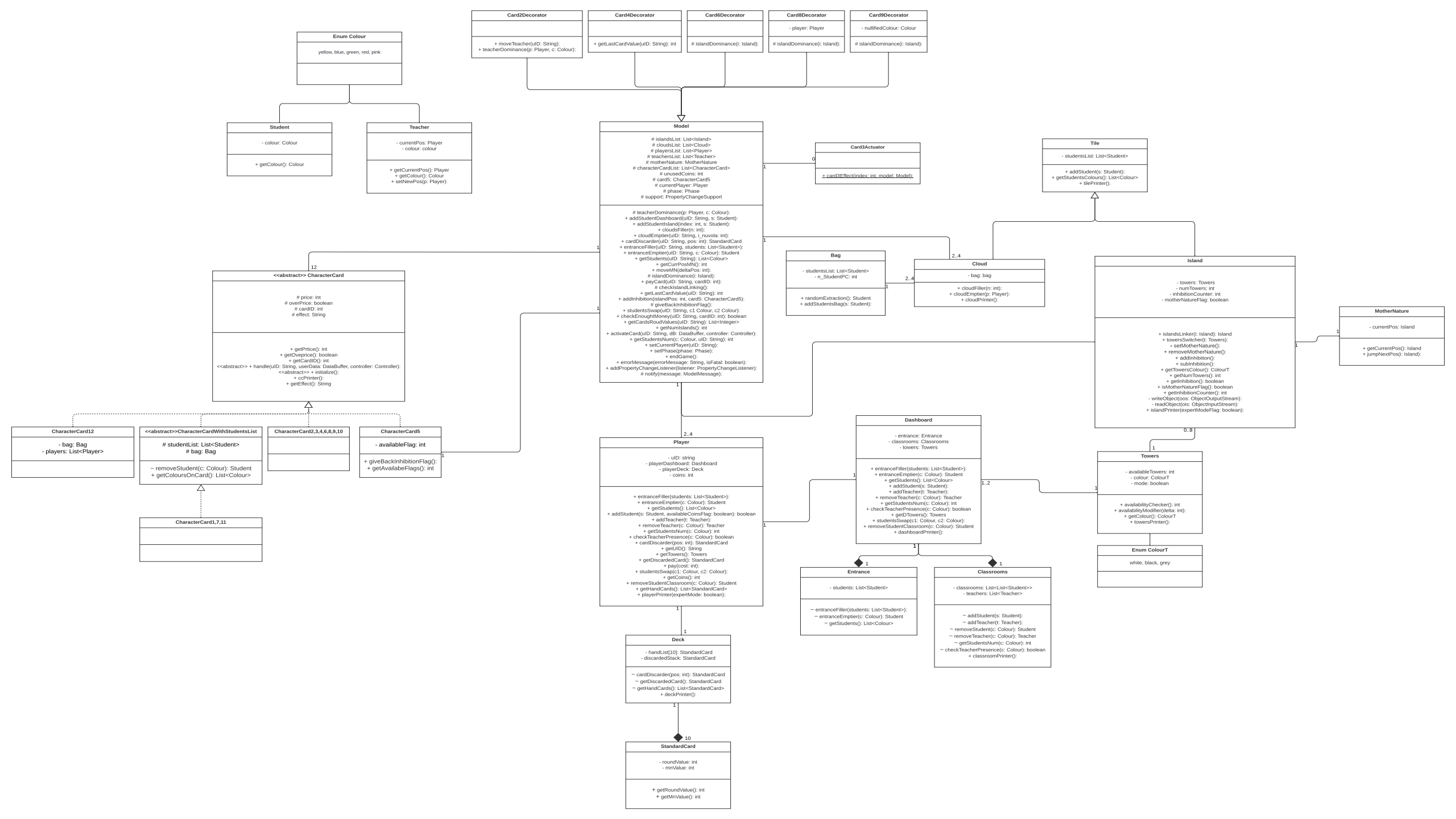
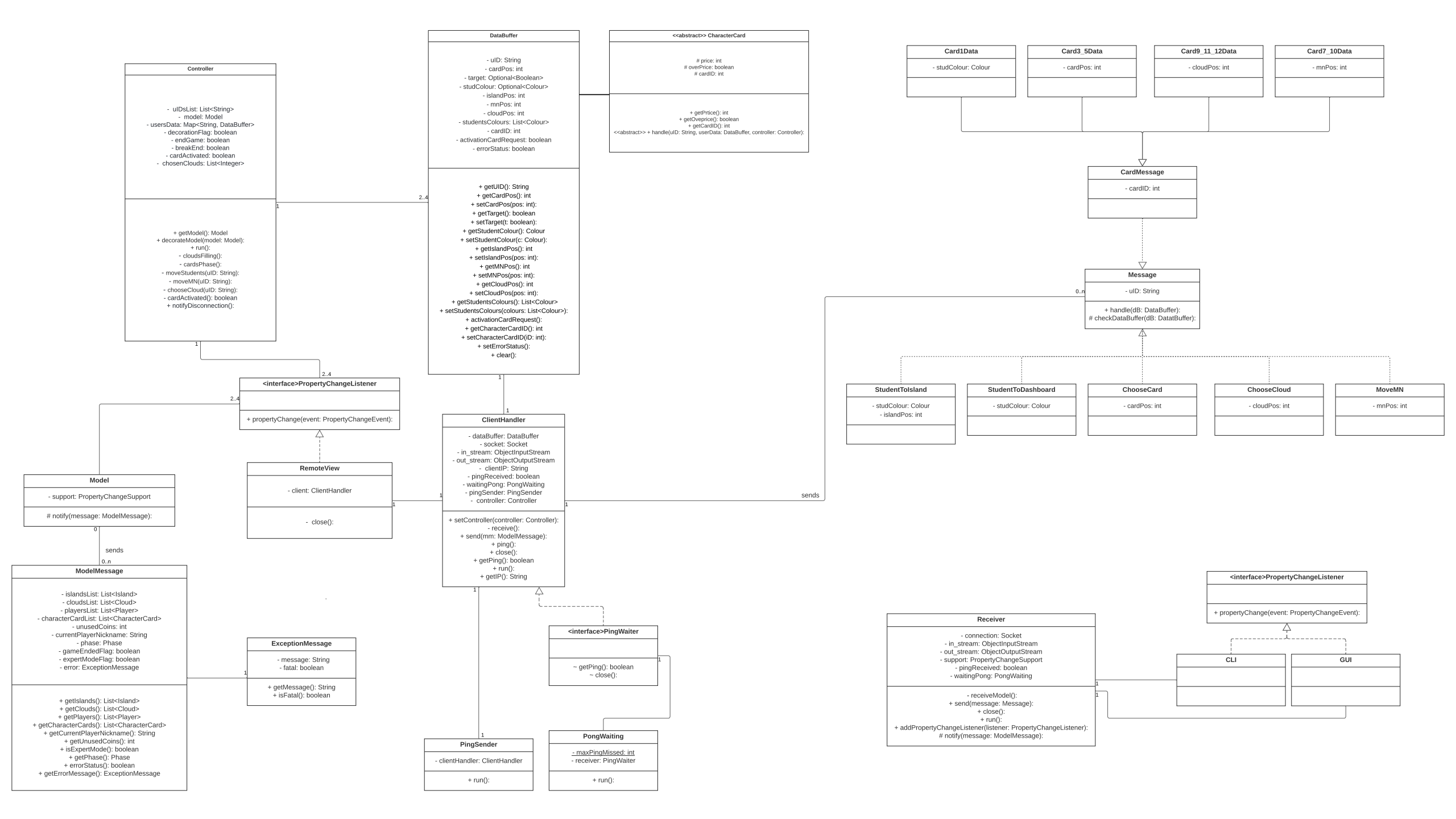
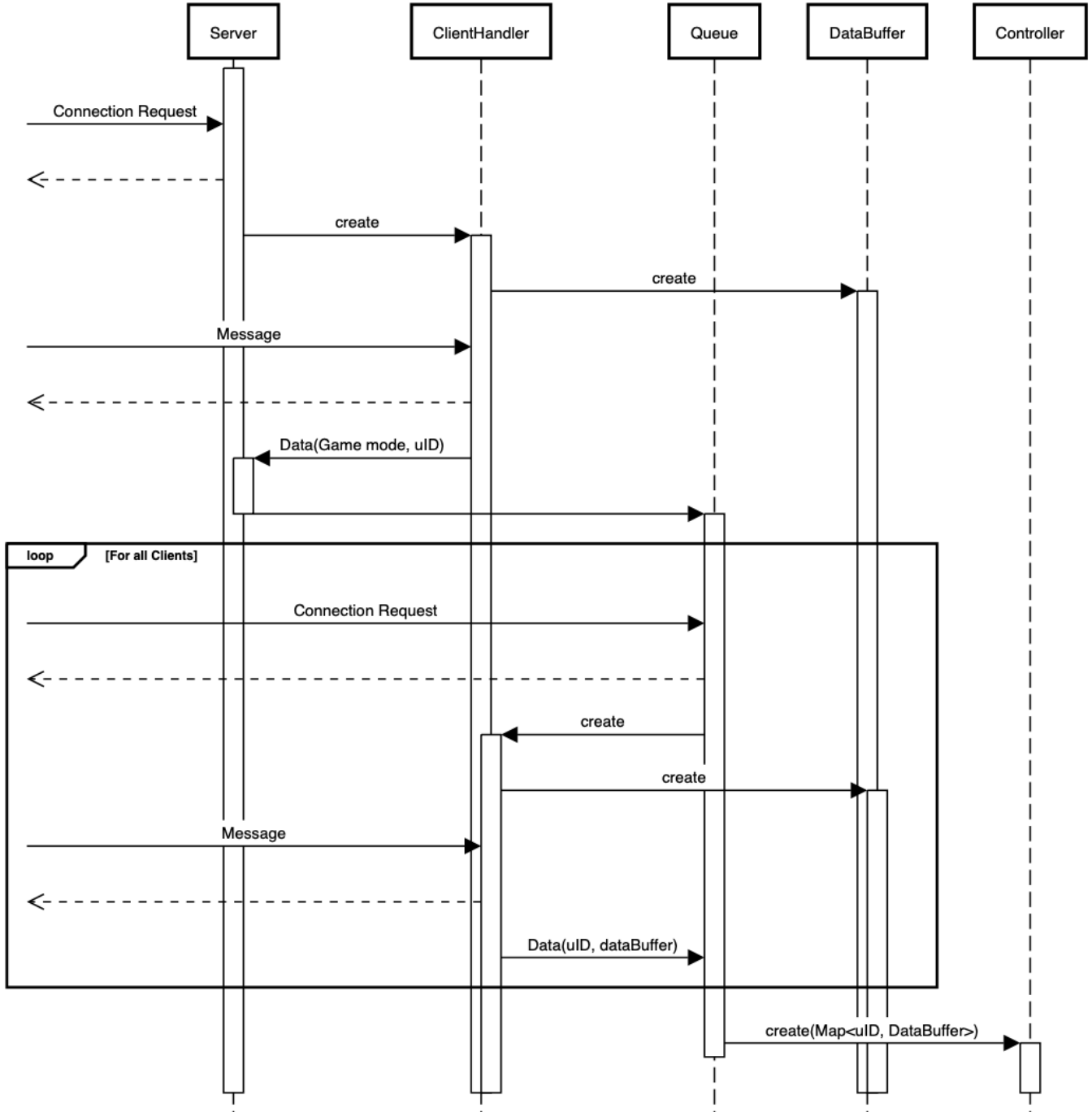
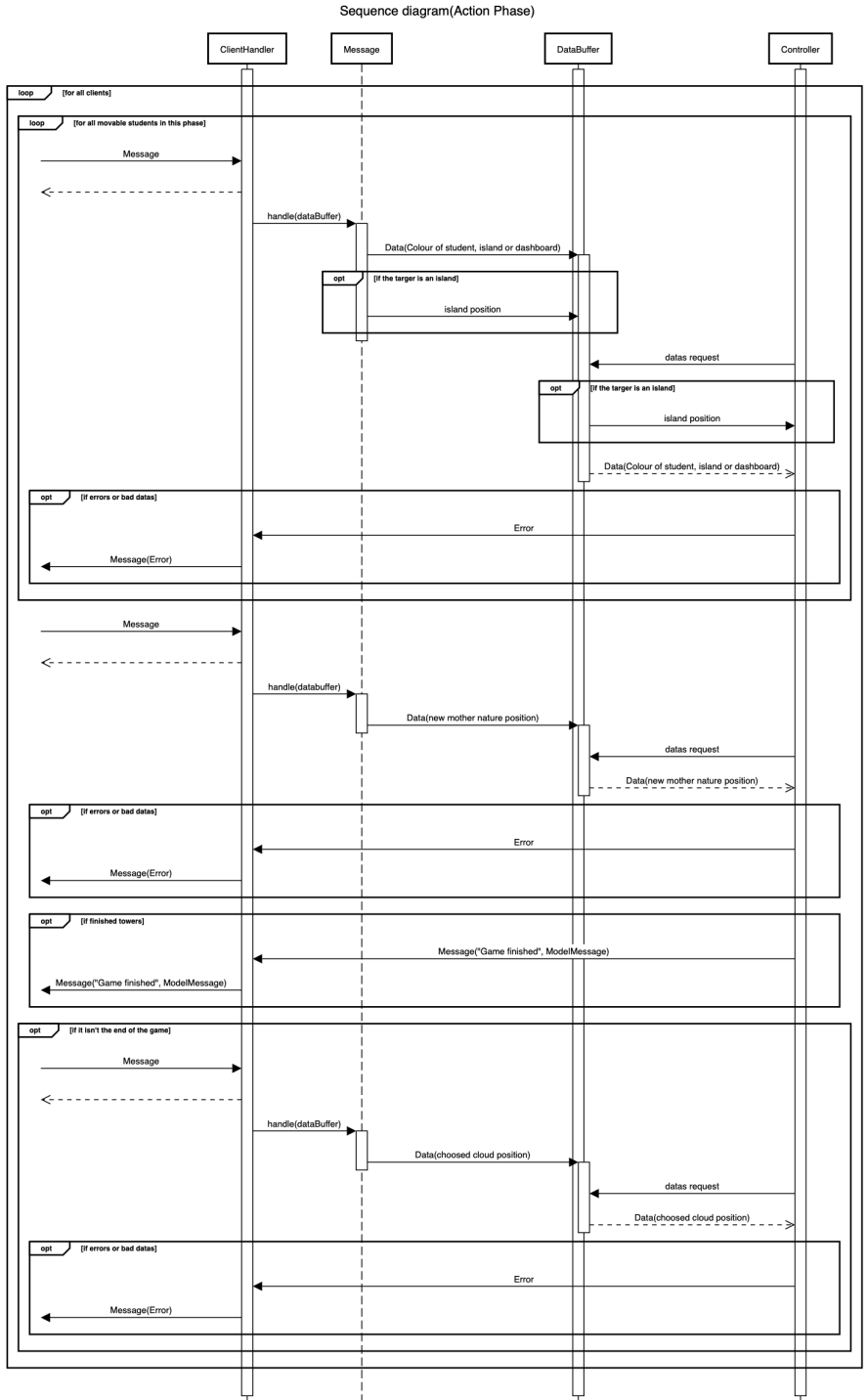
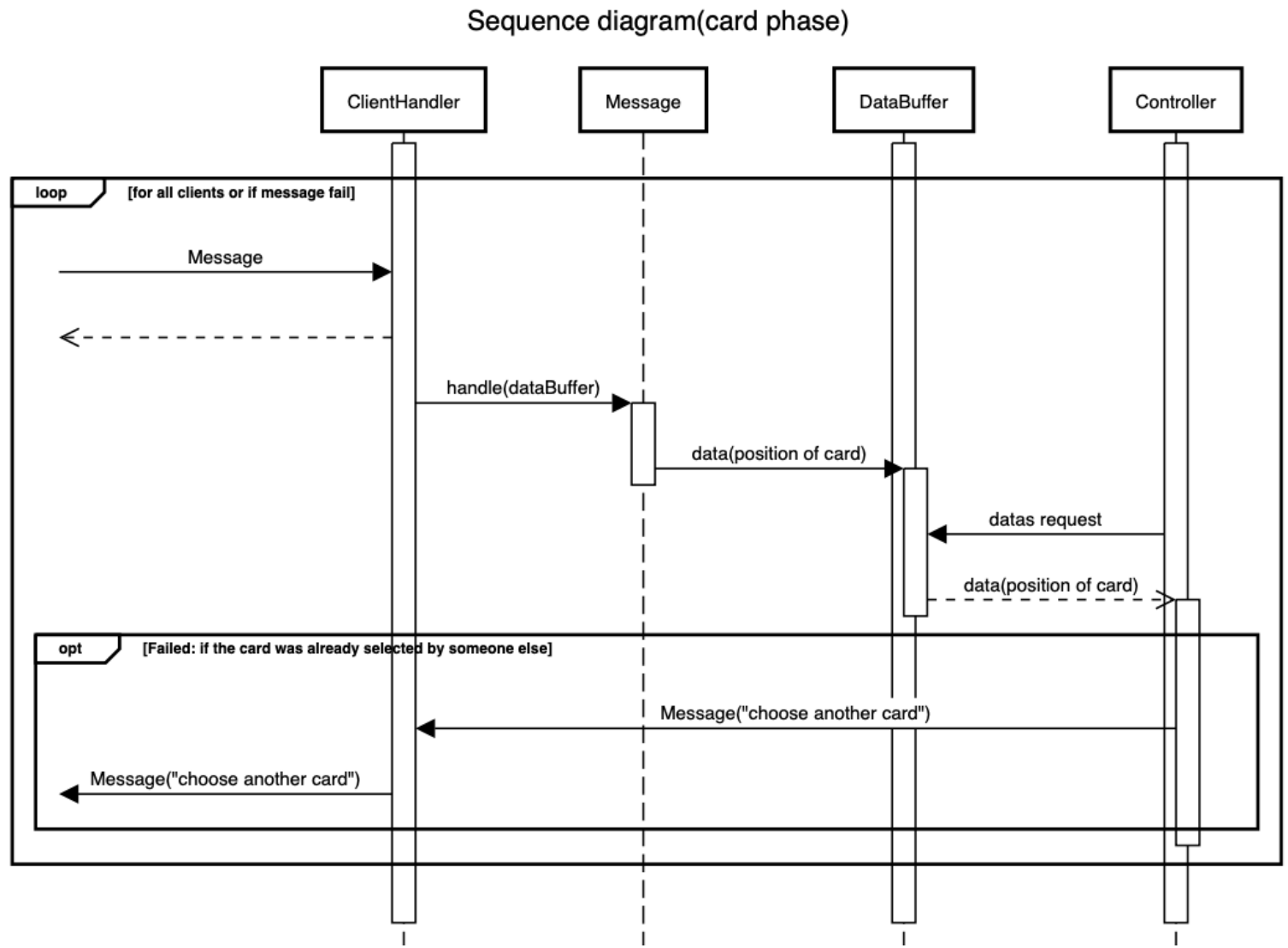

Card2Decorator
+ moveTeacher(uID: String):
+ teacherDominance(p: Player, c: Colour):
Card4Decorator
+ getLastCardValue(uID: String): int
Card6Decorator
islandDominance(i: Island):
Card8Decorator
- player: Player
islandDominance(i: Island):
Card9Decorator
- nullifiedColour: Colour
islandDominance(i: Island):
Enum Colour
yellow, blue, green, red, pink
Student
- colour: Colour
+ getColour(): Colour
Teacher
- currentPos: Player
- colour: colour
+ getCurrentPos(): Player
+ getColour(): Colour
+ setNewPos(p: Player):
Model
islandsList: List<Island>
cloudsList: List<Cloud>
playersList: List<Player>
teachersList: List<Teacher>
motherNature: MotherNature
characterCardList: List<CharacterCard>
unusedCoins: int
card5: CharacterCard5
currentPlayer: Player
phase: Phase
support: PropertyChangeSupport
teacherDominance(p: Player, c: Colour):
+ addStudentDashboard(uID: String, s: Student):
+ addStudentIsland(index: int, s: Student):
+ cloudsFiller(n: int):
+ cloudEmptier(uID: String, i_nuvola: int):
+ cardDiscarder(uID: String, pos: int): StandardCard
+ entranceFiller(uID: String, students: List<Student>):
+ entranceEmptier(uID: String, c: Colour): Student
+ getStudents(uID: String): List<Colour>
+ getCurrPosMN(): int
+ moveMN(deltaPos: int):
# islandDominance(i: Island):
+ payCard(uID: String, cardID: int):
# checkIslandLinking():
+ getLastCardValue(uID: String): int
+ addInhibition(islandPos: int, card5: CharacterCard5):
# giveBackInhibitionFlag():
+ studentsSwap(uID: String, c1 Colour, c2 Colour):
+ checkEnoughtMoney(uID: String, cardID: int): boolean
+ getCardsRoudValues(uID: String): List<Integer>
+ getNumIslands(): int
+ activateCard(uID: String, dB: DataBuffer, controller: Controller):
+ getStudentsNum(c: Colour, uID: String): int
+ setCurrentPlayer(uID: String):
+ setPhase(phase: Phase):
+ endGame():
+ errorMessage(errorMessage: String, isFatal: boolean):
+ addPropertyChangeListener(listener: PropertyChangeListener):
# notify(message: ModelMessage):
Card3Actuator
+ card3Effect(index: int, model: Model):
Tile
- studentsList: List<Student>
+ addStudent(s: Student):
+ getStudentsColours(): List<Colour>
+ tilePrinter():
Bag
- studentsList: List<Student>
- n_StudentPC: int
+ randomExtraction(): Student
+ addStudentsBag(s: Student):
Cloud
- bag: bag
+ cloudFiller(n: int):
+ cloudEmptier(p: Player):
+ cloudPrinter():
Island
- towers: Towers
- numTowers: int
- inhibitionCounter: int
- motherNatureFlag: boolean
+ islandsLinker(i: Island): Island
+ towersSwitcher(t: Towers):
+ setMotherNature():
+ removeMotherNature():
+ addInhibition():
+ subInhibition():
+ getTowersColour(): ColourT
+ getNumTowers(): int
+ getInhibition(): boolean
+ isMotherNatureFlag(): boolean
+ getInhibitionCounter(): int
- writeObject(oos: ObjectOutputStream):
- readObject(ois: ObjectInputStream):
+ islandPrinter(expertModeFlag: boolean):
MotherNature
- currentPos: Island
+ getCurrentPos(): Island
+ jumpNextPos(i: Island):
<<abstract>> CharacterCard
price: int
overPrice: boolean
cardID: int
effect: String
+ getPrtice(): int
+ getOveprice(): boolean
+ getCardID(): int
<<abstract>> + handle(uID: String, userData: DataBuffer, controller: Controller):
<<abstract>> + initialize():
+ ccPrinter():
+ getEffect(): String
CharacterCard12
- bag: Bag
- players: List<Player>
<<abstract>>CharacterCardWithStudentsList
studentList: List<Student>
bag: Bag
− removeStudent(c: Colour): Student
+ getColoursOnCard(): List<Colour>
CharacterCard2,3,4,6,8,9,10
CharacterCard5
- availableFlag: int
+ giveBackInhibitionFlag():
+ getAvailabeFlags(): int
CharacterCard1,7,11
Player
- uID: string
- playerDashboard: Dashboard
- playerDeck: Deck
- coins: int
+ entranceFiller(students: List<Student>):
+ entranceEmptier(c: Colour): Student
+ getStudents(): List<Colour>
+ addStudent(s: Student, availableCoinsFlag: boolean): boolean
+ addTeacher(t: Teacher):
+ removeTeacher(c: Colour): Teacher
+ getStudentsNum(c: Colour): int
+ checkTeacherPresence(c: Colour): boolean
+ cardDiscarder(pos: int): StandardCard
+ getUID(): String
+ getTowers(): Towers
+ getDiscardedCard(): StandardCard
+ pay(cost: int):
+ studentsSwap(c1: Colour, c2: Colour):
+ getCoins(): int
+ removeStudentClassroom(c: Colour): Student
+ getHandCards(): List<StandardCard>
+ playerPrinter(expertMode: boolean):
Dashboard
- entrance: Entrance
- classrooms: Classrooms
- towers: Towers
+ entranceFiller(students: List<Student>):
+ entranceEmptier(c: Colour): Student
+ getStudents(): List<Colour>
+ addStudent(s: Student):
+ addTeacher(t: Teacher):
+ removeTeacher(c: Colour): Teacher
+ getStudentsNum(c: Colour): int
+ checkTeacherPresence(c: Colour): boolean
+ getDTowers(): Towers
+ studentsSwap(c1: Colour, c2: Colour):
+ removeStudentClassroom(c: Colour): Student
+ dashboardPrinter():
Towers
- availableTowers: int
- colour: ColourT
- mode: boolean
+ availabilityChecker(): int
+ availabilityModifier(delta: int):
+ getColour(): ColourT
+ towersPrinter():
Enum ColourT
white, black, grey
Entrance
- students: List<Student>
− entranceFiller(students: List<Student>):
− entranceEmptier(c: Colour): Student
− getStudents(): List<Colour>
Classrooms
- classrooms: List<List<Student>>
- teachers: List<Teacher>
~ addStudent(s: Student):
~ addTeacher(t: Teacher):
~ removeStudent(c: Colour): Student
~ removeTeacher(c: Colour): Teacher
~ getStudentsNum(c: Colour): int
~ checkTeacherPresence(c: Colour): boolean
+ classroomPrinter():
Deck
- handList[10]: StandardCard
- discardedStack: StandardCard
~ cardDiscarder(pos: int): StandardCard
~ getDiscardedCard(): StandardCard
~ getHandCards(): List<StandardCard>
+ deckPrinter():
StandardCard
- roundValue: int
- mnValue: int
+ getRoundValue(): int
+ getMnValue(): int


DataBuffer
- uID: String
- cardPos: int
- target: Optional<Boolean>
- studColour: Optional<Colour>
- islandPos: int
- mnPos: int
- cloudPos: int
- studentsColours: List<Colour>
- cardID: int
- activationCardRequest: boolean
- errorStatus: boolean
+ getUID(): String
+ getCardPos(): int
+ setCardPos(pos: int):
+ getTarget(): boolean
+ setTarget(t: boolean):
+ getStudentColour(): Colour
+ setStudentColour(c: Colour):
+ getIslandPos(): int
+ setIslandPos(pos: int):
+ getMNPos(): int
+ setMNPos(pos: int):
+ getCloudPos(): int
+ setCloudPos(pos: int):
+ getStudentsColours(): List<Colour>
+ setStudentsColours(colours: List<Colour>):
+ activationCardRequest():
+ getCharacterCardID(): int
+ setCharacterCardID(iD: int):
+ setErrorStatus():
+ clear():
<<abstract>> CharacterCard
price: int
overPrice: boolean
cardID: int
+ getPrtice(): int
+ getOveprice(): boolean
+ getCardID(): int
<<abstract>> + handle(uID: String, userData: DataBuffer, controller: Controller):
Controller
- uIDsList: List<String>
- model: Model
- usersData: Map<String, DataBuffer>
- decorationFlag: boolean
- endGame: boolean
- breakEnd: boolean
- cardActivated: boolean
- chosenClouds: List<Integer>
+ getModel(): Model
+ decorateModel(model: Model):
+ run():
- cloudsFilling():
- cardsPhase():
- moveStudents(uID: String):
- moveMN(uID: String):
- chooseCloud(uID: String):
- cardActivated(): boolean
+ notifyDisconnection():
2..4
1
1
2..4
<interface>PropertyChangeListener
+ propertyChange(event: PropertyChangeEvent):
2..4
1
ClientHandler
- dataBuffer: DataBuffer
- socket: Socket
- in_stream: ObjectInputStream
- out_stream: ObjectOutputStream
- clientIP: String
- pingReceived: boolean
- waitingPong: PongWaiting
- pingSender: PingSender
- controller: Controller
+ setController(controller: Controller):
- receive():
+ send(mm: ModelMessage):
+ ping():
+ close():
+ getPing(): boolean
+ run():
+ getIP(): String
1
1
RemoteView
- client: ClientHandler
- close():
1
1
Model
- support: PropertyChangeSupport
notify(message: ModelMessage):
0
sends
0..n
ModelMessage
- islandsList: List<Island>
- cloudsList: List<Cloud>
- playersList: List<Player>
- characterCardList: List<CharacterCard>
- unusedCoins: int
- currentPlayerNickname: String
- phase: Phase
- gameEndedFlag: boolean
- expertModeFlag: boolean
- error: ExceptionMessage
+ getIslands(): List<Island>
+ getClouds(): List<Cloud>
+ getPlayers(): List<Player>
+ getCharacterCards(): List<CharacterCard>
+ getCurrentPlayerNickname(): String
+ getUnusedCoins(): int
+ isExpertMode(): boolean
+ getPhase(): Phase
+ errorStatus(): boolean
+ getErrorMessage(): ExceptionMessage
1
ExceptionMessage
- message: String
- fatal: boolean
+ getMessage(): String
+ isFatal(): boolean
<interface>PingWaiter
~ getPing(): boolean
~ close():
1
1
PingSender
- clientHandler: ClientHandler
+ run():
1
PongWaiting
- maxPingMissed: int
- receiver: PingWaiter
+ run():
sends
0..n
Card1Data
- studColour: Colour
Card3_5Data
- cardPos: int
Card9_11_12Data
- cloudPos: int
Card7_10Data
- mnPos: int
CardMessage
- cardID: int
Message
- uID: String
+ handle(dB: DataBuffer):
# checkDataBuffer(dB: DatatBuffer):
StudentToIsland
- studColour: Colour
- islandPos: int
StudentToDashboard
- studColour: Colour
ChooseCard
- cardPos: int
ChooseCloud
- cloudPos: int
MoveMN
- mnPos: int
Receiver
- connection: Socket
- in_stream: ObjectInputStream
- out_stream: ObjectOutputStream
- support: PropertyChangeSupport
- pingReceived: boolean
- waitingPong: PongWaiting
- receiveModel():
+ send(message: Message):
+ close():
+ run():
+ addPropertyChangeListener(listener: PropertyChangeListener):
# notify(message: ModelMessage):
1
1
<interface>PropertyChangeListener
+ propertyChange(event: PropertyChangeEvent):
CLI
GUI

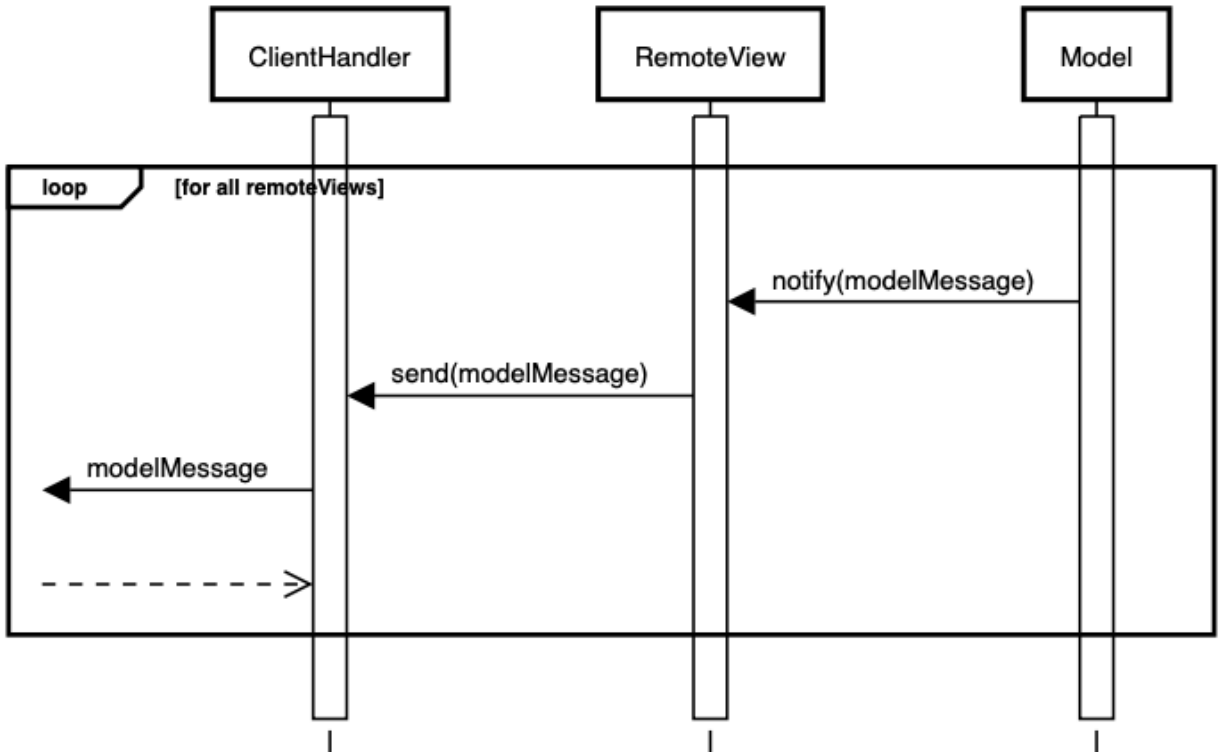
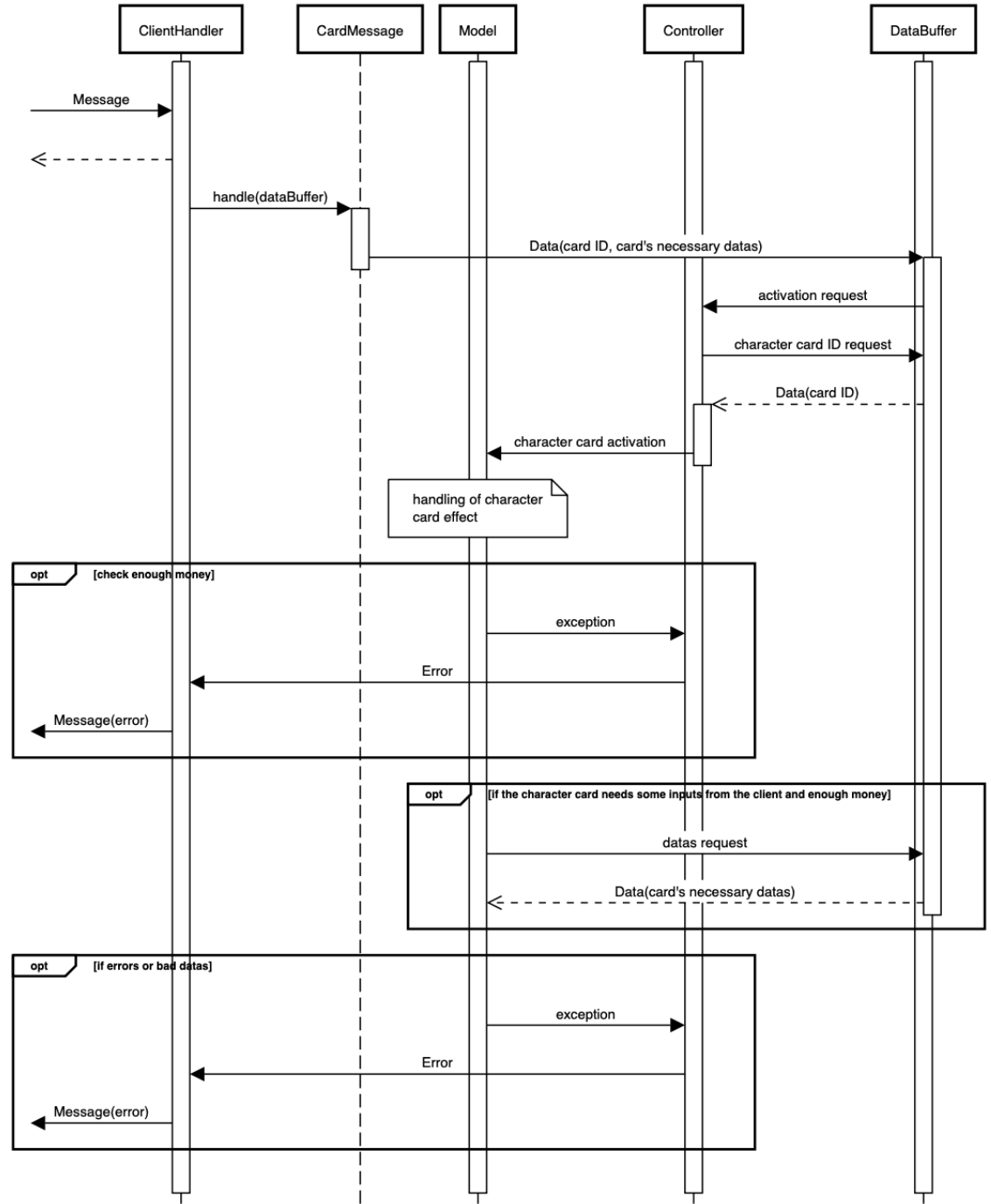
Sequence diagram (Game Creation)




Sequence diagram(Action Phase)
ClientHandler
Message
DataBuffer
Controller
loop
[for all clients]
loop
[for all movable students in this phase]
Message
handle(dataBuffer)
Data(Colour of student, island or dashboard)
opt
[if the targer is an island]
island position
datas request
opt
[if the targer is an island]
island position
Data(Colour of student, island or dashboard)
opt
[if errors or bad datas]
Error
Message(Error)
Message
handle(databuffer)
Data(new mother nature position)
datas request
Data(new mother nature position)
opt
[if errors or bad datas]
Error
Message(Error)
opt
[if finished towers]
Message("Game finished", ModelMessage)
Message("Game finished", ModelMessage)
opt
[if it isn't the end of the game]
Message
handle(dataBuffer)
Data(choosed cloud position)
datas request
Data(choosed cloud position)
opt
[if errors or bad datas]
Error
Message(Error)


Sequence diagram(card phase)
ClientHandler
Message
DataBuffer
Controller
loop
[for all clients or if message fail]
Message
handle(dataBuffer)
data(position of card)
datas request
data(position of card)
opt
[Failed: if the card was already selected by someone else]
Message("choose another card")
Message("choose another card")

Sequence diagram(Changes in Model)



Sequence diagram(character card activation)