Océane Jean-François / Chloé Bardot : S2C

Rapport IHM

SAÉ S2.01

01/04/2024

Sommaire

1. Analyse d'algorithme : validation de condition de construction

1.1. Dossier 17 :

La méthode PlayCard, prenant en paramètre le numéro du joueur et le numéro de la carte qu'il a choisi de déposer, fait appel à la surcharge de cette même méthode qui, elle, prend en paramètre non pas le numéro mais la carte elle-même. Si la carte n'existe pas, elle retourne un message d'erreur. Sinon, si le genre de la carte est un bâtiment, alors on enregistre dans une variable (reqBs) les types de cartes requis et leur quantité pour pouvoir poser la carte passée en paramètre. S'il y a au moins une carte requise, on compte le nombre de cartes de ce type qui est sur le plateau pour le joueur actif. Si ce nombre est inférieur au nombre requis pour poser la carte (la valeur de reqBs), alors on affiche un message d'erreur informant qu'on ne peut pas poser la carte. Sinon, on pose la carte. On retourne null car on ne veut pas afficher de message et true pour indiquer que le bâtiment a été créé.

Collections et tableaux associatifs :

- Buildings : Il s'agit de la liste des bâtiments qui ont été posé par le joueur actif.

- RequiredBuildings : Permet de savoir le nombre de cartes d'un certain type qu'il faut pour pouvoir poser une autre carte.

2. Conception d'algorithmes

2.1. Dossier 18 :

Dans le switch case Kind.Building de la méthode PlayCard, qui se trouve dans la classe Game, nous créons une nouvelle variable que nous nommons reqRs, qui récupère le nombre de ressources nécessaires pour créer la carte. Ensuite, comme vu précédemment, nous appliquons une condition : s'il y a au moins une ressource nécessaire, nous comptons le nombre de ressources requises pour poser la carte passée en paramètre et présentes dans la main du joueur. Si ce nombre est inférieur au nombre requis pour poser la carte, nous retournons un message d'erreur. Sinon, nous parcourons la liste des types de ressources nécessaires pour poser la carte, puis nous parcourons les cartes qui sont dans la main du joueur actif. Pour chaque carte du même type que la ressource, nous la défaussons, puis nous incrémentons la variable « nbRs » précédemment déclarés. Cette variable a pour but de compter le nombre de ressources défaussées et de ne pas défausser de ressources en plus de ce qui est nécessaire. Une fois cela fait, nous jouons la carte.

2.2. Dossier 19 :

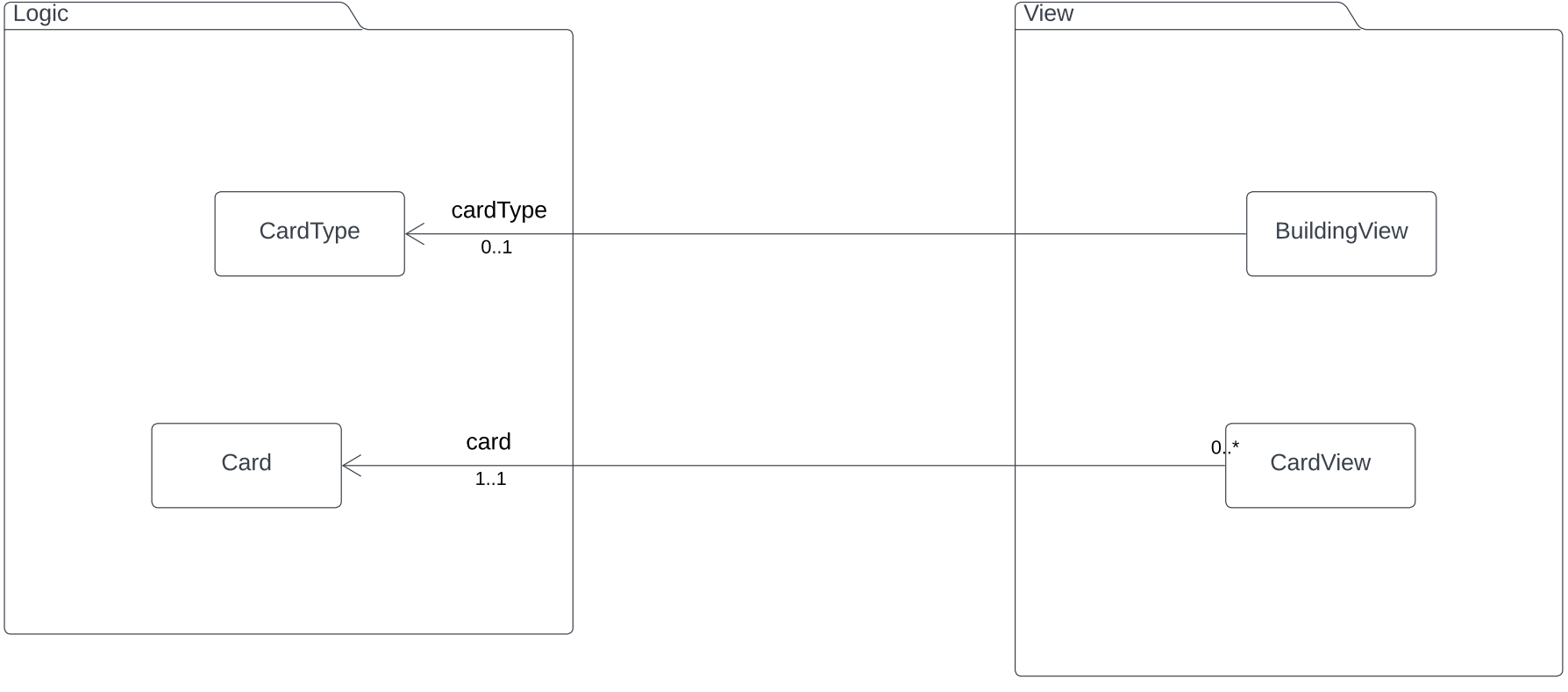
Nous pourrions créer une méthode qui prendra en paramètre la carte que le joueur souhaite poser et le joueur actif. Nous appellerons cette méthode dans PlayCard de la classe Game. À l'intérieur de cette méthode, nous pourrions effectuer un switch case en fonction du type d'effet de la carte. Si l'effet de la carte à poser est "canExchange", alors on récupère l'EffectCard de cette carte dans une variable afin de savoir quelles cartes nous pouvons échanger.

Il pourrait être utile d'ajouter un nouveau statut de jeu qui nous permettra de savoir quand le joueur pourra échanger (Exchanging) une carte ou non. Pour ce faire, nous ajouterions ce nouveau statut dans le switch case de la méthode Play() et nous afficherons un message dans la fenêtre de messagerie de l'interface. Afin d'utiliser ce nouveau statut de jeu, on pourrait ajouter un bouton dans l'interface de l'application qui aura pour but de changer le statut du jeu. Nous créerons une méthode pour l'événement clic qui modifiera la propriété GameStatus de la classe Game avec le nouveau statut, puis nous rappellerons la méthode Play() pour prendre en compte le changement de statut.

Ensuite, dans la méthode permettant de cliquer sur une carte dans la main du joueur, nous pourrions ajouter une condition permettant, si le statut du jeu correspond au nouveau statut et si la carte correspond au type enregistré dans la propriété EffectCard, de "défausser une carte" puis de tirer une carte depuis le deck.

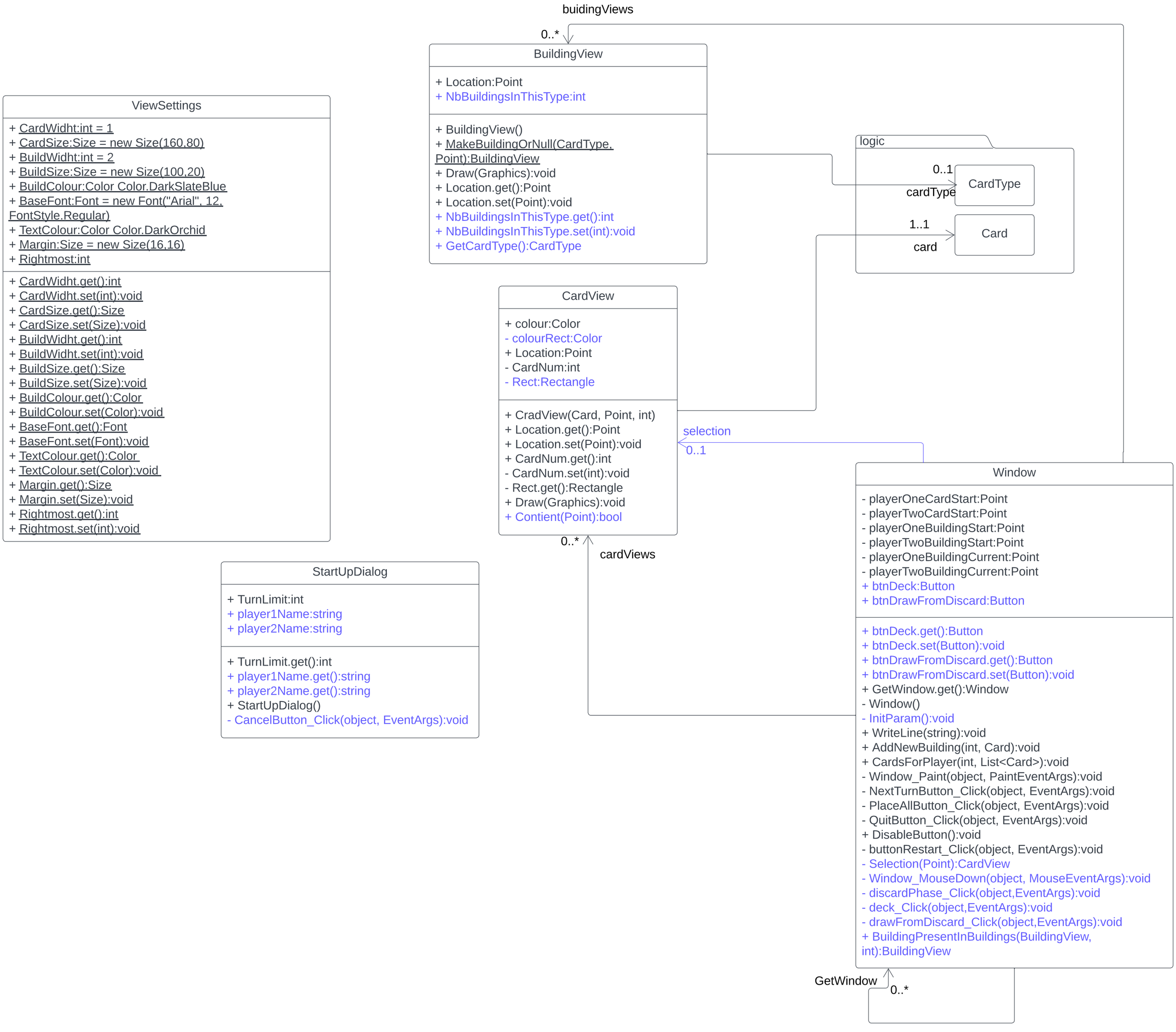
Pour que le joueur puisse continuer à jouer même après avoir échangé une carte, nous devrons passer du statut de jeu "Exchanging" au statut "Playing", c'est-à-dire mettre à jour la propriété GameStatuts. Pour ce faire, on pourrait créer une méthode dans la classe Game permettant de changer le statut du jeu de "Exchanging" à "Playing", puis appeler cette méthode juste après la condition définie précédemment, dans la même méthode permettant de cliquer sur les cartes dans la main du joueur.

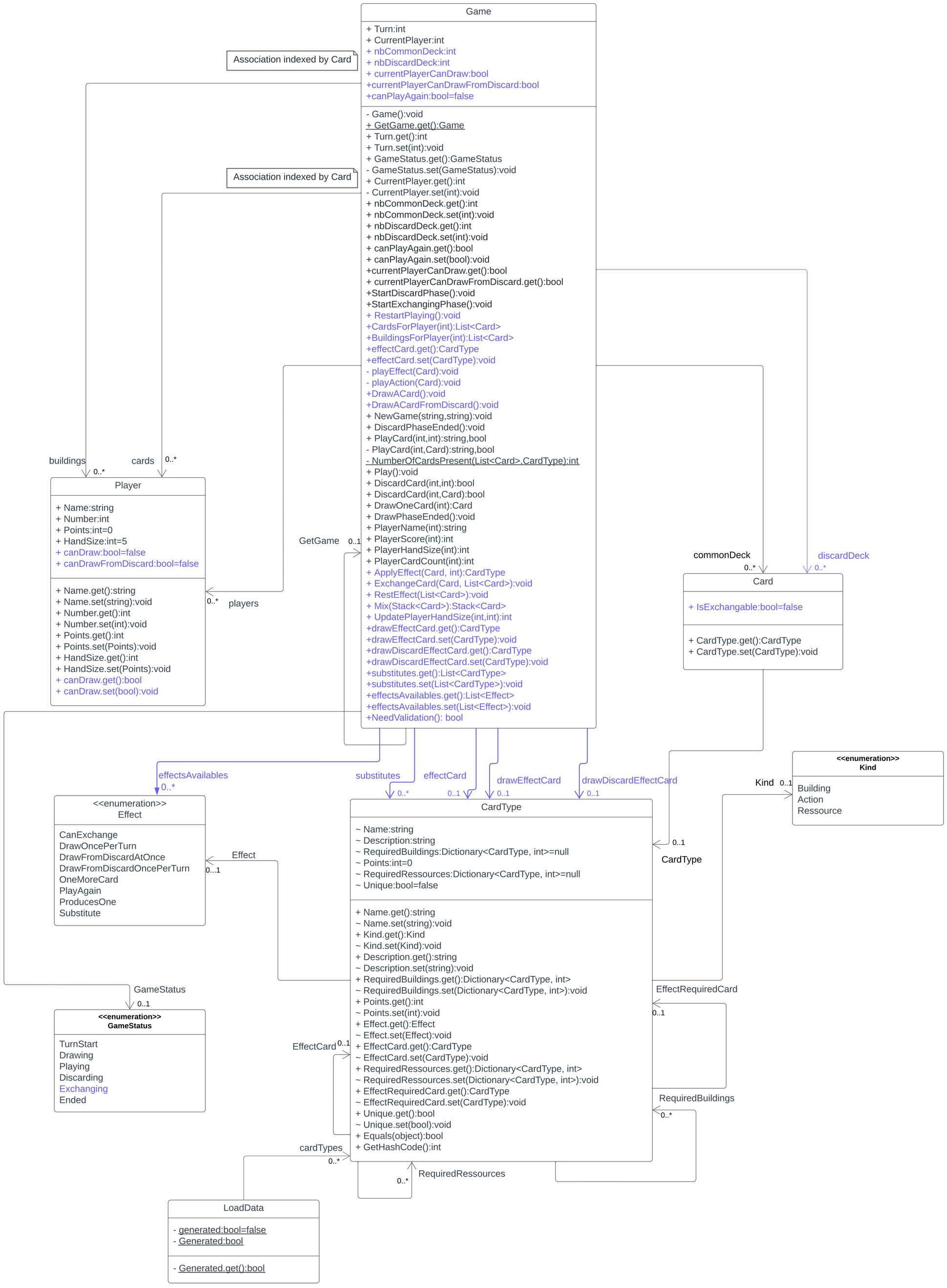
Cela permettra ainsi au joueur d'échanger une carte puis de rejouer directement la carte échangée s'il le souhaite. Cependant, si le joueur veut échanger plusieurs cartes, il devra pour chaque carte qu'il souhaite échanger, cliquer sur le bouton d'échange.


Logic
View
CardType
cardType
0..1
BuildingView
Card
card
1..1
0..*
CardView

● Propriétés et méthodes ajoutées

● Propriétés et méthodes existantes




Game
+ Turn:int
+ CurrentPlayer:int
+ nbCommonDeck:int
+ nbDiscardDeck:int
+ currentPlayerCanDraw:bool
+currentPlayerCanDrawFromDiscard:bool
+canPlayAgain:bool=false
- Game():void
+ GetGame.get():Game
+ Turn.get():int
+ Turn.set(int):void
+ GameStatus.get():GameStatus
- GameStatus.set(GameStatus):void
+ CurrentPlayer.get():int
- CurrentPlayer.set(int):void
+ nbCommonDeck.get():int
+ nbCommonDeck.set(int):void
+ nbDiscardDeck.get():int
+ nbDiscardDeck.set(int):void
+ canPlayAgain.get():bool
+ canPlayAgain.set(bool):void
+currentPlayerCanDraw.get():bool
+ currentPlayerCanDrawFromDiscard.get():bool
+StartDiscardPhase():void
+StartExchangingPhase():void
+ RestartPlaying():void
+CardsForPlayer(int):List<Card>
+BuildingsForPlayer(int):List<Card>
+effectCard.get():CardType
+effectCard.set(CardType):void
- playEffect(Card):void
- playAction(Card):void
+DrawACard():void
+DrawACardFromDiscard():void
+ NewGame(string,string):void
+ DiscardPhaseEnded():void
+ PlayCard(int,int):string,bool
- PlayCard(int,Card):string,bool
- NumberOfCardsPresent(List<Card>,CardType):int
+ Play():void
+ DiscardCard(int,int):bool
+ DiscardCard(int,Card):bool
+ DrawOneCard(int):Card
+ DrawPhaseEnded():void
+ PlayerName(int):string
+ PlayerScore(int):int
+ PlayerHandSize(int):int
+ PlayerCardCount(int):int
+ ApplyEffect(Card, int):CardType
+ ExchangeCard(Card, List<Card>):void
+ RestEffect(List<Card>):void
+ Mix(Stack<Card>):Stack<Card>
+ UpdatePlayerHandSize(int,int):int
+drawEffectCard.get():CardType
+drawEffectCard.set(CardType):void
+drawDiscardEffectCard.get():CardType
+drawDiscardEffectCard.set(CardType):void
+substitutes.get():List<CardType>
+substitutes.set(List<CardType>):void
+effectsAvailables.get():List<Effect>
+effectsAvailables.set(List<Effect>):void
+NeedValidation(): bool
Association indexed by Card
Association indexed by Card
buildings
0..*
cards
0..*
Player
+ Name:string
+ Number:int
+ Points:int=0
+ HandSize:int=5
+ canDraw:bool=false
+ canDrawFromDiscard:bool=false
+ Name.get():string
+ Name.set(string):void
+ Number.get():int
+ Number.set(int):void
+ Points.get():int
+ Points.set(Points):void
+ HandSize.get():int
+ HandSize.set(Points):void
+ canDraw.get():bool
+ canDraw.set(bool):void
0..*
players
GetGame
0..1
commonDeck
0..*
discardDeck
0..*
Card
+ IsExchangable:bool=false
+ CardType.get():CardType
+ CardType.set(CardType):void
effectsAvailables
0..*
substitutes
0..*
effectCard
0..1
drawEffectCard
0..1
drawDiscardEffectCard
0..1
<<enumeration>>
Effect
CanExchange
DrawOncePerTurn
DrawFromDiscardAtOnce
DrawFromDiscardOncePerTurn
OneMoreCard
PlayAgain
ProducesOne
Substitute
Effect
0...1
CardType
~ Name:string
~ Description:string
~ RequiredBuildings:Dictionary<CardType, int>=null
~ Points:int=0
~ RequiredRessources:Dictionary<CardType, int>=null
~ Unique:bool=false
+ Name.get():string
~ Name.set(string):void
+ Kind.get():Kind
~ Kind.set(Kind):void
+ Description.get():string
~ Description.set(string):void
+ RequiredBuildings.get():Dictionary<CardType, int>
~ RequiredBuildings.set(Dictionary<CardType, int>):void
+ Points.get():int
~ Points.set(int):void
+ Effect.get():Effect
~ Effect.set(Effect):void
+ EffectCard.get():CardType
~ EffectCard.set(CardType):void
+ RequiredRessources.get():Dictionary<CardType, int>
~ RequiredRessources.set(Dictionary<CardType, int>):void
+ EffectRequiredCard.get():CardType
~ EffectRequiredCard.set(CardType):void
+ Unique.get():bool
~ Unique.set(bool):void
+ Equals(object):bool
+ GetHashCode():int
0..1
CardType
Kind
0..1
<<enumeration>>
Kind
Building
Action
Ressource
GameStatus
0..1
<<enumeration>>
GameStatus
TurnStart
Drawing
Playing
Discarding
Exchanging
Ended
EffectCard
0..1
cardTypes
0..*
RequiredRessources
0..*
EffectRequiredCard
0..1
RequiredBuildings
0..*
LoadData
- generated:bool=false
- Generated:bool
- Generated.get():bool