# Presentation document of UML diagram
# Peer review

Berardinelli, Genovese, Grandi, Haddou
Gruppo AM21

March 26, 2024

## 1 Model

### 1.1 Defining the model and the controller's roles

As a team, we made the choice to implement part of the game logic in our model because we wanted the controller layer in the server to be as light as possible. This way the role of the controller layer is to parse the inputs coming from the client, call the model methods to update the game, players, gameboard and playerboards statuses with the parsed data and finally to signal the views to update.

### 1.2 Notes on the UML diagram of the model

For development and accessibility purposes we split the model class diagram in two parts:

1. Card hierarchy
2. Rest of the model

We deliberately omitted some of the links between some classes (notably enums), so that the most meaningful connections would be easily visible.

## 1.3 Documenting design choices

Some design choices we took that we think are worth documenting a little bit in detail are:

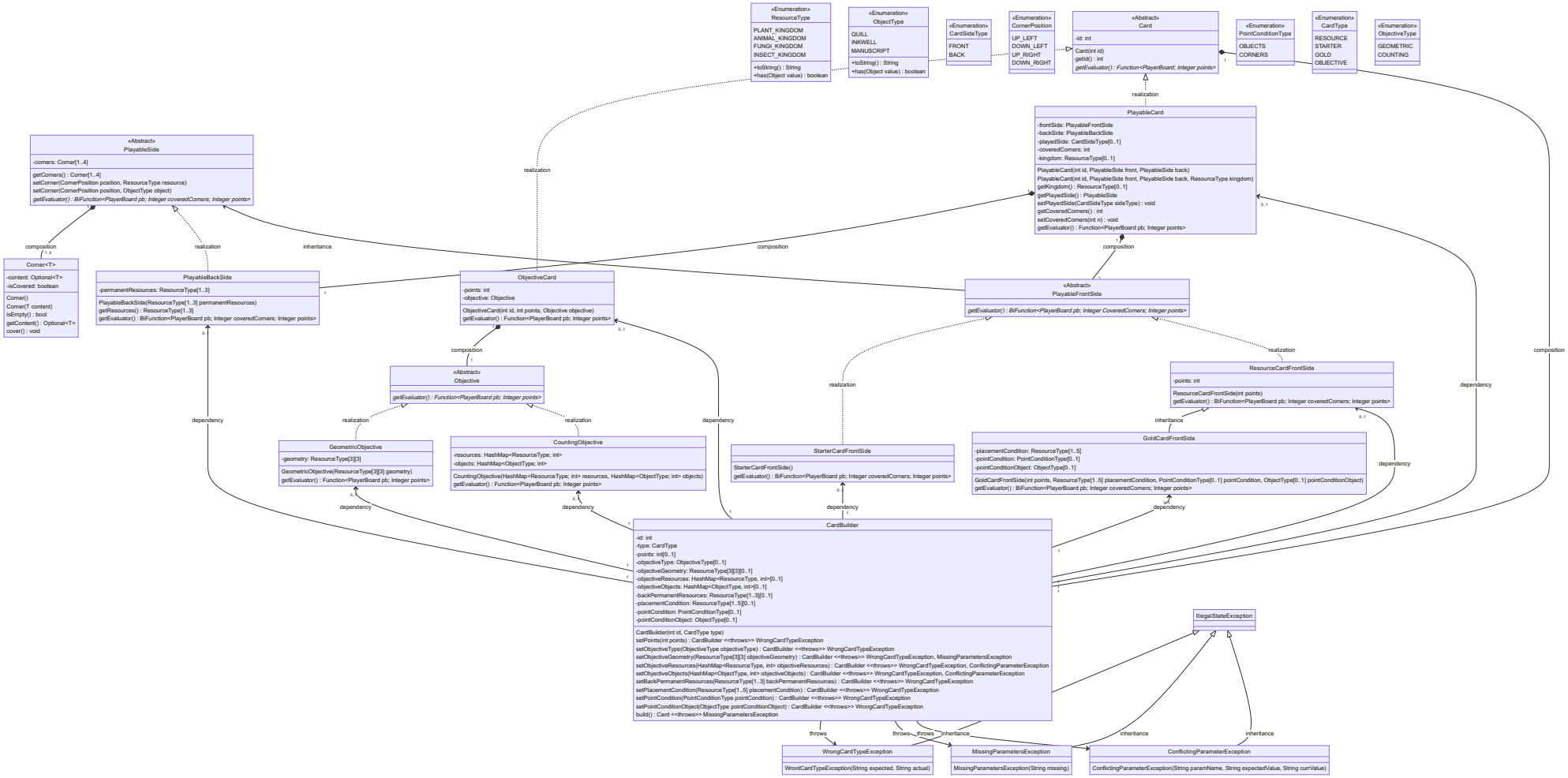
1. The hybrid approach to evaluating objectives and card placement points

   To reflect actual game dynamics we decided to make the cards return a Function which will be called with the `PlayerBoard` parameters in the `PlayerBoard` context. This way we avoid sending around the `PlayerBoard` instances, which we considered a bad practice, and avoid duplicating it just to have the cards evaluate the points with their specificity, which we obtain nonetheless with this approach.

2. The use of the builder pattern for the `Player` class

   Since the player attributes are all final once chosen by the client, we decided to store `PlayerBuilder` instances in a `HashMap` in the `Lobby` class, as the controller layer handles the parsing of the client inputs. This way a player is added to the Game's Player list only when finalized (once they have chosen their Secret Objective).

3. Some classes can be considered redundant as they could be easily implemented as part of the class they are a composition to. We decided to keep them separate to keep the code more readable and to make the implementation of the `GUI` easier as these entities are effectively functional on their own and can be considered "Drawable".


«Enumeration»
ResourceType
PLANT_KINGDOM
ANIMAL_KINGDOM
FUNGI_KINGDOM
INSECT_KINGDOM
+toString() : String
+has(Object value) : boolean
«Enumeration»
ObjectType
QUILL
INKWELL
MANUSCRIPT
+toString() : String
+has(Object value) : boolean
«Enumeration»
CardSideType
FRONT
BACK
«Enumeration»
CornerPosition
UP_LEFT
DOWN_LEFT
UP_RIGHT
DOWN_RIGHT
«Abstract»
Card
-id: int
Card(int id)
getId() : int
getEvaluator() : Function<PlayerBoard; Integer points>
«Enumeration»
PointConditionType
OBJECTS
CORNERS
«Enumeration»
CardType
RESOURCE
STARTER
GOLD
OBJECTIVE
«Enumeration»
ObjectiveType
GEOMETRIC
COUNTING
realization
PlayableCard
-frontSide: PlayableFrontSide
-backSide: PlayableBackSide
-playedSide: CardSideType[0..1]
-coveredCorners: int
-kingdom: ResourceType[0..1]
PlayableCard(int id, PlayableSide front, PlayableSide back)
PlayableCard(int id, PlayableSide front, PlayableSide back, ResourceType kingdom)
getKingdom() : ResourceType[0..1]
getPlayedSide() : PlayableSide
setPlayedSide(CardSideType sideType) : void
getCoveredCorners() : int
setCoveredCorners(int n) : void
getEvaluator() : Function<PlayerBoard pb; Integer points>
«Abstract»
PlayableSide
-corners: Corner[1..4]
getCorners() : Corner[1..4]
setCorner(CornerPosition position, ResourceType resource)
setCorner(CornerPosition position, ObjectType object)
getEvaluator() : BiFunction<PlayerBoard pb; Integer coveredCorners; Integer points>
composition
1..4
realization
inheritance
composition
composition
Corner<T>
-content: Optional<T>
-isCovered: boolean
Corner()
Corner(T content)
isEmpty() : bool
getContent() : Optional<T>
cover() : void
PlayableBackSide
-permanentResources: ResourceType[1..3]
PlayableBackSide(ResourceType[1..3] permanentResources)
getResources() : ResourceType[1..3]
getEvaluator() : BiFunction<PlayerBoard pb; Integer coveredCorners; Integer points>
1
0..1
ObjectiveCard
-points: int
-objective: Objective
ObjectiveCard(int id, int points, Objective objective)
getEvaluator() : Function<PlayerBoard pb; Integer points>
0..1
composition
1
«Abstract»
Objective
getEvaluator() : Function<PlayerBoard pb; Integer points>
«Abstract»
PlayableFrontSide
getEvaluator() : BiFunction<PlayerBoard pb; Integer CoveredCorners; Integer points>
realization
realization
realization
dependency
dependency
dependency
realization
realization
ResourceCardFrontSide
-points: int
ResourceCardFrontSide(int points)
getEvaluator() : BiFunction<PlayerBoard pb; Integer coveredCorners; Integer points>
inheritance
0..1
dependency
composition
GeometricObjective
-geometry: ResourceType[3][3]
GeometricObjective(ResourceType[3][3] geometry)
getEvaluator() : Function<PlayerBoard pb; Integer points>
CountingObjective
-resources: HashMap<ResourceType; int>
-objects: HashMap<ObjectType; int>
CountingObjective(HashMap<ResourceType; int> resources, HashMap<ObjectType; int> objects)
getEvaluator() : Function<PlayerBoard pb; Integer points>
StarterCardFrontSide
StarterCardFrontSide()
getEvaluator() : BiFunction<PlayerBoard pb; Integer coveredCorners; Integer points>
GoldCardFrontSide
-placementCondition: ResourceType[1..5]
-pointCondition: PointConditionType[0..1]
-pointConditionObject: ObjectType[0..1]
GoldCardFrontSide(int points, ResourceType[1..5] placementCondition, PointConditionType[0..1] pointCondition, ObjectType[0..1] pointConditionObject)
getEvaluator() : BiFunction<PlayerBoard pb; Integer coveredCorners; Integer points>
dependency
dependency
dependency
dependency
dependency
CardBuilder
-id: int
-type: CardType
-points: int[0..1]
-objectiveType: ObjectiveType[0..1]
-objectiveGeometry: ResourceType[3][3][0..1]
-objectiveResources: HashMap<ResourceType, int>[0..1]
-objectiveObjects: HashMap<ObjectType, int>[0..1]
-backPermanentResources: ResourceType[1..3][0..1]
-placementCondition: ResourceType[1..5][0..1]
-pointCondition: PointConditionType[0..1]
-pointConditionObject: ObjectType[0..1]
CardBuilder(int id, CardType type)
setPoints(int points) : CardBuilder <<throws>> WrongCardTypeException
setObjectiveType(ObjectiveType objectiveType) : CardBuilder <<throws>> WrongCardTypeException
setObjectiveGeometry(ResourceType[3][3] objectiveGeometry) : CardBuilder <<throws>> WrongCardTypeException, MissingParametersException
setObjectiveResources(HashMap<ResourceType, int> objectiveResources) : CardBuilder <<throws>> WrongCardTypeException, ConflictingParameterException
setObjectiveObjects(HashMap<ObjectType, int> objectiveObjects) : CardBuilder <<throws>> WrongCardTypeException, ConflictingParameterException
setBackPermanentResources(ResourceType[1..3] backPermanentResources) : CardBuilder <<throws>> WrongCardTypeException
setPlacementCondition(ResourceType[1..5] placementCondition) : CardBuilder <<throws>> WrongCardTypeException
setPointCondition(PointConditionType pointCondition) : CardBuilder <<throws>> WrongCardTypeException
setPointConditionObject(ObjectType pointConditionObject) : CardBuilder <<throws>> WrongCardTypeException
build() : Card <<throws>> MissingParametersException
IllegalStateException
throws
throws
throws
inheritance
inheritance
inheritance
WrongCardTypeException
WrontCardTypeException(String expected, String actual)
MissingParametersException
MissingParametersException(String missing)
ConflictingParameterException
ConflictingParameterException(String paramName, String expectedValue, String currValue)

Rest of the model