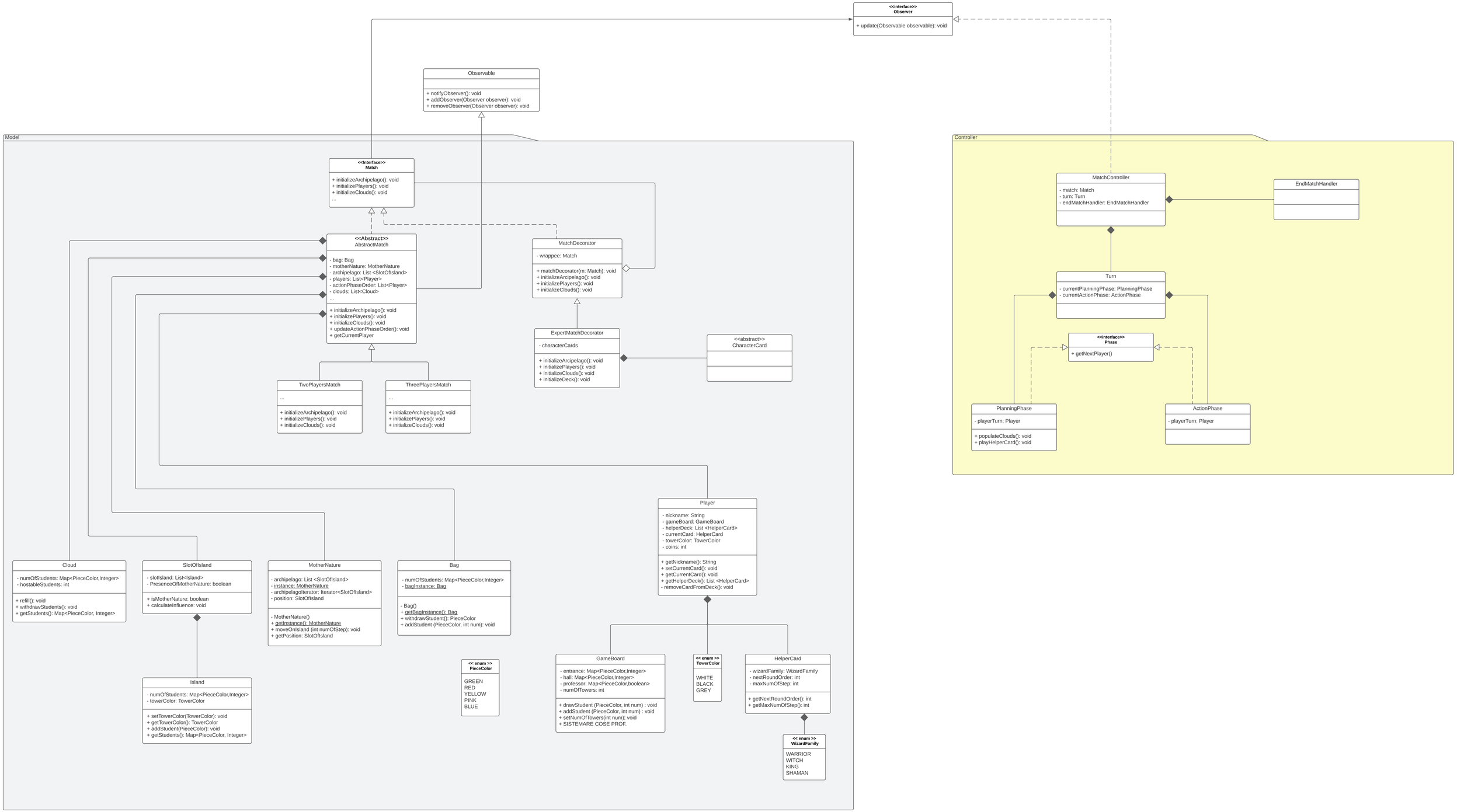
<<interface>>
Observer
+ update(Observable observable): void
Observable
+ notifyObserver(): void
+ addObserver(Observer observer): void
+ removeObserver(Observer observer): void
Model
<<Interface>>
Match
+ initializeArchipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
...
<<Abstract>>
AbstractMatch
- bag: Bag
- motherNature: MotherNature
- archipelago: List <SlotOfIsland>
- players: List<Player>
- actionPhaseOrder: List<Player>
- clouds: List<Cloud>
...
+ initializeArchipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
+ updateActionPhaseOrder(): void
+ getCurrentPlayer
MatchDecorator
- wrappee: Match
+ matchDecorator(m: Match): void
+ initializeArcipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
ExpertMatchDecorator
- characterCards
+ initializeArcipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
+ initializeDeck(): void
<<abstract>>
CharacterCard
TwoPlayersMatch
...
+ initializeArchipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
ThreePlayersMatch
...
+ initializeArchipelago(): void
+ initializePlayers(): void
+ initializeClouds(): void
Cloud
- numOfStudents: Map<PieceColor,Integer>
- hostableStudents: int
+ refill(): void
+ withdrawStudents(): void
+ getStudents(): Map<PieceColor, Integer>
SlotOfIsland
- slotIsland: List<Island>
- PresenceOfMotherNature: boolean
+ isMotherNature: boolean
+ calculateInfluence: void
Island
- numOfStudents: Map<PieceColor,Integer>
- towerColor: TowerColor
+ setTowerColor(TowerColor): void
+ getTowerColor(): TowerColor
+ addStudent(PieceColor): void
+ getStudents(): Map<PieceColor, Integer>
MotherNature
- archipelago: List <SlotOfIsland>
- instance: MotherNature
- archipelagoIterator: Iterator<SlotOfIsland>
- position: SlotOfIsland
- MotherNature()
+ getInstance(): MotherNature
+ moveOnIsland (int numOfStep): void
+ getPosition: SlotOfIsland
Bag
- numOfStudents: Map<PieceColor,Integer>
- bagInstance: Bag
- Bag()
+ getBagInstance(): Bag
+ withdrawStudent(): PieceColor
+ addStudent (PieceColor, int num): void
<< enum >>
PieceColor
GREEN
RED
YELLOW
PINK
BLUE
Player
- nickname: String
- gameBoard: GameBoard
- helperDeck: List <HelperCard>
- currentCard: HelperCard
- towerColor: TowerColor
- coins: int
+ getNickname(): String
+ setCurrentCard(): void
+ getCurrentCard(): void
+ getHelperDeck(): List <HelperCard>
- removeCardFromDeck(): void
GameBoard
- entrance: Map<PieceColor,Integer>
- hall: Map<PieceColor,Integer>
- professor: Map<PieceColor,boolean>
- numOfTowers: int
+ drawStudent (PieceColor, int num) : void
+ addStudent (PieceColor, int num) : void
+ setNumOfTowers(int num); void
+ SISTEMARE COSE PROF.
<< enum >>
TowerColor
WHITE
BLACK
GREY
HelperCard
- wizardFamily: WizardFamily
- nextRoundOrder: int
- maxNumOfStep: int
+ getNextRoundOrder(): int
+ getMaxNumOfStep(): int
<< enum >>
WizardFamily
WARRIOR
WITCH
KING
SHAMAN
Controller
MatchController
- match: Match
- turn: Turn
- endMatchHandler: EndMatchHandler
EndMatchHandler
Turn
- currentPlanningPhase: PlanningPhase
- currentActionPhase: ActionPhase
<<interface>>
Phase
+ getNextPlayer()
PlanningPhase
- playerTurn: Player
+ populateClouds(): void
+ playHelperCard(): void
ActionPhase
- playerTurn: Player