# Descrizione UML gruppo AM035

## Classe Match

Il client indica se vuole giocare una partita a 2, 3 o 4 giocatori, a seconda della sua scelta la classe FactoryMatch torna al Client un oggetto una sottoclasse di Match che rispecchia la suddetta scelta.

Le sottoclassi di Match sono appunto MatchTwoPlayer, MatchThreePlayer e MatchFourPlayer dove più precisamente:

* MatchTwoPlayer e MatchThreePlayer seguono fedelmente i metodi implementati dalla classe astratta Match, la differenza sta nell’inizializzazione degli attributi di Match che saranno diversi a seconda del numero di giocatori.
* MatchFourPlayer contiene, oltre agli attributi di Match, anche due liste di giocatori che indicano le due squadre della partita, e i rispettivi capitani, ossia coloro che avranno le torri nella plancia. Inoltre, la classe presenta un override dei metodi che differiscono da quelli per le partite da 2 e 3 giocatori.

Infine tutte e tre le sottoclassi implementeranno il metodo astratto setGame() con il quale imposteranno, attraverso il metodi della classe Game e gli attributi dello specifico oggetto Match, gli attributi dell’oggetto di tipo Game a seconda del tipo di partita.

Ogni giocatore viene assegnato ad una istanza Player attraverso il nome e l’username.

## Classe Game e Classi in relazione con Game

La classe Game presenta tutti gli attributi necessari per lo sviluppo del gioco.

Di seguito una breve spiegazione degli attributi e delle Classi usate:

* L’attributo archipelagos è una lista di oggetti di classe Archipelago. La classe Archipelago presenta a sua volta una lista di istanze di Island le quali rappresenterebbero le cosiddette Tessere Isola sulla quale verranno costruite le singole torri.

La particolarità della Classe Archipelago è che al procedere del gioco, è permesso “trasferire” (attraverso metodo mergeArchipelago(Archipelago)) la lista di Island di una oggetto Archipelago ad un altro oggetto Archipelago e di conseguenza eliminare la prima istanza in quanto non più utile al gioco.

Si è scelta questa implementazione perché al momento di inizio partita, la classe Game presenterà una lista di 12 elementi di Archipelago, ognuno composto da una singolo oggetto Island nella lista, ma a mano a mano che il gioco continua gli elementi si “fonderanno” tra di loro e il numero di elementi nella lista archipelagos diminuirà, mentre gli istanze di Archipelago al suo interno vedranno la loro lista di Island crescere in numero di elementi.

* L’attributo di Classe StudentBag funge da Sacchetto del gioco, sulla quale possiamo chiamare i metodi per pescare gli studenti.
* La classe Wizard, che viene usata per implementare l’attributo wizards, ha le istanze collegate con le istanze di Player attraverso l’attributo username, che sarà lo stesso per entrambi.

presenta di particolare l’attributo coins, usato solamente da Match decorate ExpertMatch, l’attributo archipelagosOfWizard, con il quale segniamo le isole controllate dal giocatore relativo al mago.

La classe Board che mi copre l’attributo board, invece, rispecchia le caratteristiche della plancia del gioco, presenta dunque un set di Torri, un set di Professore, una collection di TableOfStudent e un set di Studenti (che rappresentano gli studenti nell’ingresso della plancia). L’attributo limitStudentInEntrance indica la quantità di studenti che possono esserci nell’ingresso della plancia.

Infine, la classe AssistantDeck, usata per l’attributo assistantDeck di Wizard, mi gestisce il mazzo di carte assistenti (AssistantCard) di ogni giocatore, una istanza di questa classe si salva le carte giocate nel match (usedCards) e le carte giocabili (playableCards). Attravero i metodi autoesplicativi della classe Wizard e ai metodi della classe AssistantDeck è possibile pescare una carta assistente tenendo conto delle regole di gioco, in caso di scelta di una carta assistente sbagliata verranno mandate exception.

## Expert Match

La classe ExpertMatch è la classe concreta che implementa i metodi relativi alla modalità esperto del gioco, estendendo la classe astratta MatchDecorator, introducendo all’interno del gioco le carte personaggio e le monete.

La classe presenta i seguenti metodi:

* drawCharacterCard(): setta per ogni mago le 3 carte personaggio utilizzabili durante la partita.
* setFirstCoin(): assegna a ogni giocatore la prima moneta all’inizio della partita.
* moveStudentOnBoard(Player, Student): effettua l’override del metodo presente nella classe. astratta match segnalando se lo studente aggiunto è il terzo, il sesto o il nono di quel colore.
* collectCoin(): se sono verificate le condizioni di moveStudentOnBoard(Player, Student) viene chiamato questo metodo e aggiunta una nuova moneta a quelle possedute dal giocatore.
* useCoins(CarachterCard): utilizza le monete per chiamare le carte personaggio e controlla che ne possegga abbastanza per chiamare le carte.
* playCharachterCard(Player, CharacterCard): usa la carta nel gioco.

## DeckCharacterCard

La classe ha come attributo deck, ovvero un set di characterCard.

Il metodo drawCharacterCard() pesca le carte personaggio dalla collection presente nella classe CharacterCard.

## CharacterCard

La classe ha come attributo effects che è una Collection di oggetti Effetti che associa a ogni carta gli effetti corrispondenti.

## Effect

La classe ha come attributo CostOfTheEffect, arraylist di coins, che in base alla grandezza dell’arraylist decide il costo della singola carta personaggio.

Unico metodo è la useEffect, che chiama le sottoclassi che implementano ognuna un singolo effetto associabile alle carte.