Reyes, Álvaro	5	5	5	5	5	5	5	35
Viñas Sandiez, Antonio Jesús	5	5	5	5	5	5	5	35
TOTAL	30	30	30	30	30	30	30	210

4. Modelo de análisis



5.Iteraciones

5.1 Iteración 3

Añadidos en iteración 3

- Implementación parcial de la clase correspondiente al juego.
- Implementación parcial de la clase correspondiente al jugador.
- Implementación de las clases referentes al tablero.
- Documentos de asignación de responsabilidades (a continuación).
- Diagramas UML de diseño (a continuación).

Asuntos pendientes en iteración 4

- Implementación de las cartas de acción.
- Implementación de las bolsas.
- Implementación de las actuaciones.
- Implementación de talentos.

Método startGame:

Identificador	Descripción de la acción de alto nivel
T-005	Método para inicializar el juego

Pasos (usar pseudocódigo o similar)

1. Contador de semanas puesto a 0
2. Creación de estructura para añadir talentos
3. Creación del payaso inicial e inserción en la lista
4. Petición de número de jugadores
5. Petición de modo de juego
6. Petición de nombres de jugadores

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	Collections Factory	[List<T>] createListFactory().createList()		NO
3	GameFactory	[Talent] createTalent(String:name)		NO
4	GameFactory	[String] takeParametersToStringRestricted(String:message,String:condition)		NO
5	GameFactory	[String] takeParametersToStringRestricted(String:message,String:condition)		NO
6	GameFactory	[String] takeParametersToString(String:message)		NO

Método de alto nivel

[void] startGame()

Diagrama de Colaboración (Opcional)

Método refreshMonth:

Identificador	Descripción de la acción de alto nivel
T-006	Método que cambia el mes según la semana en la que se esté

Pasos (usar pseudocódigo o similar)

1. Si la semana está entre 0 y 3 el mes es abril
2. Si la semana está entre 4 y 8 el mes es mayo
3. Si la semana está entre 9 y 12 el mes es junio
4. Si la semana está entre 13 y 17 el mes es julio
5. Si la semana está entre 18 y 21 el mes es agosto
6. Si la semana está entre 22 y 26 el mes es septiembre
7. Cambio de primer jugador

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
7	CircusTrainGame	[void] rotatePlayers()		NO

Método de alto nivel

[void] refreshMonth()

Diagrama de Colaboración (Opcional)

Método rotatePlayers:

Identificador	Descripción de la acción de alto nivel
T-015	Invierte el orden de la lista de jugadores

Pasos (usar pseudocódigo o similar)

1. Creación de lista auxiliar para almacenar ahí los jugadores en el orden inverso
2. Se recorre la lista de jugadores sin contar el primero y se van añadiendo a la lista auxiliar definida en el paso 1.
3. Se le añade a la auxiliar de jugadores el jugador que ocupa la primera posición en la lista del juego.
4. Se vacía la lista de jugadores y se vuelca en ella la auxiliar

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel

[void] rotatePlayers()

Diagrama de Colaboración (Opcional)

Método monthChangeComprobator:

Identificador	Descripción de la acción de alto nivel
T-016	Comprueba si se ha cambiado de mes y actúa en consecuencia

Pasos (usar pseudocódigo o similar)

1. Guardar en una variable el mes antes de actualizar el mes
2. Actualización del mes
3. Guardar en una variable el mes tras actualizar
4. Si son distintos invoca al método que cambia el estado de los elementos en cuestión
5. Si son distintos invoca al método que intercambia el orden de los jugadores

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	CircusTrain Game	[void] refreshMonth( )		NO
4	CircusTrain Game	[void] finalMonth( )		NO
5	CircusTrain Game	[void] rotatePlayers( )		NO

Método de alto nivel

[void] monthChangeComprobator( )

Diagrama de Colaboración (Opcional)

Creación objetos PerformanceBag:

Identificador	Descripción de la acción de alto nivel
CT0001	Creación de objetos PerformanceBag

Pasos (usar pseudocódigo o similar)

1. Será inicializado a petición de startGame() al inicio del juego.
2. Creara las 3 bolsas de juego.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	PerformanceBag	Map<TypeTalentCard, Integer> createGreenBag()	001	
2	PerformanceBag	Map<TypeTalentCard, Integer> createYellowBag()		
3	PerformanceBag	Map<TypeTalentCard, Integer> createYellowBag()		

Método de alto nivel

void startGame()

Diagrama de Colaboración (Opcional)

Método toHire:

Identificador	Descripción de la acción de alto nivel
J-001	Método para contratar los talentos de un circo en bancarrota.

Pasos (usar pseudocódigo o similar)

1. Crear una lisa vacía de talentos donde se guardaran los talentos contratados.
2. Preguntar al usuario por cada talento que haya en el circo en bancarrota si quiere contratarlo.
3. Si decide el usuario contratar el talento, se lanza un dado y se comprueba si ha conseguido contratarlo mirando la tabla de reputación del jugador.
4. Se informa al usuario con el resultado obtenido.
5. Añadir la nueva lista a los talentos del jugador.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CollectionFactory	[List<T>] createList()	001	SI
2	BankruptCircus	[List<Talent>] getTalentCircus()		NO
3	GameFactory	[String] takeParametersToStringRestricted (String question, String condition)		
4	GameFactory	[Integer] throwDice()		
5	Player	[Integer] getHigherDiceScore()		

Método de alto nivel

[void] execute()

Diagrama de Colaboración (Opcional)

Diagrama UML de diseño

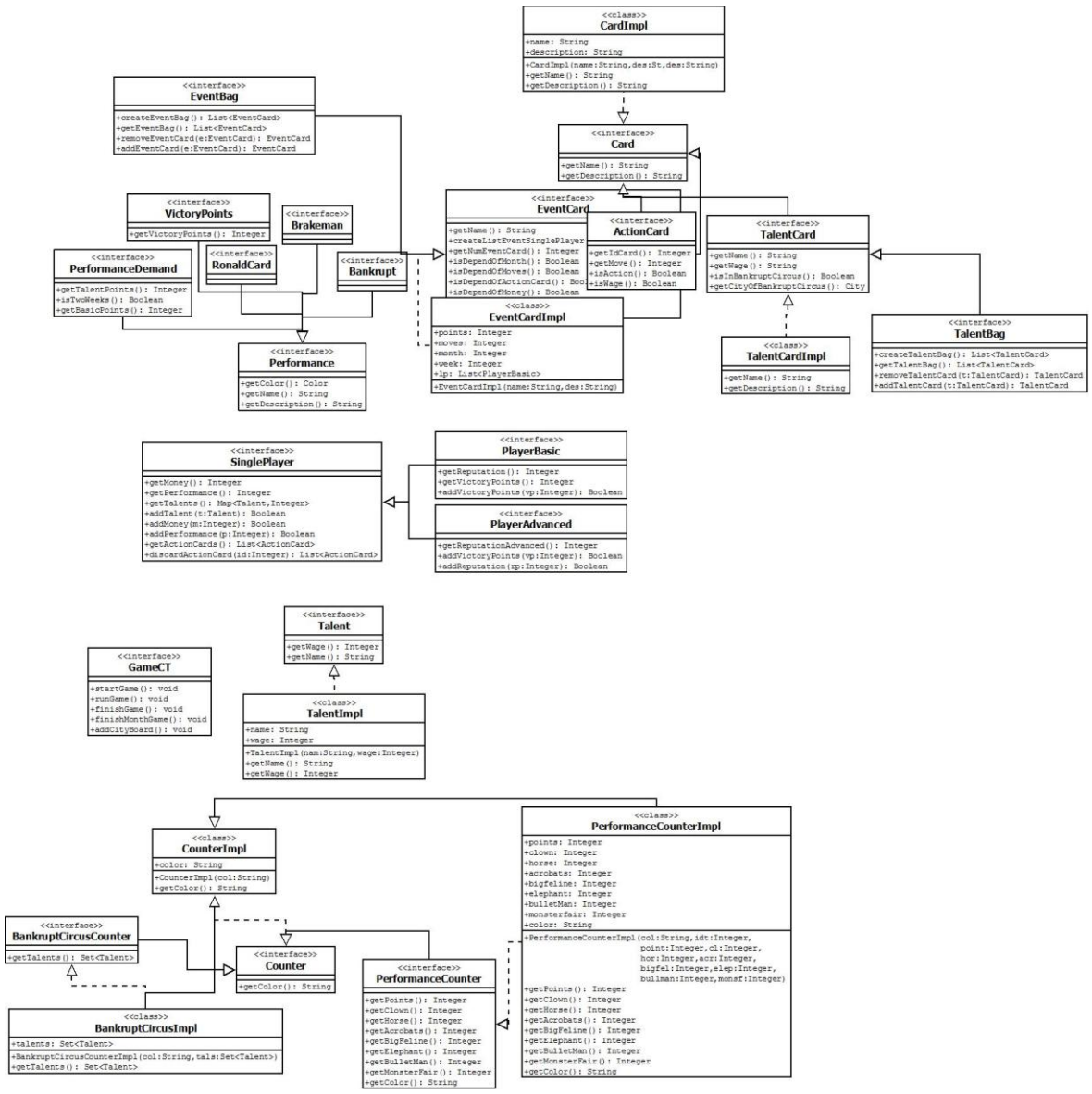


Diagrama de Bags

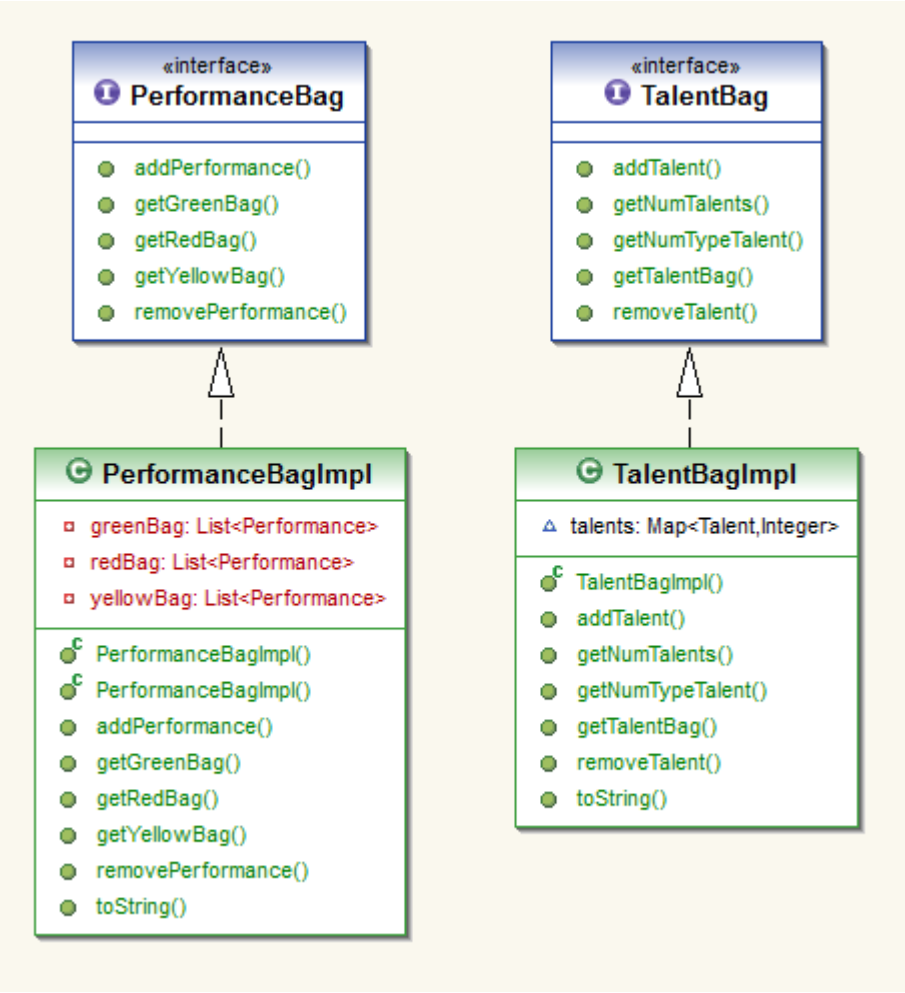


Diagrama de Board

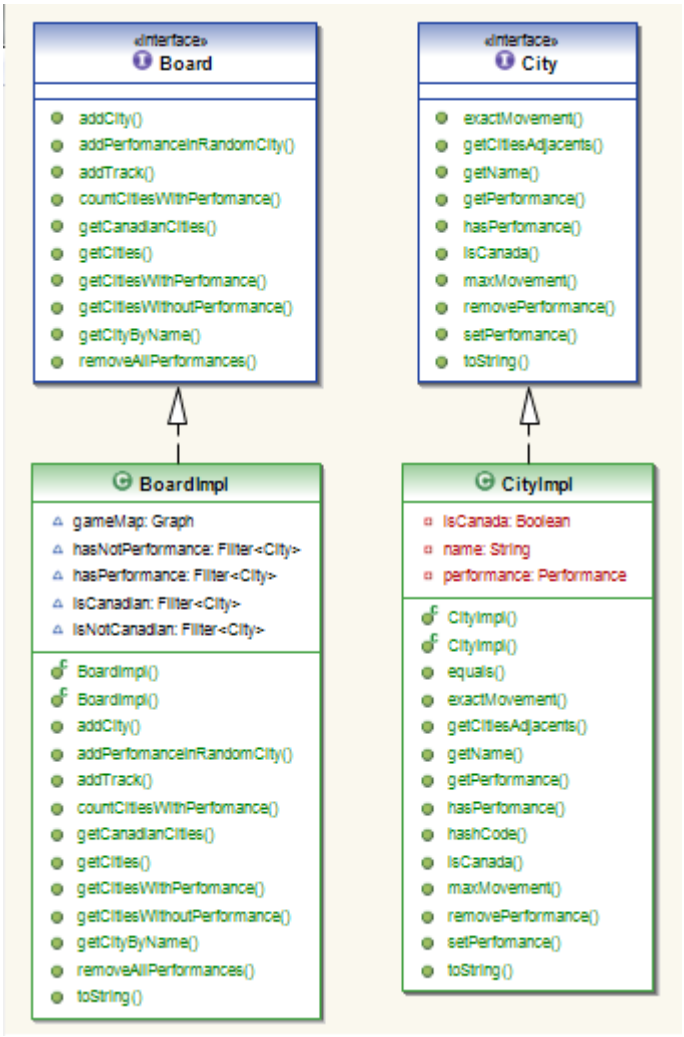


Diagrama de Card

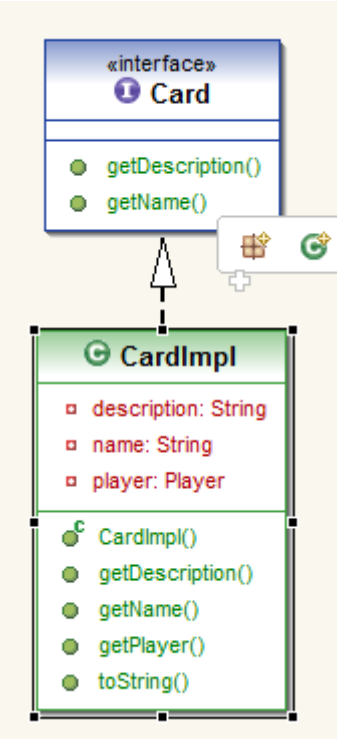
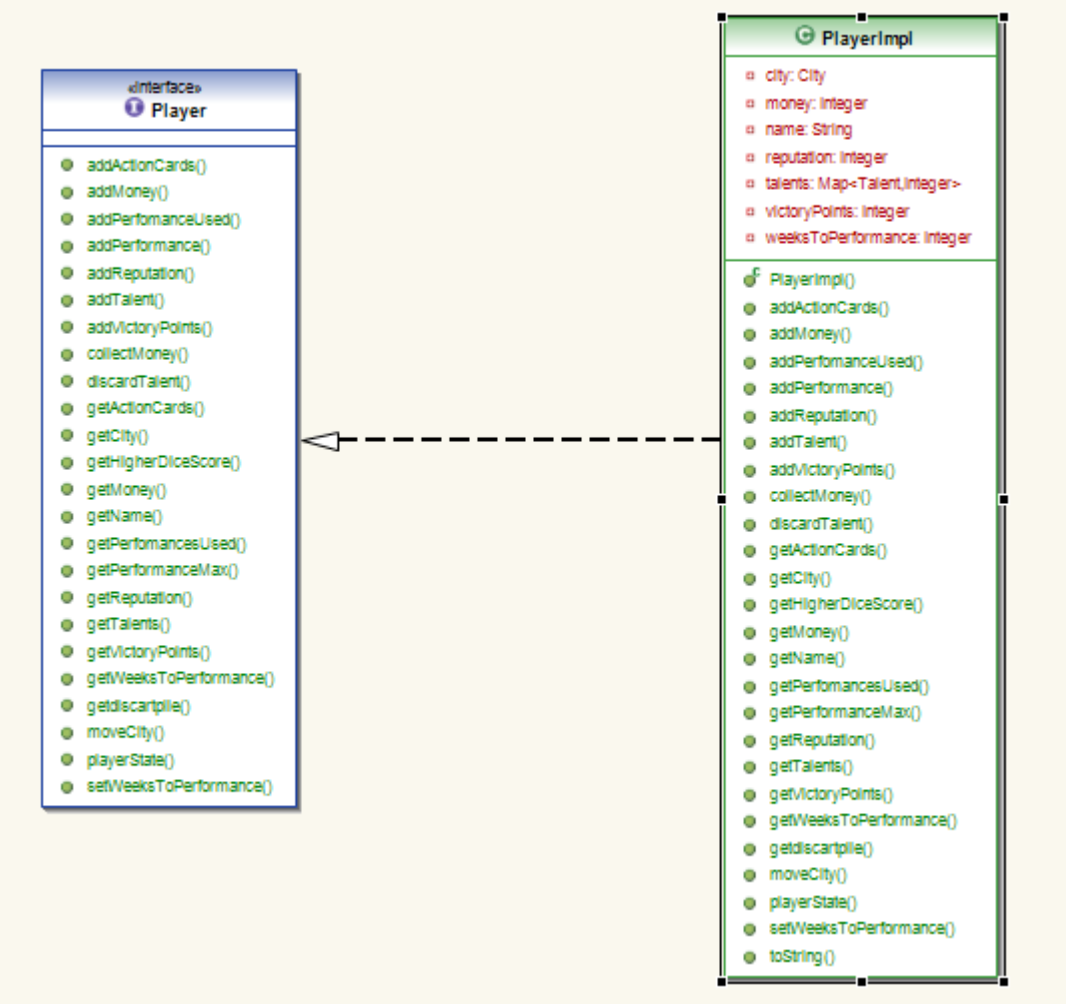


Diagrama de Player



5.2 Iteración 4

Añadidos en iteración 4

- Implementación de cartas de acción
- Implementación de las distintas bolsas (PerformanceBag, TalentBag).
- Implementación de los diferentes comandos necesarios.
- Implementación de la clase que contiene la secuencia de juego (CircusTrainGame)
- Implementación de los distintos tipos de actuaciones (BankruptCircus, PerformanceDemand,VictoryPoints).
- Implementación de la clase correspondiente al jugador (PlayerImpl).
- Implementación de los distintos talentos.
- Documentos de asignación de responsabilidades de los métodos (a continuación).
- Memorandos técnicos (a continuación).
- Diagramas UML

Asuntos pendientes en iteración 5

- Pruebas unitarias.
- Refactorización.
- Implementación de los modos de juego básicos.
- Documentos de asignación de responsabilidades.
- Integración de los diferentes componentes.

Clases Board y City:

Identificador	Descripción de la acción de alto nivel
A002	Inicializa las ciudades
Pasos (usar pseudocódigo o similar)	
1.Lee el archivo 2.Inicializa los parametros	
Diagrama de estados (Opcional)	
Métodos de bajo nivel necesarios	

Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel
BoardImpl();
Diagrama de Colaboración (Opcional)

Identificador	Descripción de la acción de alto nivel
A003	Devuelve un conjunto de ciudades a las que se puede mover como máximo.
Pasos (usar pseudocódigo o similar)	
1. Si el número de movimientos es mayor que 0. 2. Obtén las ciudades conectadas. 7. Llama a este mismo método con cada una de las ciudades, decrementando el número de movimientos. 8. Une todos los conjuntos.	
Diagrama de estados (Opcional)	
Métodos de bajo nivel necesarios	

Paso	Clase	Método	Mem. Técn.	IU
1	Vertex	getAdjacentes()		

Método de alto nivel
set<city> getMaxMovement(Integer jump)
Diagrama de Colaboración (Opcional)

Identificador	Descripción de la acción de alto nivel
A004	Lista de ciudades a con distancia maxima

Pasos (usar pseudocódigo o similar)
Ejecuta el metodo excactMovement con cada uno de los saltos

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	City	exactMovement	A012	NO

Método de alto nivel
List<City> maxMovement()

Diagrama de Colaboración (Opcional)

Identificador	Descripción de la acción de alto nivel
A005	Devolver una lista de ciudades que están a un número exacto de saltos

Pasos (usar pseudocódigo o similar)
Recursivamente, si el número de saltos es uno, devuelve los adyacentes, si no, se ejecuta a si mismo, a los adyacentes, reduciendo el número de saltos una unidad

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	Vertex	getAdjacents()		NO
2				NO
…				

Método de alto nivel
List<City> exactMovement()

Diagrama de Colaboración (Opcional)

Clase Performance y PerformanceBag:

Identificador	Descripción de la acción de alto nivel
E001	Creación de objetos Performance

Pasos (usar pseudocódigo o similar)
1. El constructor inicializará el color , la descripción y el nombre

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel

Diagrama de Colaboración (Opcional)

Identificador	Descripción de la acción de alto nivel
E002	Creación de objetos PerformanceBag

Pasos (usar pseudocódigo o similar)

3. Será inicializado a petición de startGame() al inicio del juego.
4. Creará las 3 bolsas de juego.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	PerformanceBag	Map<TypeTalentCard, Integer> createGreenBag()	001	
2	PerformanceBag	Map<TypeTalentCard, Integer> createYellowBag()		
3	PerformanceBag	Map<TypeTalentCard, Integer> createYellowBag()		

Método de alto nivel

void startGame()

Diagrama de Colaboración (Opcional)

Método movePlayer:

Identificador	Descripción de la acción de alto nivel
AC-001	Dado un movimiento máximo, el método se encarga mostrar, las ciudades adyacentes y de mover al jugador a la ciudad determinada.

Pasos (usar pseudocódigo o similar)
1. Recorrer y mostrar la lista de ciudades adyacentes. 2. Mover a "player" a la ciudad determinada.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	City	[List<City>] maxMovement(jump:Integer)	1	NO
2	Player	[void] moveCity (city: City)		

Método de alto nivel
[void] movePlayer (Integer move);

Diagrama de Colaboración (Opcional)

Método performPlayer:

Identificador	Descripción de la acción de alto nivel
AC-002	Permite a un jugador actuar o contratar

Pasos (usar pseudocódigo o similar)
1. Obtener la "performance" de la ciudad donde está el jugador. 2. Ejecutar dicho "performance".

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	City	[Performance] getPerformance()		NO
2	Performance	[void] execute (player: Player,gameState:GameState)		NO

Método de alto nivel
[void] performPlayer (gameState:GameState);

Diagrama de Colaboración (Opcional)

Constructor PlayerImpl:

Identificador	Descripción de la acción de alto nivel
AC-003	Constructor del objeto PlayerImpl(String nombre)

Pasos (usar pseudocódigo o similar)

1. El atributo name se inicializa con el parametro nombre.
2. El atributo money se inicializa a cero.
3. El atributo perfomance_max se inicializa a cero.
4. El atributo victory_points se inicializa a cero.
5. El atributo discart_pile se inicializa con la factoria.
6. El atributo talents se inicializa con la factoria.
7. El atributo performance_list se inicializa con la factoria.
8. El atributo reputationList se inicializa con un método privado.
9. El atributo reputation se inicializa a 1.
10. El atributo city se inicializa a null.
11. El atributo weeksToPerformance se inicializa a 1.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
5,7	CollectionsFactory	[List<E>] createList()		NO
6	CollectionsFactory	[Map<K,E>] createMap()		NO
8	PlayerImpl	[List<Integer>] InicializateReputation()		NO

Método de alto nivel

PlayerImpl(String nombre)

Diagrama de Colaboración (Opcional)

Constructor BankruptCircusImpl:

Identificador	Descripción de la acción de alto nivel
E003	Constructor de la clase BankruptCircusImpl

Pasos (usar pseudocódigo o similar)

1. Inicializar lista de talentos (talentCircus).
2. Inicializa el color y la descripción.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	PerformanceImpl	Super()		NO

Método de alto nivel

[BackruptCircusImpl] BankruptCircusImpl()

Diagrama de Colaboración (Opcional)

Constructor CircusTrainGame:

Identificador	Descripción de la acción de alto nivel
T-001	Constructor de la clase CircusTrainGameImpl

Pasos (usar pseudocódigo o similar)

1. Inicializar lista de jugadores (playerList).
2. Crear tablero (board).
3. Inicializar bolsa de actuaciones (performanceBag).
4. Inicializar bolsa de talentos (talentBag).
5. Preparar un talento de tipo Payaso para ser añadido a los jugadores.
6. Crear un estado de juego inicial, que es verde

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CollectionsFactory	[List<Player>] createListFactory.createList( )		NO
2	GameFactory	[Board] createBoard( )		NO
3	GameFactory	[PerformanceBag] createPerformanceBag( )		NO
4	GameFactory	[TalentBag] createTalentBag( )		NO
7	GreenState	[GreenState] GreenState (game:CircusTrainGame)		NO

Método de alto nivel

[CircusTrainGame] CircusTrainGameImpl()

Diagrama de Colaboración (Opcional)

Método gameOver AdvancedSingleGame:

Identificador	Descripción de la acción de alto nivel
T-002	Método para recuento final

Pasos (usar pseudocódigo o similar)
1. Se llama al método gameOver de la clase padre 2. Se comprueba qué jugador tiene más reputación

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CircusTrainGameImpl	[void] finalWage()		NO
2	CircusTrainGameImpl	[void] higherReputation()		NO

Método de alto nivel
[void] gameOver()

Diagrama de Colaboración (Opcional)

Método gameOver OnePlayerGame:

Identificador	Descripción de la acción de alto nivel
T-002	Método para recuento final

Pasos (usar pseudocódigo o similar)

1. Comprobar si los jugadores tienen carta de salario sin jugar
2. Comprobar si los jugadores tienen payasos o animales

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CircusTrainGameImpl	[void] finalWage()		NO
2	CircusTrainGameImpl	[void] noClownNoAnimals()		NO

Método de alto nivel

[void] gameOver()

Diagrama de Colaboración (Opcional)

Método gameOver TwoPlayersGame:

Identificador	Descripción de la acción de alto nivel
T-002	Método para recuento final de puntos de victoria

Pasos (usar pseudocódigo o similar)

1. Comprobar si los jugadores tienen carta de salario sin jugar
2. Comprobar qué jugador tiene más payasos.
3. Comprobar qué jugador tiene más dinero.
4. Comprobar qué jugador tiene mayor puntuación máxima.
5. Comprobar si los jugadores tienen payasos o animales

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CircusTrainGameImpl	[void] finalWage()		NO
2	CircusTrainGameImpl	[void] higherClownNumber()		NO
3	CircusTrainGameImpl	[void] higherMoneyAmount()		NO
4	CircusTrainGameImpl	[void] higherPerformancesNumber()		NO
5	CircusTrainGameImpl	[void] noClownNoAnimals()		NO

Método de alto nivel

[void] gameOver()

Método gameOver:

Identificador	Descripción de la acción de alto nivel
T-002	Método para recuento final de puntos de victoria

Pasos (usar pseudocódigo o similar)

1. Comprobar si los jugadores tienen carta de salario sin jugar
2. Comprobar qué jugador tiene más payasos.
3. Comprobar qué jugador tiene más dinero.
4. Comprobar qué jugador tiene mayor puntuación máxima.
5. Comprobar si los jugadores tienen payasos o animales

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CircusTrainGameImpl	[void] wageCardNoDiscarded()		NO
2	CircusTrainGameImpl	[void] higherClownNumber()		NO
3	CircusTrainGameImpl	[void] higherMoneyAmount()		NO
4	CircusTrainGameImpl	[void] higherPerformanceNumber()		NO
6	CircusTrainGameImpl	[void] noClownNoAnimals()		NO

Método de alto nivel

[void] gameOver()

Diagrama de Colaboración (Opcional)

Método finalMonth BasicTwoPlayersGame:

Identificador	Descripción de la acción de alto nivel
T-003	Método que realiza las acciones correspondientes al cambio de mes.

Pasos (usar pseudocódigo o similar)
1. Se comparan ambos jugadores y se añaden los puntos de victoria pertinentes 2. Se intenta robar un talento siguiendo los criterios del modo de juego.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	TwoPlayersGame	[void] comparePlayersAndAddVictoryPoints		NO
2	BasicTwoPlayersGame	[void] stealTalent( )		NO

Método de alto nivel
[void] finalMonth()

Diagrama de Colaboración (Opcional)

Método runGame:

Identificador	Descripción de la acción de alto nivel
T-004	Método núcleo del juego

Pasos (usar pseudocódigo o similar)

1. Mostrar en qué ciudades hay actuaciones.
2. Selección de ciudad canadiense para comenzar.
3. Mientras la semana sea menor que 27, pasos 4,5,6,7,8,9,10,11. Si no se salta al paso 12.
4. Mostrar en qué ciudades hay actuaciones.
5. Recorrer la lista de jugadores.
6. Mostrar estado actual del jugador.
7. Preguntar qué acción va a realizar el jugador a continuación.
8. Se ejecuta la acción que el jugador ha seleccionado.
9. Se ejecuta el convertidor de puntos.
10. Se ejecuta el comprobador de baraja vacía.
11. Incremento en 1 del contador de semanas en el estado. Se vuelve al paso 3.
12. Se hace el recuento final
13. Tras hacer el recuento final se muestra por pantalla el resultado del juego

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CircusTrainGame	[void] showPerformanceSituation( )		NO
2	CircusTrainGame	[void] canadianSelector( )		NO
4	CircusTrainGame	[void] showPerformanceSituation( )		NO
6	PlayerImpl	[void] playerState( )		NO
7	CircusTrainGame	[void] selectCase (player:Player)		NO
8	CircusTrainGame	[void] executeCase (player:Player)		NO
9	CircusTrainGame	[void] pointsConversor (player:Player)		NO
10	CircusTrainGame	[void] checkEmptyDeck (player:Player)		NO
11	GameState	[void] incrementTime( )		NO

12	CircusTrainGameImpl	[void] gameOver( )		NO
13	CircusTrainGameImpl	[void] results( )		NO
Método de alto nivel				
[void] runGame()				
Diagrama de Colaboración (Opcional)				

Método refreshMonth:

Identificador	Descripción de la acción de alto nivel
T-006	Método que cambia el mes según la semana en la que se esté

Pasos (usar pseudocódigo o similar)
1. Si la semana está entre 0 y 3 el mes es abril 2. Si la semana está entre 4 y 8 el mes es mayo 3. Si la semana está entre 9 y 12 el mes es junio 4. Si la semana está entre 13 y 17 el mes es julio 5. Si la semana está entre 18 y 21 el mes es agosto 6. Si la semana está entre 22 y 26 el mes es septiembre 7. Cambio de primer jugador
Diagrama de estados (Opcional)
Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
7	CircusTrainGame	[void] rotatePlayers()		NO
Método de alto nivel				
[void] refreshMonth()				
Diagrama de Colaboración (Opcional)				

Método compareTalentsCountAndAddVictoryPoints:

Identificador	Descripción de la acción de alto nivel
T-007	Método que compra el número de talentos de cada jugador y suma los puntos de victoria pertienentes

Pasos (usar pseudocódigo o similar)

1. Referenciar ambos jugadores.
2. Referenciar contadores de talentos.
3. Creación de todos los tipos de talentos.
4. Creación de un Set para contener los talentos.
5. Añadir los talentos.
6. Iterar sobre el set de talentos. Para cada iteración, pasos
7. Inicializar contadores de talentos a 0.
8. Si el jugador 1 tiene el talento, se actualiza su contador con el número de talentos del tipo.
9. Si el jugador 2 tiene el talento, se actualiza su contador con el número de talentos del tipo.
10. Si al comparar ambos contadores (contadorPlayer1-contadorPlayer2) sale positivo, añadir tantos puntos de victoria al jugador 1 como talentos del tipo tenga.
11. Si al comparar ambos contadores (contadorPlayer1-contadorPlayer2) sale negativo, añadir tantos puntos de victoria al jugador 2 como talentos del tipo tenga.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
3	GameFactory	[Talent] createTalent (type:String)		NO

Método de alto nivel

[void] compareTalentsCountAndAddVictoryPoints()

Diagrama de Colaboración (Opcional)

Método completeBoardPerformances:

Identificador	Descripción de la acción de alto nivel
T-008	Método que añade las actuaciones necesarias para que se cumpla el número según el mes en el que se encuentre

Pasos (usar pseudocódigo o similar)

1. Seleccionar una Performance aleatoria perteneciente al estado del juego.
2. Si la Performance es un BankruptCircus, pasos 4,5,6,7
3. Recorrer todos los talentos que debe tener
4. Si hay talentos del tipo en la bolsa de talentos, los saca.
5. Si no hay talentos del tipo borra el talento del circo en cuestión.
6. Se añade el circo al tablero.
7. Si la Performance no es un BankruptCricus, se añade al tablero.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
3	GameState	[Performance] getPerformance( )		NO
4	TalentBag	[Integer] removeTalent (talent:Talent)		NO
6	Board	[void] addPerformanceinRandomCity (performance:Performance)		NO
7	Board	[void] addPerformanceinRandomCity (performance:Performance)		NO

Método de alto nivel

[void] completeBoardPerformances( )

Diagrama de Colaboración (Opcional)

Método stealTalent:

Identificador	Descripción de la acción de alto nivel
J-001	Método para robar un talento de un jugador a otro según la modalidad de juego

Pasos (usar pseudocódigo o similar)

1. Referenciar la lista de jugadores e inicializar el otro jugador a null
2. Recorrer la lista de jugadores
3. Si el jugador es distinto del que se ha pasado como parámetro, reemplazar el null.
4. Obtener los talentos del otro jugador y se meten en una lista.
5. Inicializar los strings de las posibles opciones y las condiciones;
6. Obtener el Iterator del set de talentos del otro jugador.
7. Completamos los Strings de opciones y condiciones utilizando el método next del Iterator dentro de un for clásico.
8. Inicializar la pregunta con las opciones.
9. Preguntar al usuario qué talentos del otro jugador va a robar.
10. Crear una lista con el índice de los talentos a partir de las opciones que se han preguntado.
11. Recorrer la lista de los talentos que se van a robar.
12. Si el talento del otro jugador que se ha querido robar coincide con el de la lista, pasos 13,14,15,16,17
13. Referenciar el talento en cuestión.
14. Descartar el talento del otro jugador.
15. Crear una lista y añadir ahí el talento referenciado.
16. Añadir la lista del paso 15 al jugador pasado como parámetro.
17. Salir del bucle.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
4	PlayerImpl	[Map<Talent,Integer>] getTalents( )		NO
9	takeDataFromKeyboard	[String] takeParametersToStringRestricted (message:String,conditions:String)		SI
14	PlayerImpl	[void] discardTalent (talent:Talent)		NO
16	PlayerImpl	[void] addTalents (talentList:List<Talent>)		NO

Método de alto nivel

[void] stealTalent (player:Player)

Método TalentImpl:

Identificador	Descripción de la acción de alto nivel			
F-001	Constructor de la clase TalentImpl			
Pasos (usar pseudocódigo o similar)				
1. Los constructores específicos de cada talento se inicializarán heredando del constructor de TalentImpl con el salario correspondiente.				
Diagrama de estados (Opcional)				
Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU
Método de alto nivel				
[Talent] TalentImpl (Integer w)				
Diagrama de Colaboración (Opcional)				

Método TalentBag:

Identificador	Descripción de la acción de alto nivel
F-002	Constructor de la clase TalentBagImpl

Pasos (usar pseudocódigo o similar)

1. Crear la bolsa de talentos.
2. Crear el número de talentos que habrá al inicio en una bolsa de talentos.
3. Añadirlos a la bolsa de talentos.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	Collections Factory	[Map<Talent,Integer>] createMapFactory.createMap( )		NO
2	NombreTalentoImpl	[Talent] NombreTalentoImpl()		NO
3	Collections Factory	[Void] put(Talent, Integer)		NO

Método de alto nivel

[TalentBag] TalentBagImpl()

Diagrama de Colaboración (Opcional)

INFORME TÉCNICO Nº T-000
Asunto: Serias dificultades para jerarquizar las diferentes acciones que transcurren en tiempo de ejecución
Resumen de la solución: Implementación del patrón de diseño comandos
Factores causantes: Serias dificultades para modelar el comportamiento en tiempo de ejecución y la actualización de los atributos de los objetos implicados de forma dinámica
Solución: Implementación de patrón de diseño comandos. Se ha creado una clase abstracta AbstractCommand con un único método abstracto denominado execute. Los diferentes comandos son clases que extienden de ésta y, aparte de tener un constructor, deben implementar el método para un uso específico. El comando recibe los elementos con los que va a trabajar a través del constructor, y en el método execute heredado define su comportamiento mediante los métodos de dichos parámetros pasados anteriormente
Motivación: Facilidad de su implementación; poder realizar acciones en tiempo de ejecución de manera cómoda, limpia y elegante; permite la jerarquización de las acciones que se van a realizar; muy útil para la implementación de interfaces gráficas de usuario; aislamiento.
Cuestiones sin resolver: incertidumbre a la hora de considerar ciertos métodos de la clase CircusTrainGameImpl como comandos aislados.
Alternativas consideradas:

Comandos en Performance:

INFORME TÉCNICO Nº T-002
Asunto: Comandos en Performance
Resumen de la solución: Aplicación del patrón comandos a las clases hijas de Performance
Factores causantes: Complejidad para llevar a cabo acciones relacionadas con las distintas actuaciones dentro de las cartas de acción.
Solución: Se le ha añadido a la interfaz Performance el método execute con un parámetro de tipo Player. Se les ha añadido un atributo de tipo Player a las cartas de acción para que puedan hacer referencia al jugador y modificar su estado durante la ejecución. Al seleccionar una carta de acción que interactue con una Performance se le pasará como parámetro al método execute el atributo de tipo Player de las cartas de acción para que sea modificado en tiempo de ejecución.
Motivación: Facilidad de implementación y evitar estructuras comparativas complejas contempladas en la lista de malos olores.
Cuestiones sin resolver:
Alternativas consideradas:

Impl. Colores:

INFORME TÉCNICO Nº 1
Asunto: Implementación de los colores usando la clase color de java.awt que da problemas en los toString
Resumen de la solución: Cambiar los colores a toString
Factores causantes: Color de java.awt no está pensada para clasificar algo por colores, lo cual hace dificultoso mostrar la información que facilita
Solución: Cambiamos las descripciones con un string que tomará los valores “Green”,”Yellow” o “Red”
Motivación: Parece la más sencilla de implementar
Cuestiones sin resolver: No hay
Alternativas consideradas: Se ha planteado usar un enumerado, pero parecía más dificultosa la implementación.

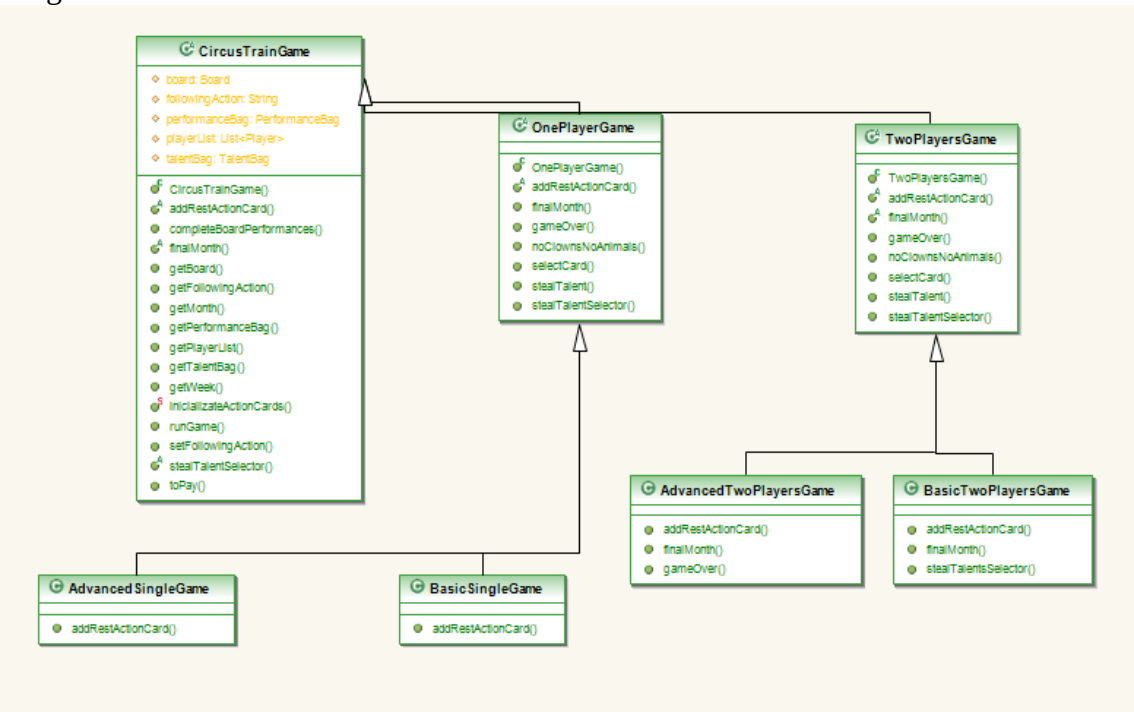
Eliminación métodos entrada por teclado GameFactory:

INFORME TÉCNICO Nº A1
Asunto: Eliminación de los métodos de entrada por teclado de la GameFactory
Resumen de la solución: Movidos a la clase ReadDataFromKeyboard del paquete utiles
Factores causantes: Dentro de la gameFactory solo debe haber métodos destinados a la creación de los objetos del juego, eliminando de ahí los métodos genéricos usados para la entrada de teclado
Solución: Se ha creado la clase ReadDataFromKeyboard dentro del paquete útiles y se han movido los métodos que había dentro de GameFactory relativos a la entrada por teclado.
Motivación: Mantener limpia la gameFactory
Cuestiones sin resolver: No hay
Alternativas consideradas: No se han considerado

Creación árbol herencia clase Game:

INFORME TÉCNICO Nº A3
Asunto: Falla la herencia en la clase juego
Resumen de la solución: Creado un árbol de herencia para la clase juego.
Factores causantes: Dentro de la clase del juego, se han aplicado soluciones para cada uno de los modos de juegos posibles, sin tener en cuenta que deberían haberse creado en clases distintas, que heredaran de una que tuviera la parte común.
Solución: Se han creado cuatro clases que heredan de CircusTrainImpl, cada una con un modo de juego
Motivación: Reutilización de código común y protección contra el cambio
Cuestiones sin resolver: Análisis detallado de que métodos/comandos deben ir en cada una de las clases. Creación de una clase para un jugador y otra para dos jugadores y de cada una de ellas deben heredar básico y avanzado.
Alternativas consideradas:

Diagrama de clases de Game



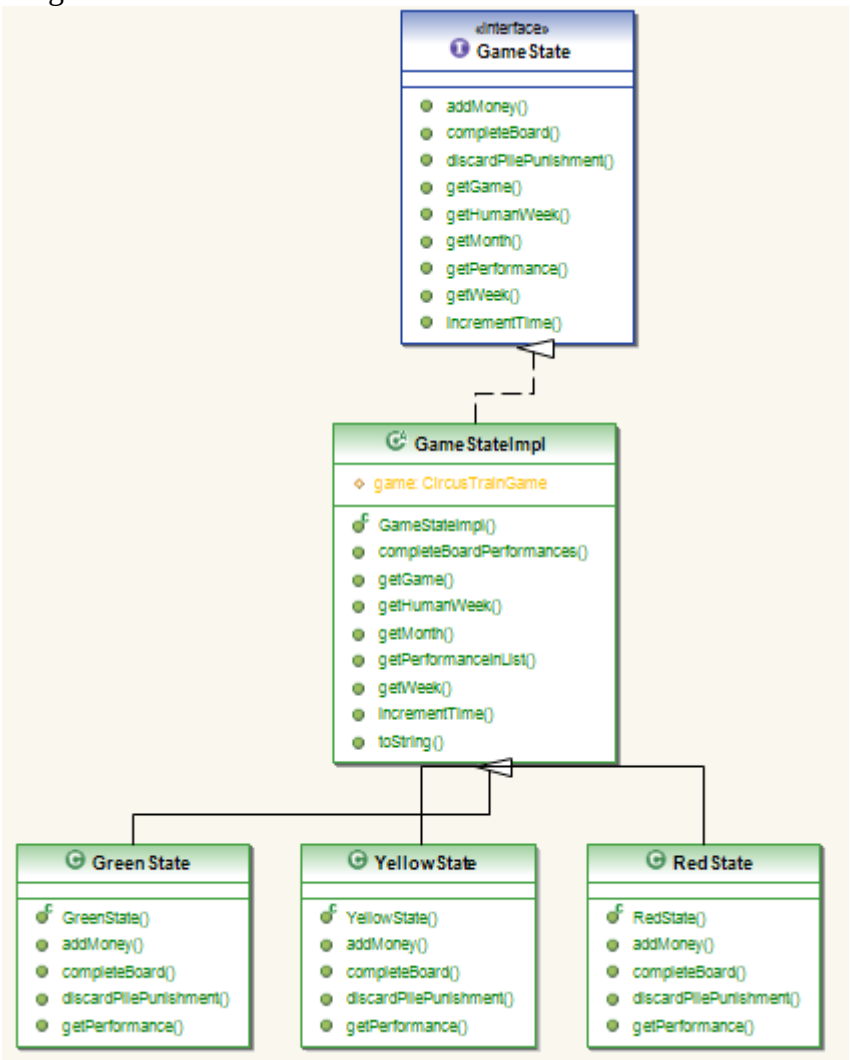
Eliminación entrySet:

INFORME TÉCNICO Nº A4
Asunto: Quitado entryset
Resumen de la solución: Sustitución por uso de los métodos propios de map
Factores causantes: A lo largo del juego se ha usado una mala solución de acceder a los datos contenidos en un mapa a través del paso intermediode usar los entryset en lugar de usar los métodos propios del mapa. (put,get)
Solución: Se han sustituidos las estructuras creadas con entryset por estructuras con mapas.
Motivación: Facilitar la lectura del código
Cuestiones sin resolver:
Alternativas consideradas: Dejarlo como estaba, ya que funcionaba.

Aplicación patrón estado:

INFORME TÉCNICO Nº A5
Asunto: Aplicación del patrón estado.
Resumen de la solución: Sustituciones de la comprobación de color por el patrón estado.
Factores causantes: Para comprobar en que fase del juego estábamos (verde,amarilla o naranja) se estaba comparando con estas palabras con series if-else encadenados.
Solución: Se ha implementado el patrón estado y dentro de este se lleva la contabilidad del tiempo, y de cada una de las fases en las que nos encontramos y realizamos las acciones adecuadas en cada una de ellas. Se ha trasladado a estado las acciones de completar el tablero y obtener performances de las bolsas de performances, pues estas también dependen del estado,
Motivación: Hacer el código mantenible y de fácil comprensión.
Cuestiones sin resolver:
Alternativas consideradas: Dejarlo como estaba, ya que funcionaba.

Diagrama del GameState



Limitación opciones al usuario:

1. INFORME TÉCNICO Nº AC-002

Asunto: Quitar opciones no posibles para la interfaz de usuario en las ActionCard's
Resumen de la solución: Se han evaluado las posibilidades de algunos métodos antes de mostrar al usuario la opción a elegir.
Factores causantes: Código no estructurado.
Solución: Se han puesto condiciones evaluadoras antes de poder mostrar al usuario la opción a usar.
Motivación: Evitar que el usuario pueda elegir una opción que no es posible en ese momento en el juego, evitando posibles errores de segmentación y protegiendo el código.
Cuestiones sin resolver: Falta por aplicarlo en la carta RESTImpl, necesito preguntarlo a la persona que hizo ese código.
Alternativas consideradas: Ser un poquito más ordenado a la hora de programar.

Refactorización de ActionCard:

INFORME TÉCNICO Nº AC-001

Asunto: Refactorización de los métodos execute's del paquete ActionCard
Resumen de la solución: Se ha reducido considerableme el código de las cartas de acción, haciendolo mucho más legible
Factores causantes: Gran vulnerabilidad ante el cambio. Patrón de diseño comandos mal aplicado. Casi nula opción de reutilizar código. Demasiada repetición de mismos fragmentos de código
Solución: Se ha cambiado la herencia, antes todas las cartas de acción heredaban de CardImpl, ahora todas heredan de ActiónCardImpl. Los la parte de código que hacía mover a un jugador y actuar/contratar se ha metido en un metodo “movePlayer” y “performPlayer” respectivamente. Este método se ha llevado a la clase ActionCardImpl, donde es usado, por todas las cartas de Acción.
Motivación: Proteger el software ante el cambio y tener una estructura de datos mucho más clara para facilitar su mantenimiento.
Cuestiones sin resolver: Dar al usuario opciones que no son posibles
Alternativas consideradas: Patrón de diseño Command.

Diagrama de las ActionCard

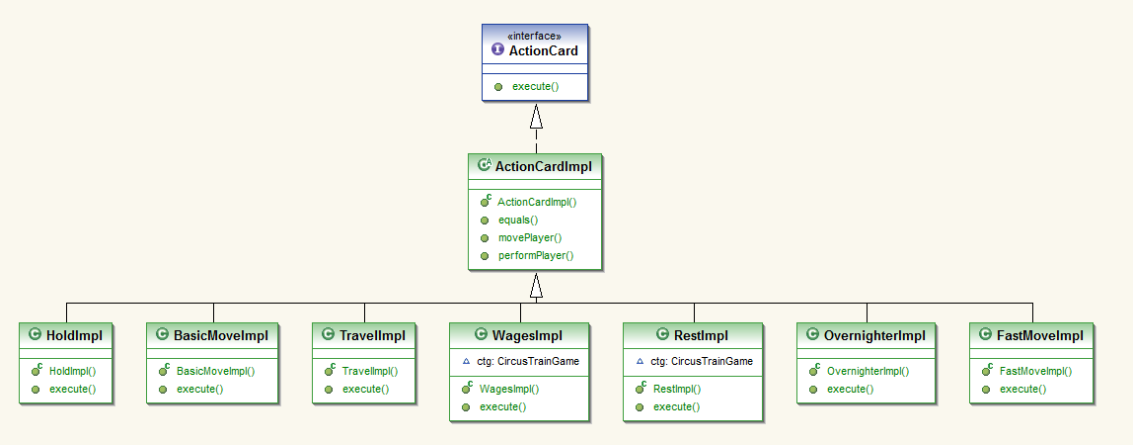


Diagrama de Player

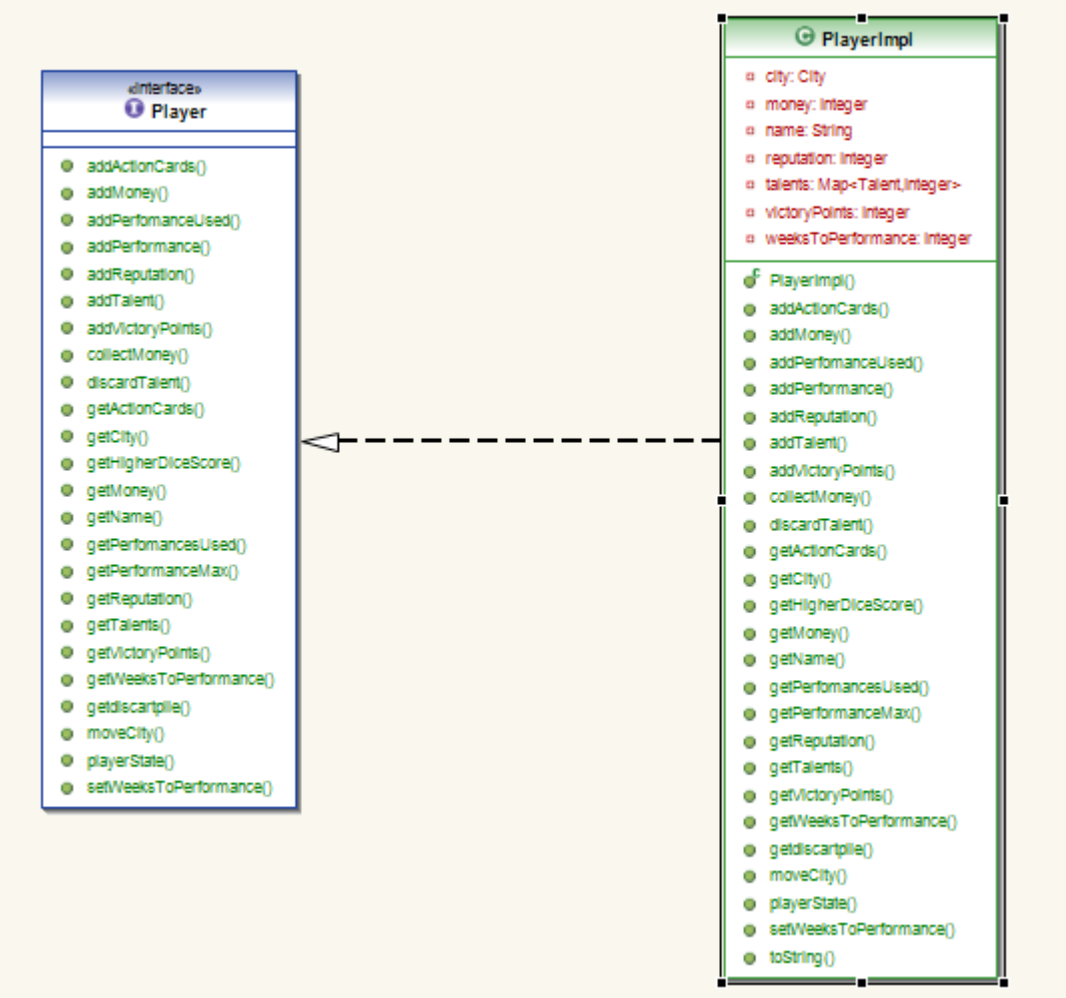
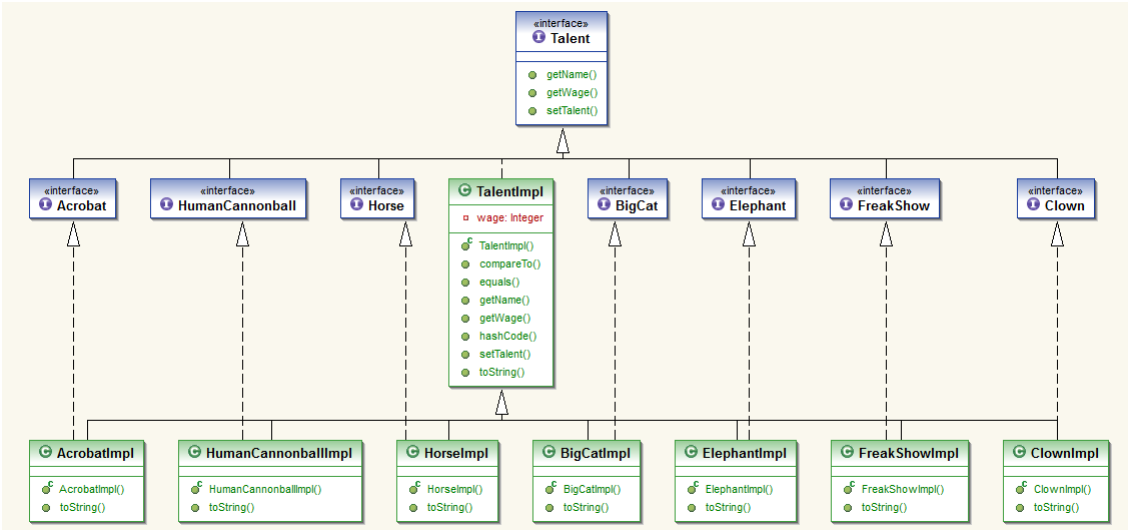


Diagrama de Talent



5.3 Iteración 5

Añadidos en iteración 5

- Reestructuración de las clases del juego (refactorización)
- Documentos de asignación de responsabilidades (a continuación).
- Memorandos técnicos (a continuación).
- Integración de los componentes del juego.
- Implementación casi total de los modos de juego pendientes.

Asuntos pendientes en iteración 6

- Pruebas unitarias.
- Terminar de implementar los modos de juego.
- Documentar las estructuras que faltan y las que puedan surgir.
- Arreglar bugs en el funcionamiento del juego.

Método higherClowNumber:

Identificador	Descripción de la acción de alto nivel
T-009	Método para ver qué jugador tiene más payasos y añadir puntos de victoria si no hay empate

Pasos (usar pseudocódigo o similar)

1. Inicializar los contadores de payasos de ambos jugadores a 0.
2. Crear un payaso para comparar.
3. Si el jugador 1 contiene payasos, el contador de payasos del jugador 1 se actualizará con la cantidad de talentos de este tipo.
4. Si el jugador 2 contiene payasos, el contador de payasos del jugador 2 se actualizará con la cantidad de talentos de este tipo.
5. Si al comparar ambas cifras (contadorPayasos1-contadorPayasos2) sale positivo, añadir 3 puntos de victoria al jugador 1
6. Si al comparar ambas cifras (contadorPayasos1-contadorPayasos2) sale negativo, añadir 3 puntos de victoria al jugador 2

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel

[void] higherClownNumber()

Diagrama de Colaboración (Opcional)

Método noClownsNoAnimals:

Identificador	Descripción de la acción de alto nivel
T-010	Método que resta puntos si el jugador no tiene ni payasos ni animales

Pasos (usar pseudocódigo o similar)

1. Recorrer los talentos del jugador
2. Si no tiene payasos restar puntos de victoria
3. Si no tiene animales (caballos, grandes felinos o elefantes) restar 3 puntos de victoria
4. Si no tiene más de 3 puntos de victoria se pondrán a 0.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
				NO

Método de alto nivel

[void] noClownsNoAnimals

Diagrama de Colaboración (Opcional)

Método higherMoneyAmount:

Identificador	Descripción de la acción de alto nivel
T-011	Método que compara la cantidad de dinero de los jugadores y suma puntos de victoria al que más tenga si no hay empate.

Pasos (usar pseudocódigo o similar)

1. Creación de una lista para guardar las cantidades de dinero manteniendo el orden en que se tratan los jugadores.
2. Definición de contadores de dinero previo y empates a 0 y de dinero máximo a -1
3. Recorrer la lista de jugadores
4. Guardar la cantidad de dinero del jugador anterior en el contador de dinero del jugador previo.
5. Añadir la cantidad de dinero del jugador actual en la lista.
6. Si el contador de dinero máximo es menor que el dinero del jugador actual, el contador se actualiza con el dinero de éste.
7. Si la cantidad de dinero del jugador actual y del previo son iguales el contador de empates se incrementa.
8. Si el contador de empates es 0 se obtiene el índice de la cantidad máxima guardada en la lista.
9. Selección del jugador de la lista de jugadores mediante el índice del paso 8 e incrementar los puntos de victoria.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
				NO

Método de alto nivel

[void] higherMoneyAmount()

Método higherPerformanceNumber:

Identificador	Descripción de la acción de alto nivel
T-012	Método que compara el número de actuaciones de los jugadores y suma puntos de victoria al que más tenga si no hay empate.

Pasos (usar pseudocódigo o similar)

1. Referenciar a ambos jugadores.
2. Extraer el tamaño de las listas de cada jugador que contienen las actuaciones que han desempeñado.
3. Comparar ambos tamaños restando tamañoListaJugador1-tamñoListaJugador2
4. Si el resultado es positivo añadir 3 puntos de victoria al jugador 1.
5. Si el resultado es negativo añadir 3 puntos de victoria al jugador 2

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
				NO

Método de alto nivel

[void] higherPerformancesNumber()

Diagrama de Colaboración (Opcional)

Método higherReputation:

Identificador	Descripción de la acción de alto nivel
T-013	Método que compara la reputación de los jugadores y suma puntos de victoria al que más tenga si no hay empate.

Pasos (usar pseudocódigo o similar)

1. Creación de una lista para guardar la reputación manteniendo el orden en que se tratan los jugadores.
2. Definición de contadores reputación del jugador previo y empates a 0 y de reputación máxima a -1
3. Recorrer la lista de jugadores
4. Guardar la reputación del jugador anterior en el contador de reputación del jugador previo.
5. Añadir la reputación del jugador actual a la lista.
6. Si el contador de reputación máxima es menor que el de reputación del jugador actual, el contador se actualiza con la cantidad de éste.
7. Si la reputación del jugador actual y del previo son iguales el contador de empates se incrementa.
8. Si el contador de empates es 0 se obtiene el índice de la cantidad máxima guardada en la lista.
9. Selección del jugador de la lista de jugadores mediante el índice del paso 8 e incrementar los puntos de victoria.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
				NO

Método de alto nivel

[void] higherReputation()

Diagrama de Colaboración (Opcional)

Método selectCanadianCity:

Identificador	Descripción de la acción de alto nivel
T-014	Devuelve la ciudad canadiense en la que quiere empezar un jugador

Pasos (usar pseudocódigo o similar)

1. Inicialización de una lista con las ciudades canadienses para elegir.
2. Generación de la pregunta y de las condiciones mediante un bucle para su posterior uso.
3. Se le pregunta al usuario en qué ciudad va a empezar usando las dos variables de tipo String obtenidas en el paso 2.
4. Se busca la ciudad seleccionada en la lista mediante el índice y se retorna.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	readDataFromKeyBoard	[String] takeParametersToStringRestricted(message: String, condition: String)		SI

Método de alto nivel

[void] selectCanadianCity()

Diagrama de Colaboración (Opcional)

Método rotatePlayers:

Identificador	Descripción de la acción de alto nivel
T-015	Invierte el orden de la lista de jugadores

Pasos (usar pseudocódigo o similar)

1. Creación de lista auxiliar para almacenar ahí los jugadores en el orden inverso
2. Se recorre la lista de jugadores sin contar el primero y se van añadiendo a la lista auxiliar definida en el paso 1.
3. Se le añade a la auxiliar de jugadores el jugador que ocupa la primera posición en la lista del juego.
4. Se vacía la lista de jugadores y se vuelca en ella la auxiliar

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1				NO

Método de alto nivel

[void] rotatePlayers()

Diagrama de Colaboración (Opcional)

Método monthChangeComprobator:

Identificador	Descripción de la acción de alto nivel
T-016	Comprueba si se ha cambiado de mes y actúa en consecuencia

Pasos (usar pseudocódigo o similar)

1. Guardar en una variable el mes antes de actualizar el mes
2. Actualización del mes
3. Guardar en una variable el mes tras actualizar
4. Si son distintos invoca al método que cambia el estado de los elementos en cuestión
5. Si son distintos invoca al método que intercambia el orden de los jugadores

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	CircusTrain Game	[void] refreshMonth( )		NO
4	CircusTrain Game	[void] finalMonth( )		NO
5	CircusTrain Game	[void] rotatePlayers( )		NO

Método de alto nivel

[void] monthChangeComprobator( )

Diagrama de Colaboración (Opcional)

Método canadianSelector:

Identificador	Descripción de la acción de alto nivel
T-017	Establece la ciudad canadiense en la que empezara un jugador

Pasos (usar pseudocódigo o similar)

1. Recorrer la lista de jugadores
2. Mover cada jugador a la ciudad canadiense seleccionada

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	Player	[void] moveCity (city: City)		NO

Método de alto nivel

[void] canadianSelector( )

Diagrama de Colaboración (Opcional)

Método results1jugador:

Identificador	Descripción de la acción de alto nivel
T-018	Se comprueba el dinero resultante tras el recuento final y se muestra el resultado

Pasos (usar pseudocódigo o similar)
1. Obtención del dinero tras el recuento final. 2. Según la cantidad de dinero se mostrara por pantalla el resultado correspondiente

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel
[void] results( )

Diagrama de Colaboración (Opcional)

Método results2jugadores:

Identificador	Descripción de la acción de alto nivel
T-018	Método que compara los puntos de victoria y de máxima actuación de los jugadores y nombra al ganador o anuncia que hay empate.

Pasos (usar pseudocódigo o similar)

1. Creación de una lista para guardar los puntos de victoria manteniendo el orden en que se tratan los jugadores.
2. Creación de una lista para guardar los puntos de máxima actuación en el orden en que se tratan los jugadores.
3. Definición de contadores de puntos de victoria del jugador previo, empates por puntos de victoria y cantidad máxima de puntos de victoria a 0
4. Definición de contadores de puntos de máxima actuación del jugador previo, empates por puntos de max. actuación y cantidad máxima de puntos de max. actuación a 0
5. Recorrer la lista de jugadores
6. Guardar los puntos de victoria del jugador anterior en el contador de puntos de victoria del jugador previo.
7. Guardar los puntos de máxima actuación del jugador anterior en el contador de puntos de actuación máxima del jugador previo.
8. Añadir los puntos de victoria del jugador actual a la lista de puntos de victoria.
9. Añadir los puntos de máxima actuación del jugador actual a la lista de puntos de máxima actuación.
10. Si el contador de cantidad máxima de puntos de victoria es menor que el de puntos de victoria del jugador actual, el contador se actualiza con la cantidad de éste.
11. Si el contador de cantidad máxima de puntos de máxima actuación es menor que el de puntos de max. actuación del jugador actual, el contador se actualiza con la cantidad de éste.
12. Si los puntos de victoria del jugador actual y del previo son iguales el contador de empates de puntos de victoria se pone a 1.
13. Si los puntos de max. actuación del jugador actual y del previo son iguales el contador de empates puntos de máxima actuación se pone a 1.
14. Si el contador de empates de puntos de victoria es 0 se obtiene el índice de la cantidad máxima guardada en la lista. Si no se salta al paso 16
15. Selección del jugador de la lista de jugadores y anunciarlo como ganador.
16. Si el contador de empates de puntos de máxima actuación es 0 se selecciona el jugador de la lista de jugadores y se anuncia como ganador. Si no se anuncia que el juego ha resultado en tablas.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
				NO

Método de alto nivel

[void] results()

Método getName TalentImpl:

Identificador	Descripción de la acción de alto nivel
F-003	Método para devolver el nombre de la interfaz de un objeto pasado por parámetro. (Nos ahorramos los toString de cada tipo de talento).

Pasos (usar pseudocódigo o similar)

1. Crear un array de objetos tipo Class<?> con los nombres de las interfaces que implementa la clase del objeto que llama al método.
2. En nuestro caso, al implementar solo una interfaz, tan solo nos basta con sacar el primer elemento del array, que contendrá el nombre de la interfaz.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	Class	[Class<?>[]] Class.getClass().getInterfaces()		NO
2	Class	[String] Class.getSimpleName()		

Método de alto nivel

[String] toString()

Diagrama de Colaboración (Opcional)

Método toString TalentBagImpl:

Identificador	Descripción de la acción de alto nivel
F-004	Método para convertir en cadena los TalentBag

Pasos (usar pseudocódigo o similar)

1. Recorrer la bolsa de talentos.
2. Por cada tipo de talento, llamar al método que devolverá el nombre de la interfaz del objeto con el que se llama y devolver su nombre junto con el número de talentos.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	TalentImpl	[String] getName()		NO

Método de alto nivel

[String] toString()

Diagrama de Colaboración (Opcional)

Método toString TalentImpl:

Identificador	Descripción de la acción de alto nivel
F-005	Método para convertir en cadena los Talents
Pasos (usar pseudocódigo o similar)	
1. Llamar al método que devolverá el nombre de la interfaz del objeto con el que se llama.	
Diagrama de estados (Opcional)	
Métodos de bajo nivel necesarios	

Paso	Clase	Método	Mem. Técn.	IU
1	TalentImpl	[String] getName()		NO

Método de alto nivel
[String] toString()
Diagrama de Colaboración (Opcional)

Método execute FastMove:

Identificador	Descripción de la acción de alto nivel
AC-004	Permite realizar la acción de un Movimiento Rápido
Pasos (usar pseudocódigo o similar)	
1. Mueve al jugador tantos pasos como la cara de acción diga.	
Diagrama de estados (Opcional)	
Métodos de bajo nivel necesarios	

Paso	Clase	Método	Mem. Técn.	IU
1	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO

Método de alto nivel
[void] execute (gameState:GameState);
Diagrama de Colaboración (Opcional)

Método execute Hold:

Identificador	Descripción de la acción de alto nivel			
AC-005	Permite realizar la acción de un Descanso			
Pasos (usar pseudocódigo o similar)				
1. Realiza la acción de actuar o contratar.				
Diagrama de estados (Opcional)				
Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU
1	ActionCardImpl	[void] performPlayer (gameState:GameState)	AC-002	NO
Método de alto nivel				
[void] execute (gameState:GameState);				
Diagrama de Colaboración (Opcional)				

Método execute OverNighter:

Identificador	Descripción de la acción de alto nivel
AC-006	Permite realizar la acción de la carta “De Noche”

Pasos (usar pseudocódigo o similar)

1. Obtiene la ciudades adyacentes y calcula si hay performance en alguna de ellas
2. Recibe opción de teclado.
3. Según posibilidades, el metodo realiza alguna/s de las siguientes acciones:

3.1 mover a un jugador
3.2 Viajar primero y luego Actuar/contratar
3.3 Actuar/contratar

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	City	[List<City>] maxMovement (jump:Integer)		NO
2	ReadDataFromTheKeyboard	[String] takeParametersToIntegerRestricted (menssage:String, condition: String)		SI
3.1	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO
3.2	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO
3.2	ActionCardImpl	[void] performPlayer (gameState:GameState)	AC-002	NO
3.3	ActionCardImpl	[void] performPlayer (gameState:GameState)	AC-002	NO

Método de alto nivel

[void] execute (gameState:GameState);

Diagrama de Colaboración (Opcional)

Método execute Rest:

Identificador	Descripción de la acción de alto nivel
AC-007	Permite realizar la acción de la carta Descanso

Pasos (usar pseudocódigo o similar)

1. Saber si el Jugador esta en Canadá y en su caso, obtener el talento a contratar.
2. Mirar si hay talentos en “TalentBag”
3. Quitar el talento de la bolsa
4. Añadir el talento al jugador
5. Aumenta la reputación en el caso que sea posible

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	ReadDataFromTheKeyboard	[String] takeParametersToIntegerRestricted (menssage:String, condition: String)		SI
2	TalentBag	[Integer] getNumTypeTalent (t: Talent)		NO
3	TalentBag	[Talent] removeTalent (t: Talent)		NO
4	Player	[void] addTalent(List<Talent> t)		NO
5	Player	[void] addReputation (reputation:Integer)		NO

Método de alto nivel

[void] execute (gameState:GameState);

Diagrama de Colaboración (Opcional)

Método execute Travel:

Identificador	Descripción de la acción de alto nivel
AC-008	Permite realizar la acción de la carta Viaje

Pasos (usar pseudocódigo o similar)

1. Mueve al jugador tantos pasos como la carta de acción diga.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO

Método de alto nivel

[void] execute (gameState);

Diagrama de Colaboración (Opcional)

Método execute Wages:

Identificador	Descripción de la acción de alto nivel
AC-009	Permite realizar la acción de la carta Salarios

Pasos (usar pseudocódigo o similar)

1. Mueve al jugador
2. Paga los salarios correspondientes

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO
2	CircusTrainGame	[void] toPay(player:Player,multiplicator:Integer)		NO

Método de alto nivel

[void] execute (gameState:GameState);

Diagrama de Colaboración (Opcional)

Método execute playerEstate:

Identificador	Descripción de la acción de alto nivel
AC-010	Su función es informar al usuario del estado del juego.

Pasos (usar pseudocódigo o similar)

1. Informar al usuario si está en una actuación de dos semanas
2. Informar al usuario de las cartas de acción de la mano.
3. Informar al usuario de las cartas de acción de la pila de descartes si las hay.
4. Informar al usuario de los talentos que posee su circo.
5. Informar al usuario de su reputación actual.
6. Informar al usuario de la tirada máxima para poder contratar.
7. Informar al usuario de su posición actual en el tablero.
8. Informar al usuario del dinero que posee.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU

Método de alto nivel

[void] playerEstate();

Diagrama de Colaboración (Opcional)

Método collectMoney:

Identificador	Descripción de la acción de alto nivel
AC-011	Realiza la acción de dar dinero a cambio de niveles de reputación

Pasos (usar pseudocódigo o similar)

1. Preguntar al usuario cuántos niveles de reputación desea bajar.
2. Dar cinco de dinero por cada nivel de reputación bajado.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	readDataFromKeyBoard	[String]TakeParametersToIntegerRestricted(question:String, restriction:String)		SI
2	Player	[void] addMoney(money:Integer)		NO

Método de alto nivel

[void] collectMoney();

Diagrama de Colaboración (Opcional)

Método addTalent:

Identificador	Descripción de la acción de alto nivel
AC-012	Realiza la acción de meter los talentos contratados al mapa de talentos del jugador. (Map<Talent,Integer>)
Pasos (usar pseudocódigo o similar)	
1. Por cada talento miramos si ya existe en nuestro mapa. 1.1 Si existe sumamos uno al valor. 1.2 Si no existe lo metemos en el mapa con valor 1.	
Diagrama de estados (Opcional)	
Métodos de bajo nivel necesarios	

Paso	Clase	Método	Mem. Técn.	IU

[void] addTalent(List<Talent> talentsToAdd);
Diagrama de Colaboración (Opcional)

Método execute BasicMove:

Identificador	Descripción de la acción de alto nivel
AC-013	Permite realizar la acción de un Movimiento Básico

Pasos (usar pseudocódigo o similar)

1. Obtener la opción de Contratar o moverse del jugador
2. Realizar la funcionalidad según el paso 1 y posibilidades.
   2.1 Moverse
   2.2 Actuar/Contratar

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	ReadDataFromTheKeyboard	[String] takeParametersToIntegerRestricted (menssage:String, condition: String)		SI
2.1	ActionCardImpl	[void] movePlayer (move: Integer)	AC-001	NO
2.2	ActionCardImpl	[void] performPlayer (gameState:GameState)	AC-002	NO

Método de alto nivel

[void] execute (gameState:GameState);

Diagrama de Colaboración (Opcional)

Método toString BankruptCircusImpl:

Identificador	Descripción de la acción de alto nivel
E-006	Establecer un formato adecuado en String

Pasos (usar pseudocódigo o similar)

1. Inicializa stringToReturn.
2. Recorre todos los talentos de la carta y los muestra por pantalla

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	Talent	[Talent] getTalentCircus()		NO
[String] toString( )				

Diagrama de Colaboración (Opcional)

Método toString de PerformanceDemandImpl:

Identificador	Descripción de la acción de alto nivel
E-007	Establecer un formato adecuado en String

Pasos (usar pseudocódigo o similar)

1. Inicializa stringToReturn.
2. Si es de dos semanas , añade “DOS SEMANAS” a la cadena stringToReturn
3. Recorre todos los talentos del Set y devuelve sus puntos.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	PerformanceDemand	[Integer] getBasicPoints()		NO
3	PerformanceDemand	[Boolean] isTwoWeeks()		NO
4	Talent	[Set] getTalentPoints()		NO
[String] toString( )				

Diagrama de Colaboración (Opcional)

Método toString de VictoryPointsImpl:

Identificador	Descripción de la acción de alto nivel
E-008	Establecer un formato adecuado en String

Pasos (usar pseudocódigo o similar)

1. Inicializa stringToReturn.
2. Añade los puntos de victoria a la cadena stringToReturn

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	VictoryPoints	[Integer] getVictoryPoints()		NO
[String] toString( )				

Diagrama de Colaboración (Opcional)

Eliminación de CommandPay:

INFORME TÉCNICO Nº T-003
Asunto: Eliminación de CommandPay
Resumen de la solución: Se ha eliminado la clase CommandPay y, por consiguiente, la interfaz Command.
Factores causantes: Presencia de muchos if/else anidados que hace que el software sea muy vulnerable al cambio, poco legible y difícil de mantener.
Solución: Se ha subdividido en 3 métodos que figuran en la clase CircusTrainGame: toPay, toFire y refreshToFire, éste último abstracto puesto que es el único que varía según la modalidad del juego.
Motivación: Disminución del acoplamiento, aumento de la cohesión, aumento de protección contra el cambio y mayor mantenibilidad.
Cuestiones sin resolver:
Alternativas consideradas:

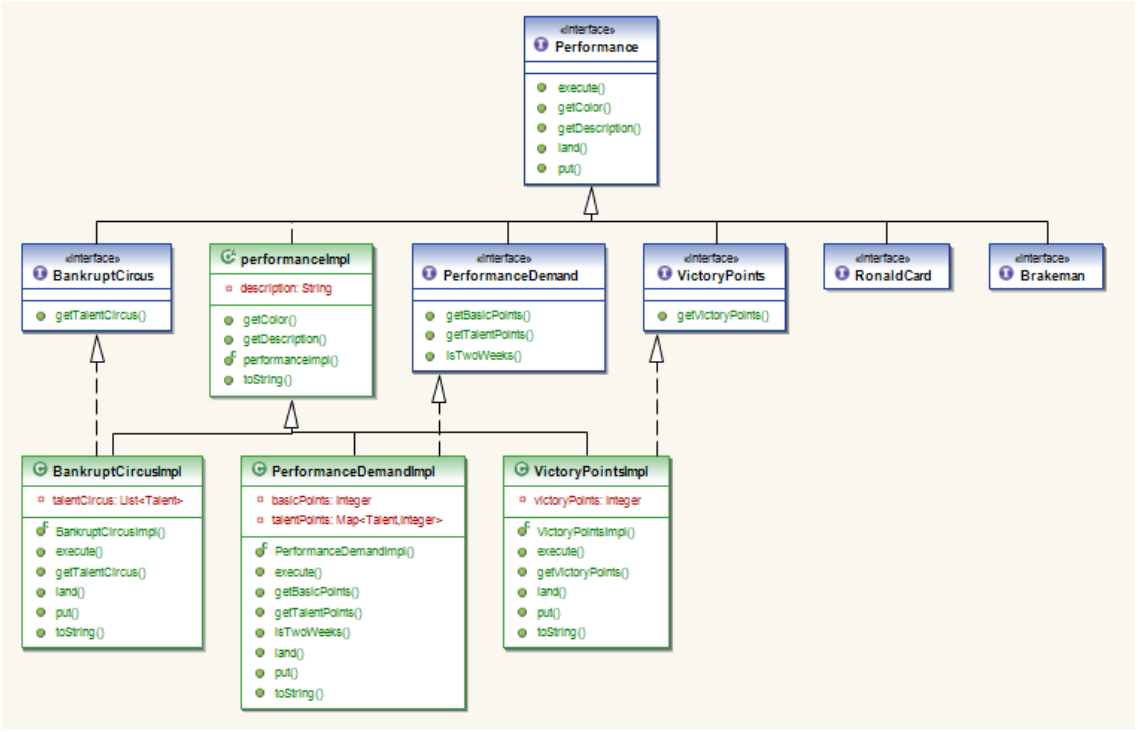
Eliminación de objetos clase Command:

INFORME TÉCNICO Nº A6
Asunto: Eliminando objetos de la clase Command
Resumen de la solución: Crear métodos de la clase circusGame y sus hijas
Factores causantes: En el desarrollo del juego se había creado una clase Command que se encargaba de ejecutar las acciones del jugo. Estas clases disminuían la coexion
Solución: Se han sustituido por métodos dentro de la clase circusTrainGame o, si solo se iba a usar en los modos de juego, en alguna de sus hijas.
Motivación: Aumentar la coexión del juego
Cuestiones sin resolver:
Alternativas consideradas: Dejarlo como estaba, ya que funcionaba.

Creación de put() y land() en las performance:

INFORME TÉCNICO Nº A7
Asunto: Creación de los métodos put() y land() en las performances
Resumen de la solución: Implementación de métodos put() y land() en las performances que permiten realizar acciones cuando el jugador cae en una ciudad o cuando la performance se asigna a una ciudad.
Factores causantes: Cuando un jugador cae en una ciudad con puntos de victoria había que sumarle esos puntos. Cuando se sacaba de la bolsa de performances un circo en bancarrota, había que sacar los talentos de la bolsa de talentos, para lo cual había que hacer una comparación al sacarlos de la bolsa.
Solución: Se ha implementado el método land() en las performances, que se ejecutara siempre que el jugador caiga en una ciudad con performance. Si no hay que hacer nada, no se hace. Se ha implementado un método put, que se ejecuta siempre que se añade una performance al tablero.
Motivación: Implentar partes del juego que estaban mal codificadas o confusas. Hacia falta gastar una carta de acción para recoger los puntos de victoria de una ciudad, cuando las reglas dicen que tiene que ser al llegar, sin gastar nada.
Cuestiones sin resolver: ¿Está bien crear métodos en algunas clases que no hacen nada?
Alternativas consideradas: No se han considerado otras alternativas.

Diagrama de Performance



Reestructuración global de árbol de clases:

INFORME TÉCNICO Nº T-001
Asunto: Reestructuración del árbol de clases
Resumen de la solución: Refactorización a nivel global de clases e interfaces.
Factores causantes: Gran vulnerabilidad ante el cambio. Patrón de diseño comandos mal aplicado. Casi nula opción de reutilizar código.
Solución: Cambiadas algunas clases a abstractas para tener alto grado de polimorfismo. Cambio de la estructura de herencia en los distintos modos de juego. Eliminación de algunas clases causadas por la mala aplicación del patrón comandos y su transformación en métodos.
Motivación: Proteger el software ante el cambio y tener una estructura de datos mucho más clara para facilitar su mantenimiento.
Cuestiones sin resolver: Código duplicado en algunos métodos
Alternativas consideradas: Patrón de diseño decorador o factorías.

5.4 Iteración 6

Añadidos en iteración 6

- Implementados los modos de juego.
- Solucionados bugs en el funcionamiento del juego.
- Añadidos documentos de asignación de responsabilidades (a continuación).
- Añadidos memorandos técnicos (a continuación).

Método execute BankruptCircusImpl:

Identificador	Descripción de la acción de alto nivel
E-004	Metodo para ejecutar la parte del circo en bancarrota

Pasos (usar pseudocódigo o similar)

1. Inicializar la lista newTalents.
2. Para cada talento del circo en bancarrota :
3. Mostrar si se quiere contratar y capturar la respuesta del usuario
4. Si el jugador selecciona SI:
5. se lanza un dado, y si es el valor adecuado, se guarda el talento en la lista newTalents
6. Si no es adecuado el valor:
7. El jugador selecciona si quiere pagarlo, en caso afirmativo se restara del su dinero
8. Se le añade al jugador los talentos de la lista newTalents.
9. Se borran los talentos del circo que están en la lista.
10. Si el circo se queda sin talentos, éste se borra del tablero.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	CollectionsFactory	[List] createListFactory( ).createList()		NO
2	TalentImpl	[String] toString( )		NO
3	BankruptCircusImpl	[List<Talent>] getTalentCircus()		NO
4	GameFactory	[Integer] throwDice( )		NO
5	PlayerImpl	[void] addMoney (int)		NO
6	PlayerImpl	[Integer] getMoney( )		NO
8	PlayerImpl	[void] addTalent (talentList:List<Talent>)		NO

Método de alto nivel

[void] execute()

Método execute PerformanceDemandImpl:

Identificador	Descripción de la acción de alto nivel
E-005	Metodo para ejecutar la parte del circo en bancarrota

Pasos (usar pseudocódigo o similar)

1. Referenciar las semanas necesarias para actuar del jugador en cuestión.
2. Si la actuación no es de 2 semanas o las semanas necesarias para actuar son 0, pasos 3,4,5,6,7,8,9,10,11,12,13,14. Si no, paso 15
3. Inicializamos el contador de puntos de actuación con los puntos de actuación máxima del jugador y el contador de nuevos puntos de actuación a 0.
4. Se recorren los talentos del jugador y el conjunto de puntos asociados a los talentos de la Performance.
5. Si coinciden incrementará el contador de nuevos puntos de actuación con los puntos correspondientes al talento.
6. Se incrementa el contador de nuevos puntos de actuación con los puntos de actuación básicos.
7. Añadir la Performance a la lista de actuaciones realizadas por el jugador.
8. Recorrer las actuaciones e ir sumándole al contador de nuevos puntos de victoria los puntos básicos de cada Performance.
9. Si el contador de nuevos puntos de victoria es mayor que el contador de puntos de actuación del jugador, paso 10.
10. Si el juego tiene más de 1 jugador, se incrementa el dinero del jugador en 10 y se actualiza el contador de puntos de actuación con el contador de nuevos puntos de actuación.
11. Añadir dinero al jugador según el estado del juego.
12. Poner a 1 las semanas que tienen que pasar para poder puntuar en una actuación.
13. Añadir la Performance a la lista de Performances usadas del jugador.
14. Borrar la actuación del tablero.
15. Decrementar el contador de semanas para actuar del jugador en una unidad.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	PlayerImpl	[Integer] getWeeksToPerformance( )		NO
11	GameState	[void] addMoney (player:Player)		NO
12	PlayerImpl	[void] setWeeksToPerformance (weeks:Integer)		NO
14	City	[void] removePerformance (performance:Performance)		NO
15	PlayerImpl	[void] setWeeksToPerformance (weeks:Integer)		NO

Método de alto nivel

[void] execute()

Diagrama de Colaboración (Opcional)

Método finalMonth AdvancedTwoPlayersImpl:

Identificador	Descripción de la acción de alto nivel
T-019	Método que realiza las acciones correspondientes al cambio de mes.
Pasos (usar pseudocódigo o similar)	
1. Se comparan ambos jugadores y se añaden los puntos de victoria pertinentes	
Diagrama de estados (Opcional)	

Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU
1	TwoPlayers Game	[void] comparePlayersAndAddVicotryPoints( )		NO
Método de alto nivel				
[void] finalMonth()				
Diagrama de Colaboración (Opcional)				

Método comparePlayersAndAddVictoryPoints en TwoPlayersGame:

Identificador	Descripción de la acción de alto nivel
T-020	Método que compara a ambos jugadores y modifica los puntos de victoria
Pasos (usar pseudocódigo o similar)	
1. Se compara el número de talentos de los jugadores y se suman los puntos de victoria pertinentes 2. Se comparan los puntos de actuación máxima de los jugadores y se añaden 4 puntos de victoria al que tenga mayor puntuación.	
Diagrama de estados (Opcional)	

Métodos de bajo nivel necesarios				
Paso	Clase	Método	Mem. Técn.	IU
1	TwoPlayers Game	[void] compareTalentsCountAndAddVicotryPoints( )		NO
Método de alto nivel				
[void] comparePlayersAndAddVictoryPoints( )				
Diagrama de Colaboración (Opcional)				

Método finaWage TwoPlayersGame:

Identificador	Descripción de la acción de alto nivel
T-021	Método comprueba si está la carta de acción de salarios no está descartada y hace pagar a todos los talentos el doble de su salario

Pasos (usar pseudocódigo o similar)

1. Se recorre la lista de jugadores.
2. Se crea una carta de acción de salarios pasándole el jugador tratado.
3. Se recorre la lista de cartas no descartadas de cada jugador.
4. Si existe la carta de salarios en la lista de cartas no descartadas, pasos 5 y 6.
5. Se invoca al método en el que el jugador paga a los talentos el doble de sus salarios.
6. Si el jugador tiene 3 o más puntos de victoria perderá 3.
7. Si el jugador tiene menos de 3 puntos de victoria se le pondrán a 0.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
2	WagesImpl	[WagesImpl] WagesImpl (game:CircusTrainGame, player:Player)		NO
5	CircusTrain Game	[void] toPay (player:Player, multiplicator:Integer)		NO
6	PlayerImpl	[void] addVictoryPoints (victoryPoints:Integer)		NO
7	PlayerImpl	[void] addVictoryPoints (victoryPoints:Integer)		NO

Método de alto nivel

[void] finalWage( )

Diagrama de Colaboración (Opcional)

Método selectCase en AdvancedTwoPlayersGame:

Identificador	Descripción de la acción de alto nivel
T-022	Pregunta al jugador la acción que quiere realizar según su estado

Pasos (usar pseudocódigo o similar)

1. Inicializa las variables de tipo String con las preguntas y posibles respuestas
2. Si el mes actual no es agosto o septiembre ejecutar el paso 6.
3. Si el jugador tiene cartas de acción descartadas y puede perder 2 puntos de reputación se le mostraran todas las opciones. Si no ejecutar el paso 4
4. Si un jugador solo puede perder 1 punto de reputación solo podrá jugar una carta de acción no descartada o coger dinero del banco. Si no ejecutar paso 5
5. Se le muestra al jugador solamente la opción de jugar una carta de acción no descartada
6. Si el jugador puede perder 4 puntos de victoria se le dará, además de la opción básica, la opción de jugar una carta de acción descartada. Si no ejecutar paso 7.
7. Se le muestra al jugador solamente la opción de jugar una carta de acción no descartada
8. Según la opción elegida se modifica el atributo de siguiente acción a realizar

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
3	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
4	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
5	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
6	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
7	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
8	CircusTrainGame	[void] setFollowingAction(action:String)		NO

Método de alto nivel

[void] selectCase (player:Player)

Diagrama de Colaboración (Opcional)

Método toPay:

Identificador	Descripción de la acción de alto nivel
T-024	Si el jugador tiene talentos, decide si pagarles o despedirles

Pasos (usar pseudocódigo o similar)

1. Referencia a los talentos del jugador mediante un mapa
2. Inicializa el dinero total a pagar a 0 y la pregunta con sus restricciones.
3. Recorre los talentos del jugador y calcula el dinero necesario para pagar a todos los talentos de una sola vez (dineroTotal=dineroTotal+salarioDelTalento*multiplicadorMonetarioBasico* multiplicador*cantidadDeTalentos).
4. Comprueba si el jugador tiene dinero suficiente para pagar a todos a la vez.
5. Si el dinero es suficiente, se le preguntará si pagará a todos. En caso negativo saltar al paso 8.
6. En caso afirmativo se le resta el dinero.
7. En caso negativo se invoca al método para despedir
8. Se llama recursivamente el método toPay a sí mismo.
9. Se invoca al método para despedir.
10. Se llama recursivamente el método toPay a sí mismo.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
6	Player	[void] addMoney (money : Integer)	T-004	NO
7	CircusTrain Game	[void] toFire (player:Player)	T-004	NO
8	CircusTrain Game	[void] toPay (player:Player, multiplicator:Integer)	T-004	NO
9	CircusTrain Game	[void] toFire (player:Player)	T-004	NO
10	CircusTrain Game	[void] toPay (player:Player, multiplicator:Integer)	T-004	NO

Método de alto nivel

[void] toPay (player:Player, multiplicator:Integer)

Diagrama de Colaboración (Opcional)

Método toFire:

Identificador	Descripción de la acción de alto nivel
T-025	Despide a los talentos del jugador pasado como parámetro

Pasos (usar pseudocódigo o similar)

1. Referencia a los talentos del jugador mediante un mapa.
2. Inicializa una lista vacía para meter los talentos.
3. Inicializa la pregunta, las restricciones y el índice selector de respuesta.
4. Recorre el conjunto de talentos del jugador añadiendo las restricciones, actualizando la pregunta y añadiendo a la lista los talentos.
5. Se pregunta qué talento va a ser despedido.
6. Se selecciona el talento de la lista y se descarta de los talentos del jugador.
7. Se llama al método refreshToFire para realizar las acciones derivadas de despedir talentos.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
5	readDataFromKeyBoard	[String] takeParametersToStringRestricted (message:String, conditions:String)		SI
6	PlayerImpl	[void] discardTalent (talent:Talent)		NO
7	CircusTrainGame	[void] refreshToFire (player:Player)	T-004	NO

Método de alto nivel

[void] toFire (player:Player)

Diagrama de Colaboración (Opcional)

Método refreshToFire AdvancedTwoPlayersGame

Identificador	Descripción de la acción de alto nivel
T-026	Lleva a cabo las acciones derivadas de despedir a los talentos.

Pasos (usar pseudocódigo o similar)

1. Si el mes es agosto o septiembre, pasos 2 y 3. En caso negativo, paso 4
2. Si el jugador tiene menos de 3 puntos de victoria, se le pondrán a 0.
3. Si el jugador tiene 3 o más puntos de victoria se le restarán 3 puntos.
4. Si el jugador no está en el mínimo de reputación, bajará un nivel.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	GameState	[String] getMonth( )		NO
2	PlayerImpl	[void] addVictorypoints (victoryPoints:Integer)		NO
3	PlayerImpl	[void] addVictorypoints (victoryPoints:Integer)		NO
4	PlayerImpl	[void] addReputation (reputation:Integer)		NO

Método de alto nivel

[void] refreshToFire (player:Player)

Diagrama de Colaboración (Opcional)

Método refreshToFire AdvancedSingleGame

Identificador	Descripción de la acción de alto nivel
T-027	Lleva a cabo las acciones derivadas de despedir a los talentos.

Pasos (usar pseudocódigo o similar)

1. Si el jugador no está en el mínimo de reputación, bajará un nivel.

Diagrama de estados (Opcional)

Métodos de bajo nivel necesarios

Paso	Clase	Método	Mem. Técn.	IU
1	PlayerImpl	[void] addReputation (reputation:Integer)		NO

Método de alto nivel

[void] canadianSelector( )

Diagrama de Colaboración (Opcional)

Creación gameState:

INFORME TÉCNICO Nº A8
Asunto: Creación de la clase GameState
Resumen de la solución: Creada clase GameState, usando el patrón State, que se encargará de gestionar el tiempo
Factores causantes: Durante todo el juego es necesario realizar acciones que dependen o bien del mes en el que nos encontramos. Durante todo el juego había situaciones en las que se comprobaba el mes en el que estamos.
Solución: Creada la clase GameState, la cual tiene tres implementaciones: greenState, orangeState y redState. Dentro de se implementan el método incrementTime, que hace pasar las semanas, y un selector de Performances, que lo sacará de la bolsa adecuada (verde, naranja o roja). También se implementan las penalizaciones a diversas acciones realizadas por los jugadores, que dependiendo de la "época" del juego, son unas u otras.
Motivación: Evitar encadenamientos if-elseif
Cuestiones sin resolver: No hay
Alternativas consideradas: No se han considerado

6.URL de implementación

http://code.google.com/p/circustrain/

7.Pruebas unitarias

- Pruebas de verificación (jugar a los diversos modos de juego para comprobar que la mecánica es correcta)
- Board (TestBoardJUnit)
- PerformanceBag (TestPerformanceJUnit)
- TwoPlayersGame (TwoPlayersGameJUnit)
- GameState (TestStateJUnit)
- ActionCards (TestActionCard)

8.Extensiones

Ninguna incluida.

9.Documento post-mortem

Qué ha ido mal:

- Dificultad para la organización y coordinación debido a los horarios de disponibilidad de los miembros del grupo.
- Dificultad para documentar debido a la inexperiencia.
- Disparidad en el código en los momentos iniciales.

Qué ha ido bien:

- Refactorizaciones hechas en el código inicial que lo han simplificado, aumentando la legibilidad.
- Gran uso de la herencia y del polimorfismo.

Qué se puede incluir/mejorar:

- Creación de una clase que contendrá la información a mostrar al usuario.
- Interfaz gráfica de usuario.
- Refactorización para eliminar la clase PerformanceBagImpl e incluirla en la clase GameState.
- Implementación de las reglas opcionales: cartas de evento, talentos especiales, modo fácil…
- Posibilidad de guardar partida.