Model

Classes

Cards

<<enumeration>>
KingdomResource

ANIMAL
PLANT
FUNGI
INSECT

<<enumeration>>
Resource

FEATHER
POTION
SCROLL

implements

null if there is no Corner(where Cards can't be placed on top of)

Indicates if this Corner is covered by another Card in the playAreas.
Default: false
When a Card is added to the playAreas, it covers the nearby Cards' Corners

<<record>>
CardSide

- topLeftCorner: Corner
- topRightCorner: Corner
- bottomLeftCorner: Corner
- bottomRightCorner: Corner

Corner

- isCovered: boolean
- item: Item

item = null is the Empty Corner (where Cards can be placed on top of)

Coordinates

- x: int
- y: int

implements

true if the Card is showing its front side, false otherwise

Contain the urls of the Image resources

<<abstract>>
Card

[final] - listeners: ArrayList<Listener>
isFrontFacing: boolean
- id: int
- frontImage: String
- backImage: String
+ getShowingImage(): String
+ flip(): void

<<enumeration>>
CardType

RESOURCECARD
GOLDCARD
STARTERCARD
OBJECTIVECARD

[final] - coordinates

2 - frontSide, backSide

Extends

x and y are the card's coordinates, assigned when the card is placed. Default: 0,0

<<abstract>>
PlayableCard

[final] - permanentResources: ArrayList<Item>
[final] - kingdom: KingdomResource
- earnedPoints: int
+ getShowingSide(): CardSide
[abstract] + canBePlaced(ArrayList<PlayableCard>): boolean

canBePlaced() always returns true in StarterCard and ResourceCard

Extends

ResourceCard

@Override
+ canBePlaced(ArrayList<PlayableCard>): boolean

GoldCard

[final] - costs: HashMap<Item, Integer>
+ initMap(): void
+ setCost(Item, int): void
+ getCost(Item): int
- increaseValueHashMap<Item, Integer>, Item): void
@Override
+ canBePlaced(ArrayList<PlayableCard>): boolean

StarterCard

@Override
+ canBePlaced(ArrayList<PlayableCard>): boolean

ObjectiveCard

- objective

Creates all the <String, Integer> values for the KingdomResources inside the HashMap (with 0 as Value)

if the GoldCard is showing its front side, it checks if the cost requirement are met. If it's showing its back side it always returns true

[final] - objective

Objective is satisfy to obtain the card points. The Count condition for placement is seen as a CountObjective with number = 1.

A String containing the Objective's name and description, used in the GUI

<<abstract>>
Objective

- points: int
- name: String
- description: String
[abstract] # check(ArrayList<PlayableCard>): int
+ calculatePoints(ArrayList<PlayableCard>): int

A copy of Player's PlayField is passed. This allows to make a single implementation for every type of Objective. Placement included

The amount of things it needs to count before considering the Objective as satisfied once.

Extends

<<abstract>>
CountObjective

- number: int

<<abstract>>
PlacementObjective

primaryType: KingdomResource

Counts the number of Corners the Card is covering (only for GoldCard)

CornerCountObjective

- coordinates: Coordinates
@Override
check(ArrayList<PlayableCard>): int

Concrete Method implementation of the Abstract Method from Objective

Returns the number of times the Objective has been satisfied.
Ex: 3 covered Corners => 3

ItemCountObjective

- item: Item
@Override
check(ArrayList<PlayableCard>): int

Counts the number of times a certain item is visible inside the PlayAreas

SetItemCountObjective

@Override
check(ArrayList<PlayableCard>): int

DiagonalPlacementObjective

- isLeftToRight: boolean
@Override
check(ArrayList<PlayableCard>): int

true: \
false: /

<<enum>>
LPlacementOffset

TOP_LEFT(0, 1, -1)
TOP_RIGHT(1, 0, -1)
BOTTOM_LEFT(-1, 0, 1)
BOTTOM_RIGHT(0, -1, 1)
[final] - xOffset: int
[final] - yOffset: int
[final] - inlineOffset: int

L_ShapedPlacementObjective

- secondaryType: KingdomResource
@Override
check(ArrayList<PlayableCard>): int
- countOverlap(ArrayList<ArrayList<PlayableCard>>)

positionCornerCard

The offset of the Card placed under or over the base Card. In this case x and y are always equal

The positional offset of the corner Card from the base Card

Creates and initializes a new Deck. Used like a Constructor.
Imports the real Cards from a JSON file, puts them inside the Deck, and shuffles the Deck before returning it.

Support Methods for initDeck()

<<interface>>
Deck

[final] - listeners: ArrayList<Listener>
[final] - cards: ArrayList<Card>
- cardType: CardType
+ initDeck(CardType): Deck
- getCorner(String): Corner
- getKingdom(String): Item
- getObjective(int, String, boolean, String, String): Objective
+ getNumberOfCards(): int
+ shuffle(): void
+ draw(): Card
+ getTopCard(): Card
+ getCopy(): ArrayList<Card>

"Peeking" methods: used to see the status of the Deck without making changes to it.
Used in GUI and TUI

0..102

- deck

PlayingDeck

- isIt1: Card
- isIt2: Card

A PlayingDeck is a group of elements common to all players.
It contains a Deck and two Cards from that same Deck. The Deck's Cards are used for the act of drawing them from that Deck, which can be grabbed by any player instead of drawing from the Deck.

Game

Starter Cards don't need a PlayingDeck, so they are stored in a Deck.

- starterDeck

Default: false
Becomes true once the corresponding Deck is empty

- resourcePlayingDeck
- goldPlayingDeck
- objectivePlayingDeck

Default: false
Becomes true once one of the players has reached 20 or more points.

Creates all the Decks and PlayingDecks, and calls initDeck() on them to import the Cards from the JSON file.

Fill the slot with a Card (after Player.grabCard())

<<interface>>
Game

- isResourceDeckEmpty: boolean
- isGoldDeckEmpty: boolean
- playerHasReachedTwentyPoints: boolean
- playerTurn: int
[final] - players: ArrayList<Player>
[final] - listeners: ArrayList<Listener>
+ initPlayingDecks(): void
+ startGame(): void
+ endGame(): void
- checkEndGame(): void
+ addPlayer(Player): void
+ kickPlayer(Player): void
+ getPlayer(int): Player
- getCurrentPlayer(): Player
+ setPlayerTurn(int): void
+ getNumberOfPlayers(): int
- getIndexOfPlayer(String): int
+ putDown(PlayingDeck, int): void
- setFirstPlayer(): void
+ countPoints(): ArrayList<HashMap<String, String>>

2..4

An integer to signal whose Turn the current one is.

[final] - chat

PlayField

[final] - playedCards: ArrayList<PlayableCard>
[final] - counter: HashMap<Item, Integer>
[final] - listeners: ArrayList<Listener>
+ getPlayedCards(): ArrayList<PlayableCards>
+ getNumberOf(Item): int
+ add(Item): void
+ remove(Item): void
+ addCardPlayableCard, int, int): void
- coverNearbyCorners(int, int): void
+ getAvailablePlacements(): ArrayList<Coordinates>
- removeDuplicatePlacements(ArrayList<Coordinates>): ArrayList<Coordinates>
- removeIllegalPlacements(ArrayList<Coordinates>): ArrayList<Coordinates>
- isThereACardin(int, int): boolean
+ getLastPlayedCard(): PlayableCard

Counters used in GUI/TUI

Returns a copy of playedCards, used to show them in GUI and for Objective's count

Returns all the Coordinates corresponding to a valid placement for a Card

Player

[final] - listeners: ArrayList<Listener>
- nickname: String
- isFirst: boolean
- token: Token
- points: int
- secretObjective: ObjectiveCard
[final] - hand: ArrayList<PlayableCard>
- tempObjective1: ObjectiveCard
- tempObjective2: ObjectiveCard
- temporaryStarterCard: StarterCard
- status: GameStatus
- isDisconnected: boolean
+ setStarterCard(): void
+ getTemporaryObjectiveCards(): ArrayList<ObjectiveCard>
+ drawStartingHand(PlayingDeck, PlayingDeck): void
+ drawSecretObjectives(PlayingDeck): void
+ drawTemporaryStarterCard(Deck): void
+ playCard(PlayableCard, int, int): void
+ drawCard(PlayingDeck): void
+ grabCard(PlayingDeck,int): void
+ getHandCard(int): PlayableCard
+ setHandCard(int, PlayableCard): void
+ getHandSize(): int

Used to temporarily store the Cards when the Player has to make some choices

Plays the temporaryStarterCard in (0, 0)

Draws from the Deck

Grabs the Card from the table. The int indicates which slot

Draws the first 3 Cards to put in hand

Puts the Card on the table at the coordinates (x, y)

<<enumeration>>
Token

RED
BLUE
GREEN
YELLOW

implements

Chat

[final] - listeners: ArrayList<Listener>
[final] - messages: ArrayList<ChatMessage>
+ sendMessage(ChatMessage): void
- addMessage(ChatMessage): void
+ getLastChatMessage(): ChatMessage
+ getChatMessage(int): ChatMessage
+ getChatSize(): int
+ getFullChat(): ArrayList<ChatMessage>

*

- sender

ChatMessage

- sender: String
- text: String
- time: LocalDateTime

Exceptions

Exception

IllegalActionException
NoSuchCardException
RemovingFromZeroException
IllegalPlacementException
PlacementConditionNotMetException

Extends

Interfaces

<<interface>>
Item

<<interface>>
StatusListener<>
<<interface>>
HandListener
<<interface>>
TokenListener
<<interface>>
CommonObjectivesListener
<<interface>>
CardPickerListener
<<interface>>
PlayAreaListener
<<interface>>
PointsListener
<<interface>>
LastTurnListener
<<interface>>
ReadyToChooseSecretObjectiveListener
<<interface>>
GoldNotifyListener
<<interface>>
GoldNotifyUpdateListener
<<interface>>
ResourceNotifyListener
<<interface>>
ResourceNotifyUpdateListener
<<interface>>
EmptyDeckListener
<<interface>>
SecretObjectiveListener
<<interface>>
PlayerListener

<<interface>>
Listener

onEvent(): void

work together

<<interface>>
Observable

setListener(Listener): void
removeListener(Listener): void
notifyListeners(): void