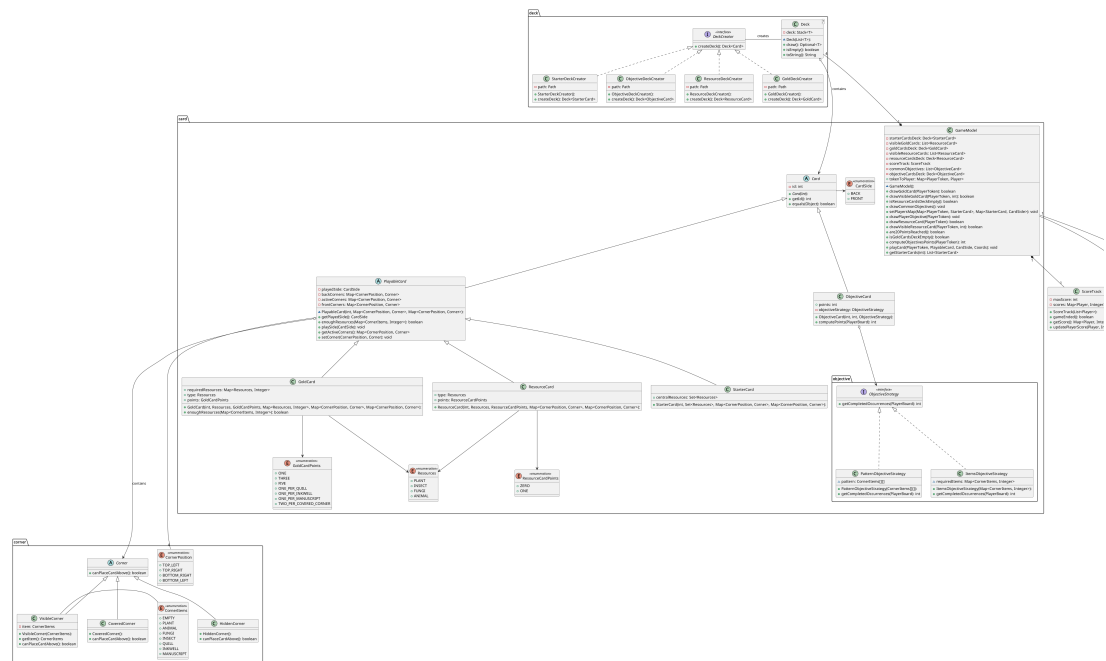# Peer review #1 - GC11 - Model

Samuele Pischedda, Angelo Prete, Gabriele Raveggi, Andrea Sanvito

26/03/2024

General Overview

In this document we describe the UML of our Codex Naturalis java implementation. In particular, we will explain how we implemented the model of the MVC architectural pattern.

This is the full UML represententig the model, to see it in detail see the file attached.
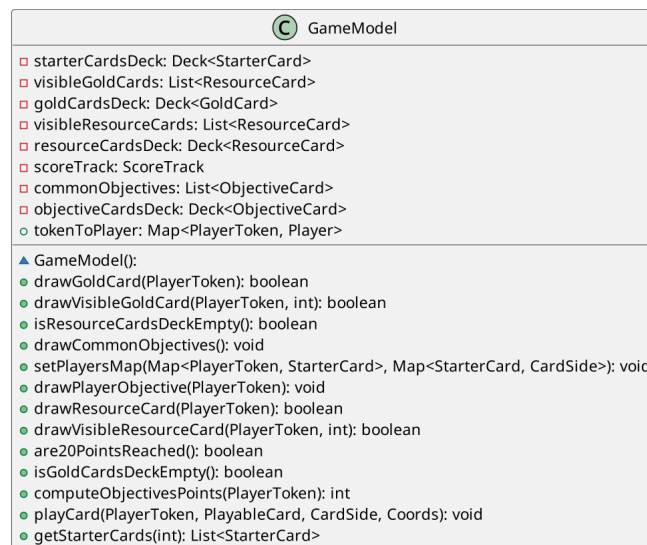
Components description

1   Class: `GameModel`

The `GameModel` class is the core component of a game. While it doesn't contain a finite state machine, which we will implement in the controller package and that will define the flow and behavior of a match, it holds the state of the match. In fact, it contains numerous informations about the state of the different game entities:

- a map defining the color picked by each player;
- all the decks found on the board (the gold, resource, objective and starter decks);
- the visible drawable cards and common objectives;
- the score board (`ScoreTrack`).

Moreover, it contains many methods that allow retrieving and modifying the state:

- functions to use during the setup phase, allowing players to draw their starter card and pick their own token;
- functions to draw and play cards;
- a function to check if twenty points have been reached by a player;
- functions to check if the decks are empty.

Ⓒ GameModel

- ▫ starterCardsDeck: Deck<StarterCard>
- ▫ visibleGoldCards: List<ResourceCard>
- ▫ goldCardsDeck: Deck<GoldCard>
- ▫ visibleResourceCards: List<ResourceCard>
- ▫ resourceCardsDeck: Deck<ResourceCard>
- ▫ scoreTrack: ScoreTrack
- ▫ commonObjectives: List<ObjectiveCard>
- ▫ objectiveCardsDeck: Deck<ObjectiveCard>
- ○ tokenToPlayer: Map<PlayerToken, Player>

- ▲ GameModel():
- ● drawGoldCard(PlayerToken): boolean
- ● drawVisibleGoldCard(PlayerToken, int): boolean
- ● isResourceCardsDeckEmpty(): boolean
- ● drawCommonObjectives(): void
- ● setPlayersMap(Map<PlayerToken, StarterCard>, Map<StarterCard, CardSide>): void
- ● drawPlayerObjective(PlayerToken): void
- ● drawResourceCard(PlayerToken): boolean
- ● drawVisibleResourceCard(PlayerToken, int): boolean
- ● are20PointsReached(): boolean
- ● isGoldCardsDeckEmpty(): boolean
- ● computeObjectivesPoints(PlayerToken): int
- ● playCard(PlayerToken, PlayableCard, CardSide, Coords): void
- ● getStarterCards(int): List<StarterCard>

2 Class: `ScoreTrack`

This class tracks the individual scores of players. In particular, it allows updating the score and checking if twenty points have been reached, through methods.
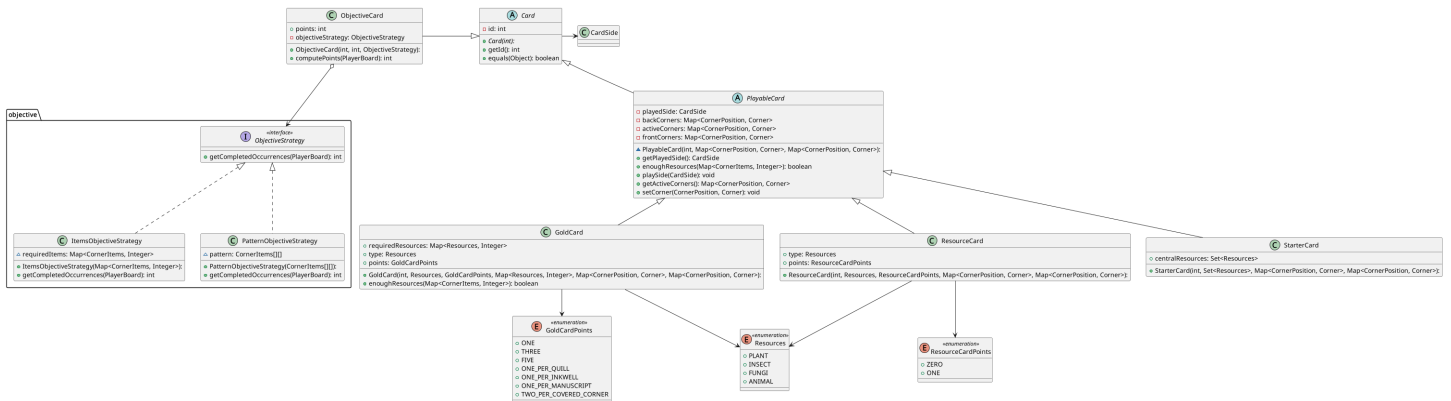
3   Package: `card`

All the informations about cards and their properties are saved in this package. Starting from the abstract class Card, two different subclasses are derived:

- `ObjectiveCard`, that comes with its own `objective` package, implements the strategy pattern used to easily define different ways of computing points for objective cards;

- `PlayableCard` (abstract), that contains maps that track card corners' properties (explained later in the card package). The `PlayableCard` class is divided into the following subclasses (each with their own properties):

  - `ResourceCard`;
  - `GoldCard`;
  - `StarterCard`.

This package also contains the enumerations needed to represent each propery:

- `CardSide`;

- `Resources`;

- `ResourceCardPonts`;

- `GoldCardPoints`.

4 Package: `deck`

The package contains a main generic `Deck` class, which implements a deck as a stack of a generic card type. It is used in the `GameModel` class, to represent the different decks used during the game. Furthermore, using the factory method pattern we load the decks into the `GameModel` from json files.

For every card type, we initialize its deck through different deck creators, each implementing a common `DeckCreator` interface:

- `GoldDeckCreator`

- `ObjectiveDeckCreator`

- `ResourceDeckCreator`

- `StarterDeckCreator`
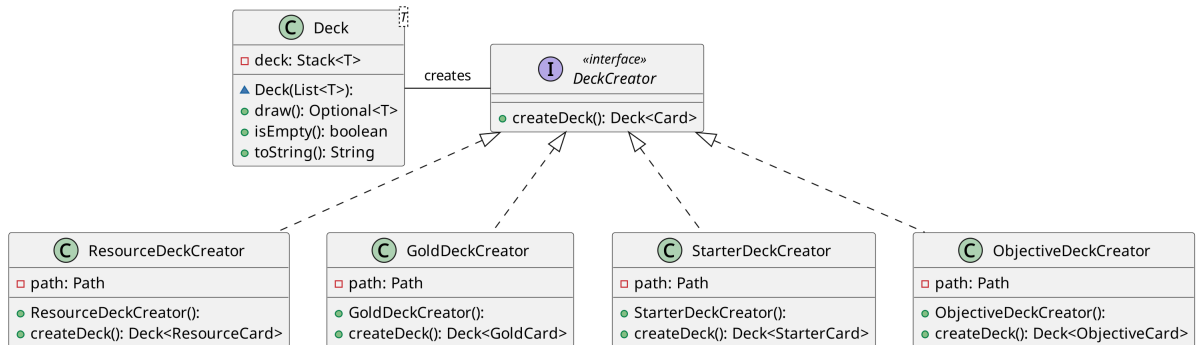
5 Package: `corner`

The package includes an abstract `Corner` class which represents a generic corner of a card. The main class is extended by the following classes:

- `CoveredCorner`: a corner of a card that has been covered by another card, thus it is not visible;

- `HiddenCorner`; a missing corner (hidden), where you can't place any card;

- `VisibleCorner`: a corner of a card that is either empty or contains an item.

To keep track of the position of a corner in a card, we use the `CornerPosition` enum. The `CornerItems` enum is used to represent an item in a visible corner.
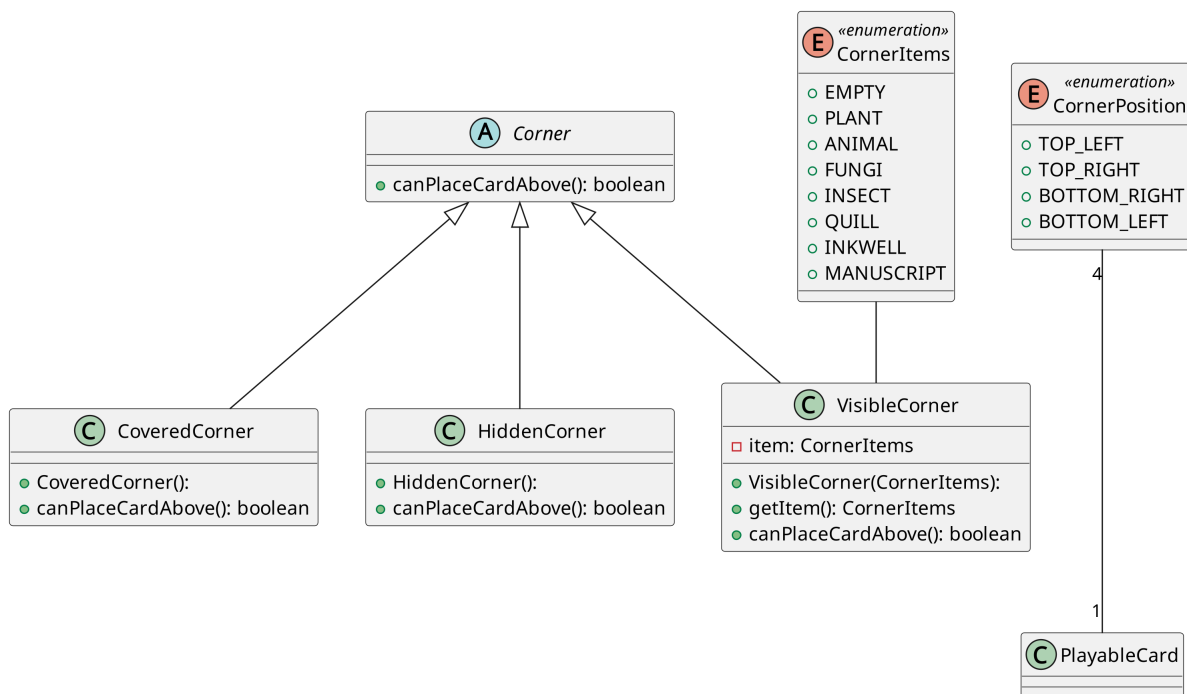
6   Package: `player`

The package contains a main `Player` class which represents a single player in the game. The player class contains different attributes to identify their game entities:

- a `PlayerBoard` object that models the board of the player;

- a `PlayerHand` object that represents the three cards in a player hand;

- the `ObjectiveCard` secret objetive.

More specifically, the other classes are:

- `PlayerBoard`: contains two maps, one to model the player's board, and one to keep track of the amount of visible items by item type;

- `PlayerHand`: implemented as a list of `PlayableCard`;

- `Coords`: represents 2D coordinates that can be used to identify cards' position in the player board.

The package also contains the PlayerToken enumeration, used to represent a single player in the model side of the project thanks to a mapping in the GameModel.