pkg

Controller

<<interface>>
Stream E

<<interface>>
List E

GameTest

Controller

Exceptions

<<interface>>
GameTests
+ test_cases : List<GameTest> = new ArrayList<>()
+ arguments_supplier : String = "argumentsSupplier"
+ argumentsSupplier() : Stream<Arguments>

<<bind>>
<E->Arguments>

<<bind>>
<E->GameTest>

<<interface>>
Stream<Arguments>

<<interface>>
List<GameTest>

Server
+ PERMANENT_DISCONNECTION_TIME : int = 60000
- port : int
- client_deletion_threads : Map<Integer,Thread>
+ Server(port : int)
+ startServer() : void
- updatesHandler() : void
+ update(client_handler : SocketHandler, message : Message) : void
+ handleDisconnection(clientID : int) : void
- deleteClient(clientID : int) : void
- hello(client_hander : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- temp1(client_hander : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- temp2(client_hander : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
+ main(argv : String[]) : void

Client
+ RECONNECTION_TRY_INTERVAL : int = 1000
- myClientID : Integer
- nickname : String = "Pippo"
- server_ip : String
- server_port : int
- socket_receiver_thread : Thread
- socket_sender_thread : Thread
+ Client(server_ip : String, server_port : int)
+ handleDisconnection(clientID : int) : void
+ update(socket_handler : SocketHandler, message : Message) : void
- ok(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- ack(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- error(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
+ main(argv : String[]) : void

ServerLobby
+ update(socket_hander : SocketHandler, message : Message) : void
+ handleDisconnect(clientID : int) : void

GameStartedTest
~ createGameTest() : void
- initializedIslands(gameTest : GameTest) : void
- initializedPlayer(gameTest : GameTest) : void
- preparedDashboard(gameTest : GameTest) : void

GameTest
+ designated_expert_mode : boolean
+ designated_num_of_players : int
+ gameName : String
+ clientIDs : List<Integer>
+ GameTest(designated_expert_mode : boolean, designated_num_of_players : int)
+ GameTest(designated_expert_mode : boolean, designated_num_of_players : int, game : Game, clientIDs : List<Integer>)
+ GameTest(gameName : String, designated_expert_mode : boolean, designated_num_of_players : int)
+ GameTest(gameName : String, designated_expert_mode : boolean, designated_num_of_players : int, game : Game, clientIDs : List<Integer>)
+ toString() : String

Model

- incoming_updates
(ordered)

Update
+ Update(client_handler : SocketHandler, message : Message)

- socket_handler

SocketHandler

GameStateTest
~ mode(gameTest : GameTest) : void
~ turnAndPhase(gameTest : GameTest) : void
~ playingOrder(gameTest : GameTest) : void
~ players(gameTest : GameTest) : void
~ decks(gameTest : GameTest) : void
~ coins(gameTest : GameTest) : void
~ clouds(gameTest : GameTest) : void
~ students(gameTest : GameTest) : void
~ motherNature(gameTest : GameTest) : void
~ professors(gameTest : GameTest) : void
~ islands(gameTest : GameTest) : void
~ npcs(gameTest : GameTest) : void

+ game

Game
+ expert_mode : boolean
- game_turn : int
- phase : int
- player_turn : int
- step : int
- moved_students : int
- current_phase_order : List<Integer>
- available_decks : boolean[]
- mother_nature : int
npc_effect : int
bank : int
- last_game_turn : boolean
- game_ended : boolean
- winnerID : int
- serialized_game_copy : byte[]
+ Game(expert_mode : boolean, clientIDs : List<Integer>)
+ Game(random_students : List<Colors>, players : List<Player>, expert_mode : boolean, game_turn : int, phase : int, player_turn : int, step : int, moved_students : int, current_phase_order : List<Integer>, clouds : List<Colors>[], islands : List<Island>, mother_nature : int, uncalimed_professors : List<Colors>, npcs : Npc[], npc_effect : int, bank : int, last_game_turn : boolean, game_ended : boolean, winnerID : int)
+ pickDeck(clientID : int, deck_index : int) : void
+ playCard(clientID : int, card_index : int) : void
+ setStudentToHall(clientID : int, student_index : int) : void
+ setStudentToIsland(clientID : int, student_index : int, island_index : int) : void
+ moveMotherNature(clientID : int, moves : int) : void
+ chooseCloud(clientID : int, cloud_index : int) : void
+ activateEffect(clientID : int, npc_index : int, effect_parameters : EffectParameters) : void
+ skipTurn() : void
+ revertToPreviousState(game : Game) : Game
+ currentlyPlayingPlayer() : int
+ currentPlayersTurnOrder() : List<Integer>
+ getPlayers() : List<Player>
+ getGameTurn() : int
+ getPhase() : int
+ getPlayerTurn() : int
+ getStep() : int
+ getMovedStudents() : int
+ getAvailableDecks() : boolean[]
+ getClouds() : List<Colors>[]
+ getMotherNature() : int
+ getIslands() : List<Island>
+ getUnclaimedProfessors() : List<Colors>
+ getNpcsIDs() : int[]
+ getNpcsCosts() : int[]
+ getNpcEffect() : int
+ getBank() : int
+ isLastGameTurn() : boolean
+ isGameEnded() : boolean
+ getWinnerID() : int
+ getGameSerialization() : byte[]
# isMyTurn(clientID : int) : boolean
# playerIndexFromID(clientID : int) : int
# extractStudents(num : int) : List<Colors>
# disputeIsland(index : int) : void
# checkForMerges() : void
# updateProfessors() : void
# computeWinnerBasedOnTowers() : int
# goToNextGameTurn() : void
# updateSerializedGameCopy() : void

- message

Message

pkg Controller

Controller

<<interface>>
Map

K, V

TestClient

- ip : String
- port : int

+ TestClient(ip : String, port : int)
+ startActualClient() : void
+ main(argv : String[]) : void

Controller

Client

+ RECONNECTION_TRY_INTERVAL : int = 1000
- myClientID : Integer
- nickname : String = "Pippo"
- server_ip : String
- server_port : int
- socket_receiver_thread : Thread
- socket_sender_thread : Thread

+ Client(server_ip : String, server_port : int)
+ handleDisconnect(clientID : int) : void
+ update(socket_handler : SocketHandler, message : Message) : void
- ok(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- ack(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- error(handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
+ main(argv : String[]) : void

<<bind>>
<K->Integer,V->Thread>

<<interface>>
Map<Integer,Thread>

Server

+ PERMANENT_DISCONNECTION_TIME : int = 60000
- port : int
- client_deletion_threads : Map<Integer,Thread>

+ Server(port : int)
+ startServer() : void
- updatesHandler() : void
+ update(client_handler : SocketHandler, message : Message) : void
+ handleDisconnect(clientID : int) : void
- deleteClient(clientID : int) : void
- hello(client_handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- temp1(client_handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
- temp2(client_handler : SocketHandler, arg1 : int, arg2 : int, payload : String) : void
+ main(argv : String[]) : void

ServerLobby

+ update(socket_handler : SocketHandler, message : Message) : void
+ handleDisconnect(clientID : int) : void

- socket_handler

- incoming_updates
(ordered)

key : Integer

- clients

- random_number_generator

SocketHandler

Update

+ Update(client_handler : SocketHandler, message : Message)

ClientData

+ clientID : int
+ nickname : String

+ ClientData(clientID : int, nickname : String, socket_handler : SocketHandler)

Random

+ message

Message

+ socket_handler + socket_handler

SocketHandler

pkg Model

GameTest

+ designated_expert_mode : boolean
+ designated_num_of_players : int
+ gameName : String
+ clientIDs : List<Integer>

+ GameTest(designated_expert_mode : boolean,
designated_num_of_players : int)
+ GameTest(designated_expert_mode : boolean,
designated_num_of_players : int, game : Game, clientIDs :
List<Integer>)
+ GameTest(gameName : String, designated_expert_mode :
boolean, designated_num_of_players : int)
+ GameTest(gameName : String, designated_expert_mode :
boolean, designated_num_of_players : int, game : Game,
clientIDs : List<Integer>)
+ toString() : String

DeckFactory

factoryMethod(deck_id : int) : List<Card>

<<interface>>
List

<<interface>>
Serializable

NpcFactory

factoryMethod(game : Game) : Npc[]

<<bind>>
E->Player

<<bind>>
E->Island

<<bind>>
E->Card

<<bind>>
E->Integer

<<bind>>
E->Colors

Game

+ expert_mode : boolean
- game_turn : int
- phase : int
- player_turn : int
- step : int
- moved_students : int
- current_phase_order : List<Integer>
- available_decks : boolean[]
- mother_nature : int
npc_effect : int
bank : int
- last_game_turn : boolean
- game_ended : boolean
- winnerID : int
- serialized_game_copy : byte[]

+ Game(expert_mode : boolean, clientIDs : List<Integer>)
+ Game(random_students : List<Colors>, players : List<Player>, expert_mode :
boolean, game_turn : int, phase : int, player_turn : int, step : int, moved_students : int,
current_phase_order : List<Integer>, clouds : List<Colors>[], islands : List<Island>,
mother_nature : int, unclaimed_professors : List<Colors>, npcs : Npc[], npc_effect : int,
bank : int, last_game_turn : boolean, game_ended : boolean, winnerID : int)
+ playDeck(clientID : int, deck_index : int) : void
+ playCard(clientID : int, card_index : int) : void
+ setStudentToHall(clientID : int, student_index : int) : void
+ setStudentToIsland(clientID : int, student_index : int, island_index : int) : void
+ moveMotherNature(clientID : int, moves : int) : void
+ chooseCloud(clientID : int, cloud_index : int) : void
+ activateEffect(clientID : int, npc_index : int, effect_parameters : EffectParameters) :
void
+ skipTurn() : void
+ revertToPreviousState(game : Game) : Game
+ currentlyPlayingPlayer() : int
+ currentPlayersTurnOrder() : List<Integer>
+ getPlayers() : List<Player>
+ getGameTurn() : int
+ getPhase() : int
+ getPlayerTurn() : int
+ getStep() : int
+ getMovedStudents() : int
+ getAvailableDecks() : boolean[]
+ getClouds() : List<Colors>[]
+ getMotherNature() : int
+ getIslands() : List<Island>
+ getUnclaimedProfessors() : List<Colors>
+ getNpcsIDs() : int[]
+ getNpcsCosts() : int[]
+ getNpcEffect() : int
+ getBank() : int
+ isLastGameTurn() : boolean
+ isGameEnded() : boolean
+ getWinnerID() : int
+ getGameSerialization() : byte[]
# isMyTurn(clientID : int) : boolean
# playerIndexFromID(clientID : int) : int
# extractStudents(num : int) : List<Colors>
# disputeIsland(index : int) : void
# checkForMerge() : void
# updateProfessors() : void
# computeWinnerBasedOnTowers() : int
# goToNextGameTurn() : void
# updateSerializedGameCopy() : void

EffectParameters

+ args : List<Integer>

+ EffectParameters(args : Integer)

Random

-ran

+ game

<<interface>>
List<Player>

<<interface>>
List<Island>

<<interface>>
List<Card>

<<interface>>
List<Integer>

<<interface>>
List<Colors>

#islands

Island

- students : int[]
- owner_index : Integer
- num_of_merged_islands : int
- interdiction : boolean

+ Island(starting_student : Colors)
+ getStudents(color : Colors) : int
+ getStudents(index : int)
+ addStudent(color : Colors) : void
+ getOwnerIndex() : Integer
+ setOwnerIndex(index : int) : void
+ getNumOfMergedIslands() : int
+ getInterdiction() : boolean
# setInterdiction(interdiction : boolean) : void
# merge(island : Island) : boolean

* players
(ordered)

Player

+ clientID : int
+ player_index : int
- coins : int

+ Player(clientID : int, initial_dashboard_entrance :
List<Colors>, players_number : int, player_index : int,
coins : int)
+ getDashboard() : Dashboard
+ getCards() : List<Card>
# playCard(card_index : int) : void
# addDeckToHand(deck_index : int) : void
+ getLastCardPlayed() : Card
+ getCoins() : int
# addCoin() : void
# pay(cost : int) : void

<<interface>>
Npc

+ activateEffect(game : Game, effect_parameters : EffectParameters) : void
+ getId() : int
+ getCost() : int
+ extraEffect() : void

Npc1

+ uid : int = 1
- resolves_on_activation : boolean = true
+ base_cost : int = 1
- cost : int

+ Npc1(students : List<Colors>)
+ activateEffect(game : Game, effect_parameters : EffectParameters) : void
+ getId() : int
+ getCost() : int
+ extraEffect() : void

Npc2

+ uid : int = 2
- resolves_on_activation : boolean = false
+ base_cost : int = 2
- cost : int

+ Npc2()
+ activateEffect(game : Game, effect_parameters : EffectParameters) : void
+ getId() : int
+ getCost() : int
+ extraEffect() : void

* cards
(ordered)

- last_card_played

Card

+ id : int
+ order_value : int
+ movements_value : int

+ Card(id : int, order_value : int, movements_value : int)
+ equals(o : Object) : boolean
+ hashCode() : int

- dashboard

Dashboard

- hall : int[]
- professors : boolean[]
- rooks : int

+ Dashboard(entrance : List<Colors>, players_number : int,
player_index : int)
+ getEntrance() : List<Colors>
# removeFromEntrance(index : int) : Colors
# addToEntrance(to_add : List<Colors>) : void
+ getHallRow(index : int) : int
# addStudentToHall(color : Colors) : void
+ getProfessor(index : int) : boolean
# setProfessor(color : Colors, status : boolean) : void
+ getRooks() : int
# increaseRooks(quantity : int) : void
# decreaseRooks(quantity : int) : void

- npcs

- unclaimed_professors   - random_students   - students   - entrance
(ordered)   (ordered)   (ordered)   (ordered)

<<enum>>
Colors

+ <<enum constant>> YELLOW : Colors = Colors(0)
+ <<enum constant>> BLUE : Colors = Colors(1)
+ <<enum constant>> GREEN : Colors = Colors(2)
+ <<enum constant>> RED : Colors = Colors(3)
+ <<enum constant>> PURPLE : Colors = Colors(4)
+ index : int

- Colors(index : int)
+ fromColorIndex(index : int) : Colors

+ rooks_color

<<enum>>
Rooks

+ <<enum constant>> WHITE : Rooks = Rooks(0)
+ <<enum constant>> BLACK : Rooks = Rooks(1)
+ <<enum constant>> GREY : Rooks = Rooks(2)
+ index : int

+ Rooks(index : int)
+ fromPlayerIndex(index : int) : Rooks