War of F(u)nctions
Messir Analysis Document v1.0

Georgi Tyufekchiev

January 27, 2023

Contents

Notes

- For the pre-protocol of oeLoadGame it is stated the user needs a save file which is not empty and exists. In the final version the oeLoadGame is always available but in case there is nothing to load, just return error message.
- Throughout the document, I state that the admin receives file signatures, and sends them back when requested to verify integrity of save files. On implementation level, this is done directly with the server (for simplicity).
- Update the Concept model diagrams at the beginning of the section

1 Introduction

1.1 Overview

War of F(u)nctions is a simple game for anyone who wants to test their knowledge of mathematical functions. It is a two-player game, where one person tries to guess the function of the other one. The goal is to win games, earn achievements and in-game money to unlock new game modes.

1.2 Purpose and recipients of the document

This document is an analysis document complying with the **Messir** methodology. Its intent is to provide specification of the functional properties of the game *War of F(u)nctions*. The recipients of this document are:

- The game buyer company (i.e me myself): the document will be used as a basis for the validation of the game using specification based testing.
- The game development company (i.e me myself) will use this document for the development process - design and implementation. It is also used for verification and validation according to the **Messir** methodology.

1.3 Application Domain

The game *War of F(u)nctions* belongs to the Educational Domain. It is dedicated to people who want to learn more about how mathematical functions work. The game is not certified by any educational entity.

1.4 Definitions, acronyms and abbreviations

This section will provide high-level definitions of some mathematical terms. The goal is to use the same notations throughout the whole document.

- **Function** of the form $f : X -> Y$ is an assignment of an element in X to an element in Y. The following notation will be used: $f(x) = y$
- **Linear** function is a straight line on the coordinate plane. The following notation will be used: $f(x) = ax + b$
- **Quadratic** function is a parabola on the coordinate plane. The following notation will be used: $f(x) = ax^2 + bx + c$
- **Cubic** function is a curve on the coordinate plane. The following notation will be used: $f(x) = ax^3 + bx^2 + cx + d$

2 General Description

The information in this section is intended to present the product for which the **Messir** analysis is provided.

2.1 Domain Stakeholders

This section will present the stakeholders of the product. A brief description will be given of each stakeholder, along with their objectives and responsibilities. Both the objectives and responsibilities describe actions that are expected from the stakeholders. In this version of the document, a high level of use cases (summary level) will be presented.

2.1.1 Humans

A human is any person who is able to play the game. The objectives of the player are:

- Win games against another player.
- Earn achievements and in-game money.
- Unlock new game modes.

To achieve these objectives, the player has the following responsibilities:

- Start a game against a second player and play honestly.
- Save their in-game progress.
- Keep track of their in-game money and spend it accordingly.

2.1.2 Administrators

The administrator is an employee of the gaming company that will acquire the game. The objectives of an administrator are:

- Add a player to the game.
- Ban a player in the game.
- Restore corrupted files.

To achieve these objectives, the administrator has the following responsibilities:

- Have a valid reason to ban a player.
- Keep track of the integrity of files.
- Have backup files in case of corrupted ones.

2.2 System's Actors

Among all the stakeholders presented in the previous section we can determine two types of direct actors:

- actHuman: for the Human stakeholder
- actAdministrator: for the Administrator stakeholder

2.3 Use Cases Model

This section contains summary level use cases. The use cases are textually descirbed as suggested by the **Messir** method.

2.3.1 Use Cases

2.3.1.1 summary-suPlaySaveExit

The goal is to play a single game, save the progress of the players and then exit the game.

Use-Case Description
Name suPlaySaveExit *Scope* system *Level* summary
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
The goal is to play a game, save the player's progress and exit the game

Reuse
ugPlayGame [2..2] oeSaveGame [1..2] oeExitGame[1..2]
Protocol condition(s)
The player can start the game
Pre-condition(s)
The player will enter a valid username and choose a valid playing option
Main post-condition(s)
The player has successfully played a game, updating their statistics appropriately, created a save file, and exited the game.
Main Steps
a the actor actHuman executes the ugPlayGame use case b the actor actHuman executes the ugSaveGame use case c the actor actHuman executes the oeExitGame use case
Step Ordering Constraints
1 step (a) must be always the first step. 2 step (b) is optional. 3 step (c) is always the last step.

2.3.1.2 summary-suLoadUnlockCheckBan

The goal is to load the saved progress of the player(s), unlock a new game mode, and then play a game.

Use-Case Description
Name suLoadUnlockCheckBan *Scope* system *Level* summary
Primary actor(s)
1 actHuman [active] 2 actAdministrator [active]
Secondary actor(s)
None
Goal(s) description
The goal is for the player to load saved progress, check money and achievements, buy a game mode. While loading the file, the admin will check if the file is corrupted. The admin will check the player stats and ban the player due to cheating.

Reuse
oeLoadGame [1..2]
ugCheckIntegrity [1..1]
oeRestoreFile [1..1]
ugCheckProgress [1..2]
ugUnlock [1..2]
oeCheckStats [1..1]
oeBanPlayer [1..1]
Protocol condition(s)
The player can start the game. The admin is connected to the system.
Pre-condition(s)
There is a file with saved progress, which is not empty and not corrupted.
Main post-condition(s)
The save file is not corrupted, the progress is loaded successfully, the player is able to play new game mode(s). A player has been banned, erasing their progress.
Main Steps
a the actor actHuman executes the oeLoadGame use case
b the actor actAdministrator executes the ugCheckIntegrity use case
c the actor actAdministrator executes the oeRestoreFile use case
d the actor actHuman executes the ugCheckProgress use case
e the actor actHuman executes the ugUnlock use case
f the actor actAdministrator executes the oeCheckStats use case
g the actor actAdministrator executes the oeBanPlayer use case
Step Ordering Constraints
1 step (a) must be always the first step.
2 step (b) must be the second step.
3 step (c) is required if step (b) fails.
4 step (d) is optional.
5 step (f) is optional.

2.3.1.3 summary-suPlaySaveExit

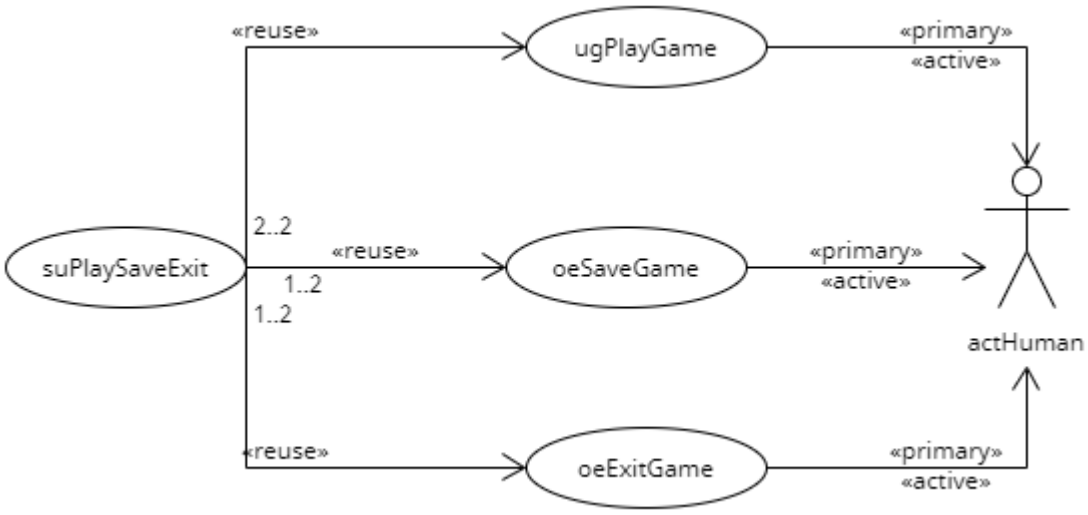

Figure 1: suPlaySaveExit summary use case

2.3.1.4 summary-suLoadUnlockCheckBan

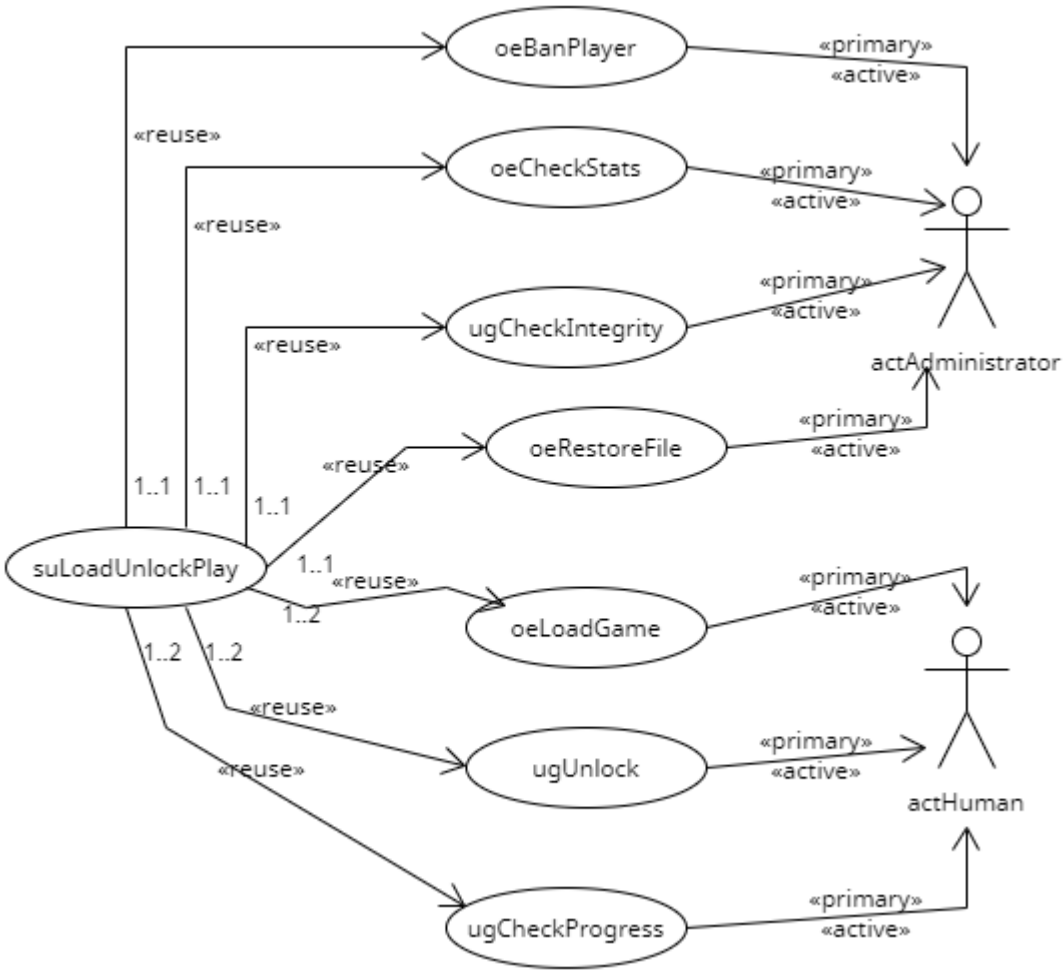

Figure 2: suLoadUnlockCheckBan summary use case

2.3.1.5 usergoal - ugPlayGame

The goal of the player is to start a game against another player.

Use-Case Description
Name ugPlayGame
Scope system
Level usergoal
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
the actHuman's goal is to play a game against another player
Reuse
oeSendUsername [2..2]
oeChooseMode [1..1]
oeSendCoeff[1..2]
Protocol condition(s)
The player can start the game
Pre-condition(s)
The player entered a valid username.
Main post-condition(s)
The player has successfully played a game against another player.
Main Steps
a the actor actHuman executes the oeSendUsername use case
b the actor actHuman executes the oeChooseMode use case
c the actor actHuman executes the oeSendCoeff use case
Step Ordering Constraints
1 step (a) must be the first step.
2 step (b) must be the second step.
3 step (c) must be last step.

2.3.1.6 usergoal - ugUnlock

The goal of the player is to purchase new game mode.

Use-Case Description
Name ugUnlock
Scope system
Level usergoal

Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
the actHuman's goal is to purchase a new game mode
Reuse
oeDisplayModes [1..2] oePurchaseMode [1..2]
Protocol condition(s)
The player can start the game
Pre-condition(s)
The player has enough money. The game mode has not been purchased yet.
Main post-condition(s)
The new game mode is available to the player. Money is subtracted from the player's account. The purchased game mode is not displayed in the "unlock menu".
Main Steps
a the actor actHuman executes the oeDsiplayModes use case b the actor actHuman executes the oePurchaseMode use case
Step Ordering Constraints
1 step (a) must be the first step. 2 step (b) must be the second step.

2.3.1.7 usergoal - ugCheckProgress

The goal of the player is to check their progress in the game.

Use-Case Description
Name ugCheckProgress *Scope* system *Level* usergoal
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
The player has chosen a valid option
Goal(s) description
the actHuman's goal is to check their available money, unlocked achievements.
Reuse
oeCheckMoney [1..2]

oeCheckAchievements [1..2]
Protocol condition(s)
The player can start the game
Pre-condition(s)
The player has chosen a valid option
Main post-condition(s)
The money/achievements is displayed to the player.
Main Steps
a the actor actHuman executes the oeCheckMoney use case b the actor actHuman executes the oeCheckAchievements use case
Step Ordering Constraints
1 step ordering does not matter.

2.3.1.8 usergoal - ugCheckIntegrity

The goal of the administrator is check the integrity of the save file and restore it if it corrupted.

Use-Case Description
Name ugCheckIntegrity *Scope* system *Level* usergoal
Primary actor(s)
1 actAdministrator [active]
Secondary actor(s)
None
Goal(s) description
The actAdministrator's goal is to ensure the save file is not corrupted
Reuse
oeSendSignature [1..1] oeRestoreFile [1..1]
Protocol condition(s)
The admin is connected to the system
Pre-condition(s)
A save file exists, which is not empty
Main post-condition(s)
The file is not corrupted.
Main Steps
a the actor actHuman executes the oeSendSignature use case b the actor actHuman executes the oeRestoreFile use case
Step Ordering Constraints

1 step (a) must be the first step. 2 step (b) must be executed only if the file is corrupted.

2.3.1.9 usergoal - ugPlayGame

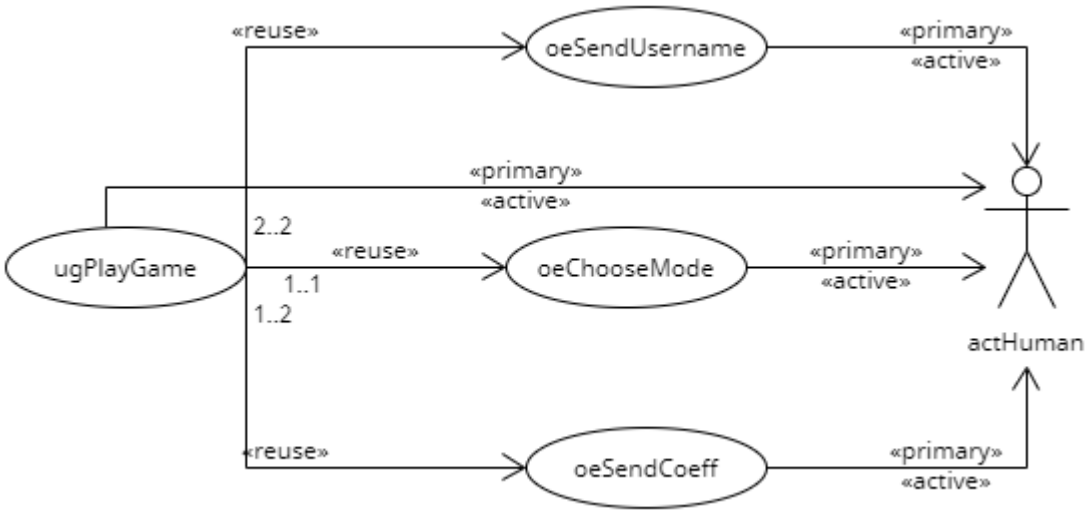

Figure 3: ugPlayGame user goal use case

2.3.1.10 usergoal - ugUnlock

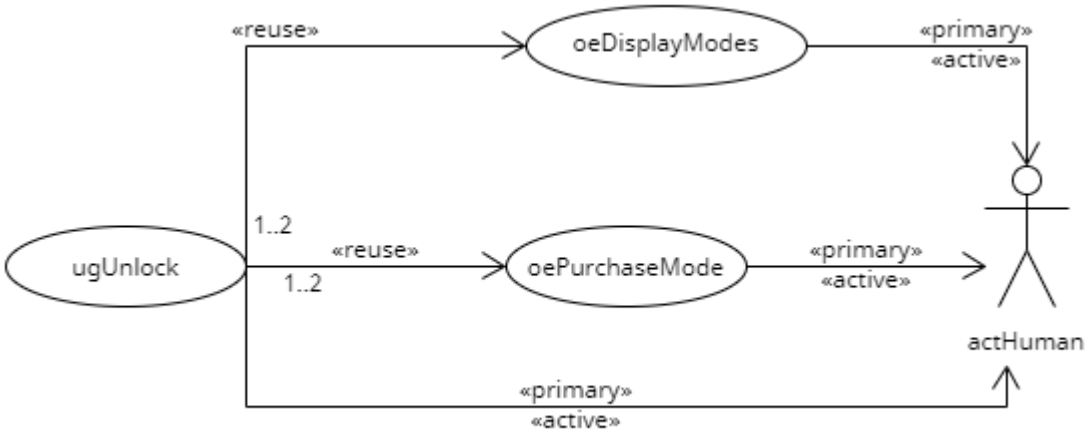

Figure 4: ugUnlock user goal use case

2.3.1.11 usergoal - ugCheckProgress

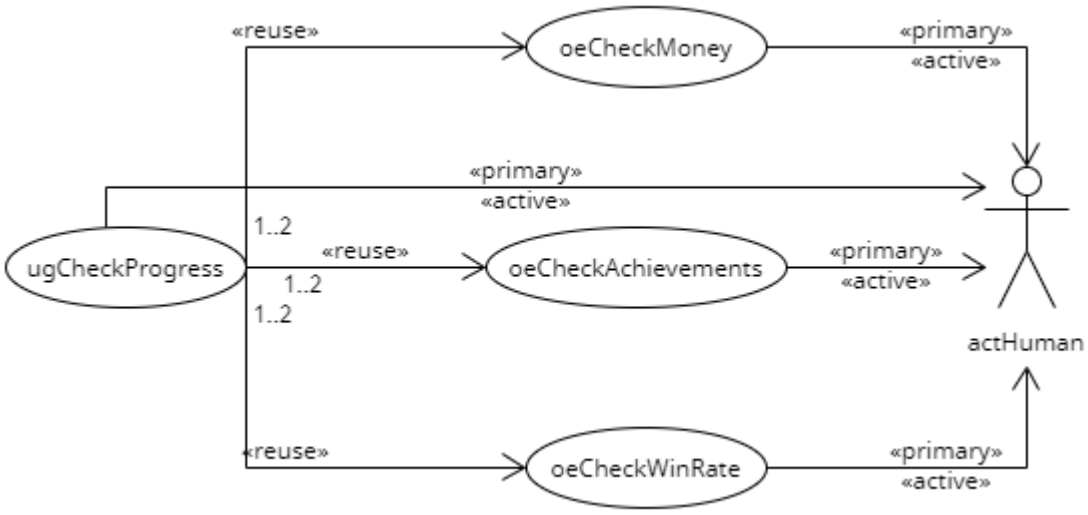

Figure 5: ugCheckProgress user goal use case

2.3.1.12 usergoal - ugCheckIntegrity

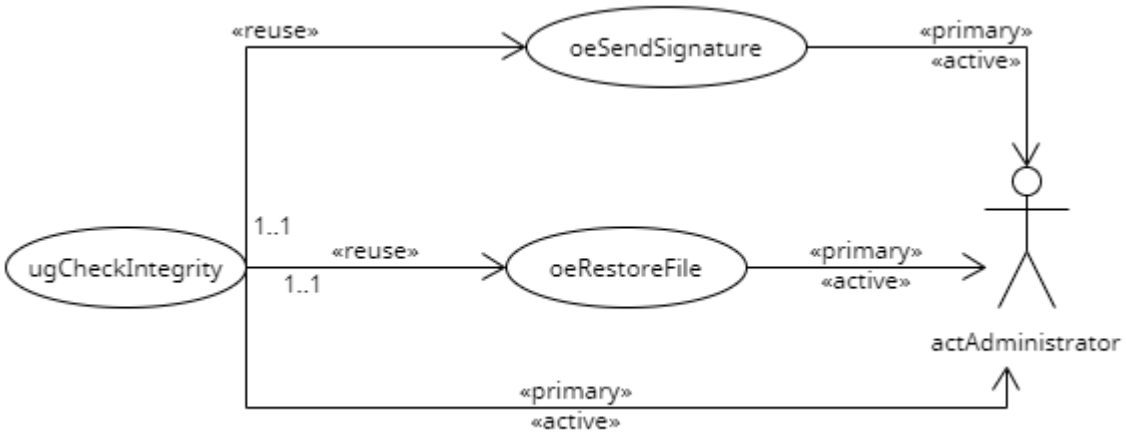

Figure 6: ugCheckIntegrity user goal use case

2.3.1.13 subfunction - oeSendCoeff

The goal is to send valid coefficients to the system.

Use-Case Description
Name oeSendCoeff *Scope* system *Level* subfunction
Parameters
AdtCoeff: dtCoeff
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
The actHuman's goal is to guess the other player's function. They do this by sending coefficients for the polynomial to the system
Protocol condition(s)
The player can start the game and has chosen a game mode to play
Pre-condition(s) A game has been started.
Main post-condition(s)
A valid set of coefficients have been send to the system.
Additional Information
none

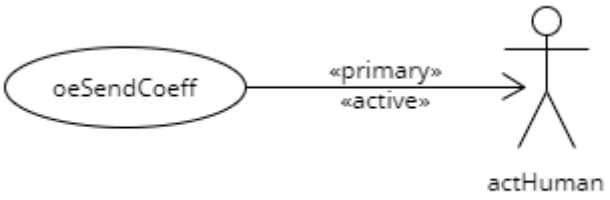

Figure 7: oeSendCoeff subfunction use case

2.3.1.14 subfunction - oePurchaseMode

The goal is to purchase a new game mode.

Use-Case Description
Name oePurchaseMode *Scope* system *Level* subfunction
Parameters
AdtMode: etMode
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
The actHuman's goal is to purchase a new game mode to play
Protocol condition(s)
The player can start the game
Pre-condition(s)
The player has chosen a valid option
Main post-condition(s)
1 The new game mode is available to the player 2 Money has been subtracted from the player's account 3 The game mode is not displayed in the unlock menu 4 Insufficient funds lead to an error message
Additional Information
none

2.3.1.15 subfunction - oeSaveGame

The goal is to save the player's game progress.

Use-Case Description
Name oeSaveGame *Scope* system *Level* subfunction
Parameters
none
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None

Goal(s) description
The actHuman's goal is to save their game's progress
Protocol condition(s)
The player can start the game
Pre-condition(s) The player has entered their username
Main post-condition(s)
1 The game progress has been saved in a file 2 A signature of the file has been sent to the administrator
Additional Information
The user only chooses the option to save their progress and system will do the rest automatically. The user does not need to provide any additional information.

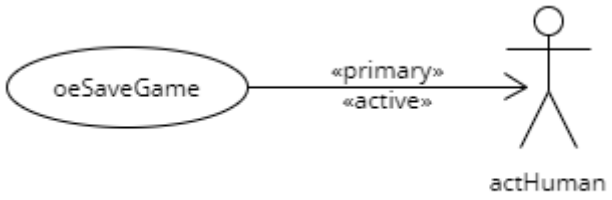

Figure 8: oeSaveGame subfunction use case

2.3.1.16 subfunction - oeLoadGame

The goal is to load the player's game progress.

Use-Case Description
Name oeLoadGame *Scope* system *Level* subfunction
Parameters None
Primary actor(s)
1 actHuman [active]
Secondary actor(s)
None
Goal(s) description
The actHuman's goal is to load their game progress

Protocol condition(s)
The player can start the game
Pre-condition(s) A non-empty and not-corrupted save file exists
Main post-condition(s)
1 Money has been loaded 2 Achievements have been loaded 3 Unlocked game modes have been loaded
Additional Information
none

2.3.1.17 subfunction - oeSendSignature

The goal is to send a file signature to the system.

Use-Case Description
Name oeSendSignature *Scope* system *Level* subfunction
Parameters
AdtHash: dtHash
Primary actor(s)
1 actAdministrator [active]
Secondary actor(s)
None
Goal(s) description
The actAdministrator's goal is to send a file signature to the system
Protocol condition(s)
The administrator has access to the system
Pre-condition(s) A request for the signature has been received
Main post-condition(s)
1 File signature has been reliably send to the system
Additional Information
We assume the administrator is an honest party and will provide the correct signature.

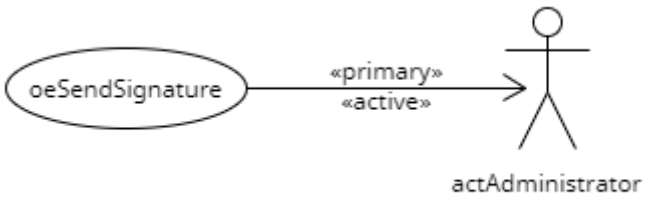

Figure 9: oeSendSignature subfunction use case

2.3.1.18 subfunction - oeRestoreFile

The goal is to restore a corrupted save file with a valid one.

Use-Case Description
Name oeRestoreFile *Scope* system *Level* subfunction
Parameters
AdtFileName: dtFileName
Primary actor(s)
1 actAdministrator [active]
Secondary actor(s)
None
Goal(s) description
The actAdministrator's goal is to restore the save file
Protocol condition(s)
The administrator has access to the system
Pre-condition(s) 1 The system has identified the file as corrupted. 2 The administrator has a backup save file.
Main post-condition(s)
1 The corrupted save file has been replaced with a new valid one
Additional Information
none

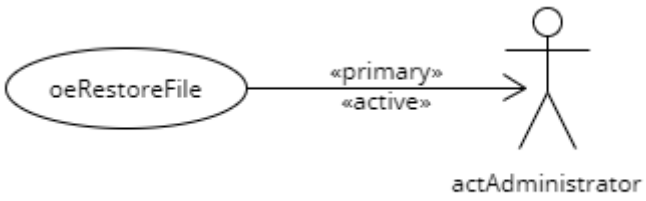

Figure 10: oeRestoreFile subfunction use case

2.3.2 Use Case Instance

2.3.2.1 Use-Case Instance - uciPlayOneGamePart01:suPlaySaveExit

The first part of the use case instance for the summary use case PlaySaveExit will illustrate a simple interaction scenario between two players, who will play the game for the first time.

SUMMARY Use-Case Instance
Instantiated Use Case suPlaySaveExit
Instance ID
uciPlayOneGamePart01
Remarks
Two players start the game. It is their first time playing, so they must create username. One player chooses the game mode to play. This player will be the first try to guess the function. In this case the first player won from the first try. They earn money (and achievements). The second player only receives a message they lost.

2.3.2.2 Use-Case Instance - uciPlayOneGamePart02:suPlaySaveExit

The second part of the use case instance for the summary use case suPlaySaveExit will illustrate what the players can do after playing one game.

SUMMARY Use-Case Instance
Instantiated Use Case suPlaySaveExit
Instance ID
uciPlayOneGamePart02
Remarks

The two players decide to save the game. With each save the system will send the signature of the save file to the administrator. The first player decides to exit the game. The second player can remain connected.

2.3.2.3 Use-Case Instance - uciLoadUnlockCheckBanPart01:suLoadUnlockCheckBan

The first part of the case instance will illustrate the summary use case of suLoadUnlockCheck-Ban. The illustration will show how the player can load their progress and check their achievements.The administrator will also perform integrity check on the save file.

SUMMARY Use-Case Instance
Instantiated Use Case suLoadUnlockCheckBan
Instance ID
uciLoadUnlockCheckBanPart01
Remarks
The first player wants to load game. The system requests the file signature from the administrator. The integrity check fails due to file corruption. The administrator will restore the save file with a backup file. The progress is loaded for the player and they check what achievements have been unlocked.

2.3.2.4 Use-Case Instance - uciLoadUnlockCheckBanPart02:suLoadUnlockCheckBan

The second part of the case instance will illustrate the summary use case of LoadUn-lockCheckBan. The illustration will show how the player can check the amount of money they have. The player can unlock a new game mode if they have enough money. The admin-istrator will ban a player due to cheating and erase their progress.

SUMMARY Use-Case Instance
Instantiated Use Case suLoadUnlockCheckBan
Instance ID
uciLoadUnlockCheckBanPart02
Remarks
The player check the amount of money. They have enough money to unlock a new game mode and do so. The administrator check the statistic of player. The administrator bans them due to cheating and erase the progress they made.

2.3.2.5 Use-Case Instance - uciugPlayOneGame

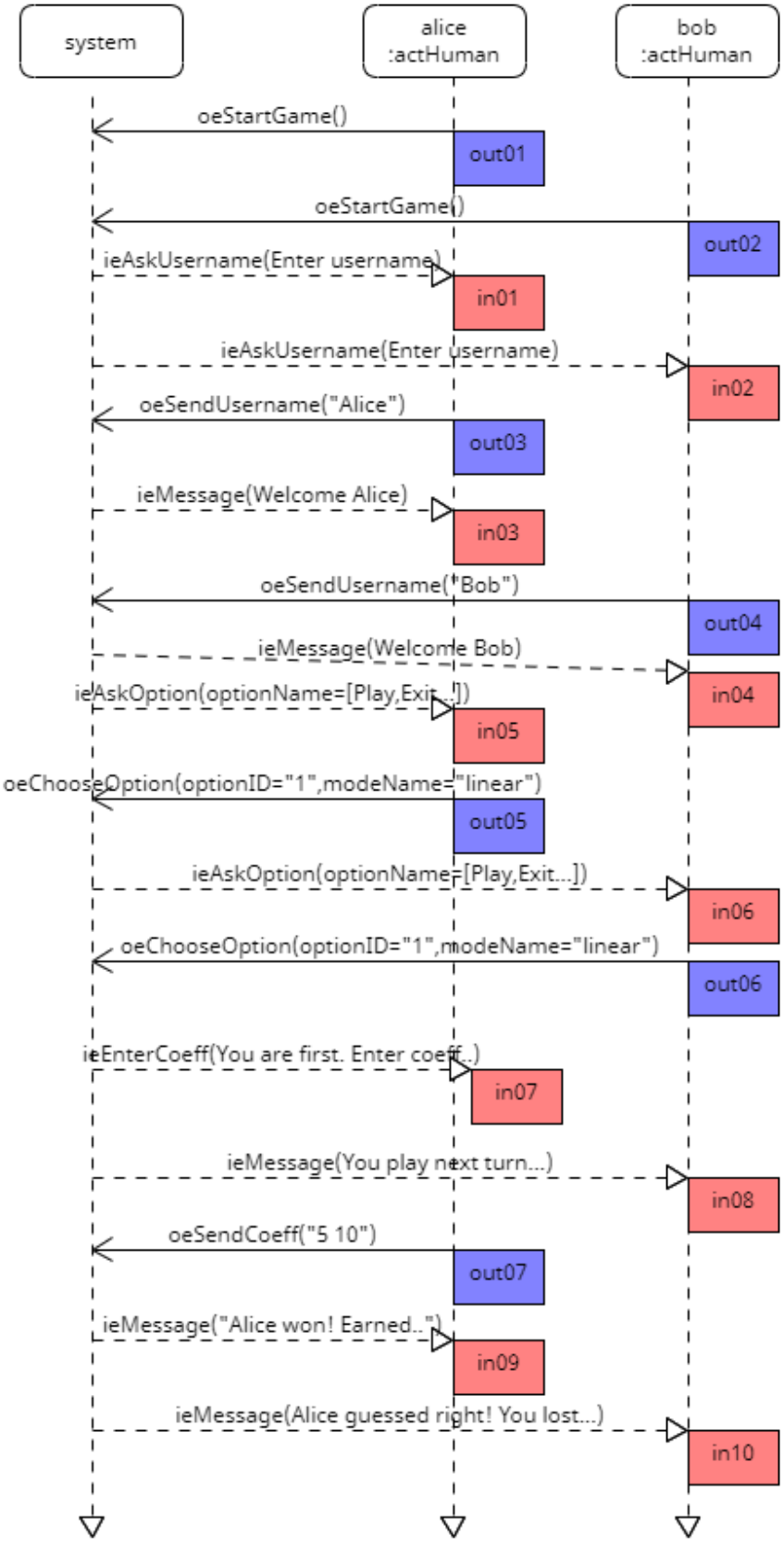

Figure 11: uci-suPlaySaveExit-Part01

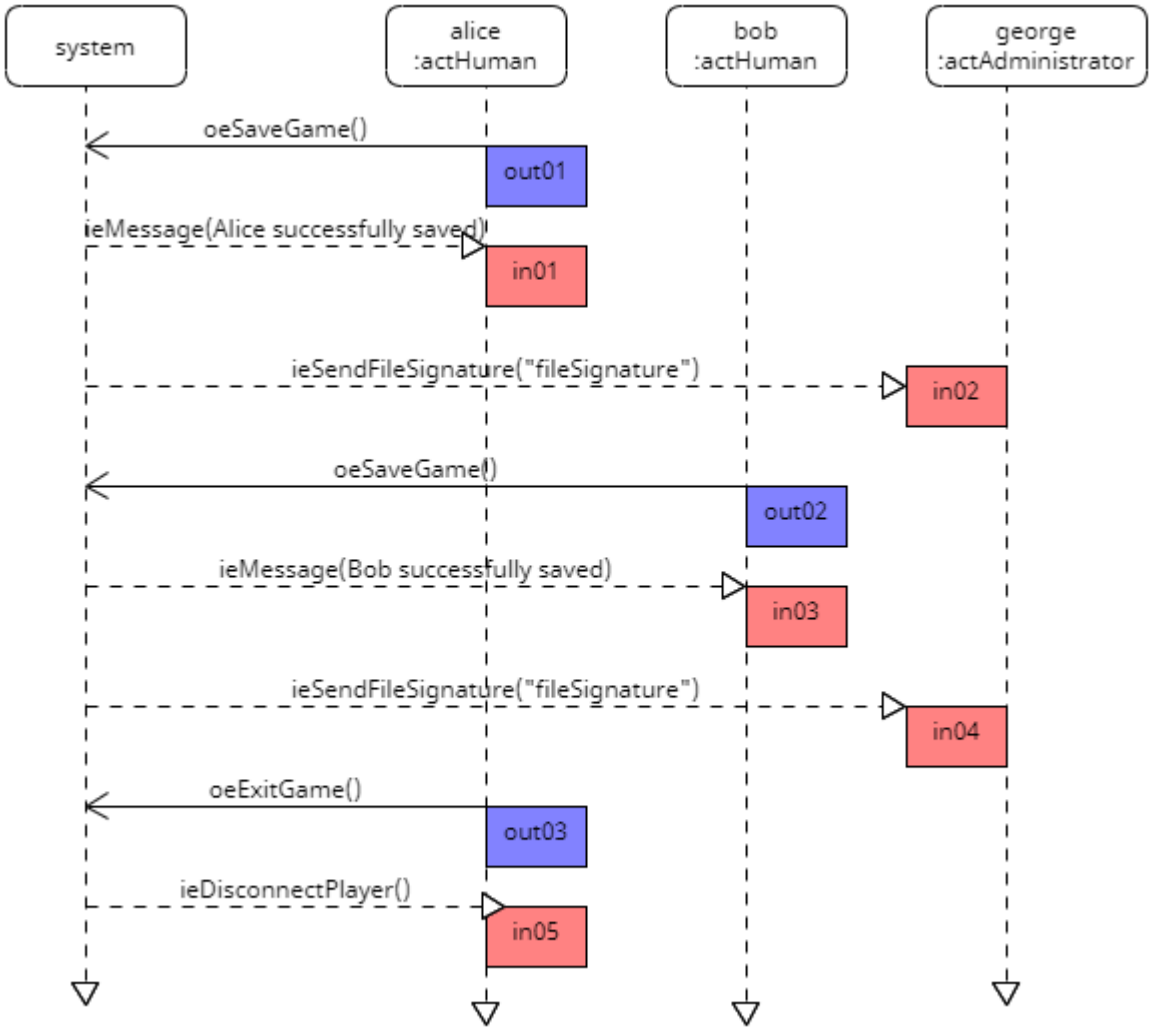

Figure 12: uci-suPlaySaveExit-Part02

2.3.2.6 Use-Case Instance - uciugLoadCheckUnlockBan

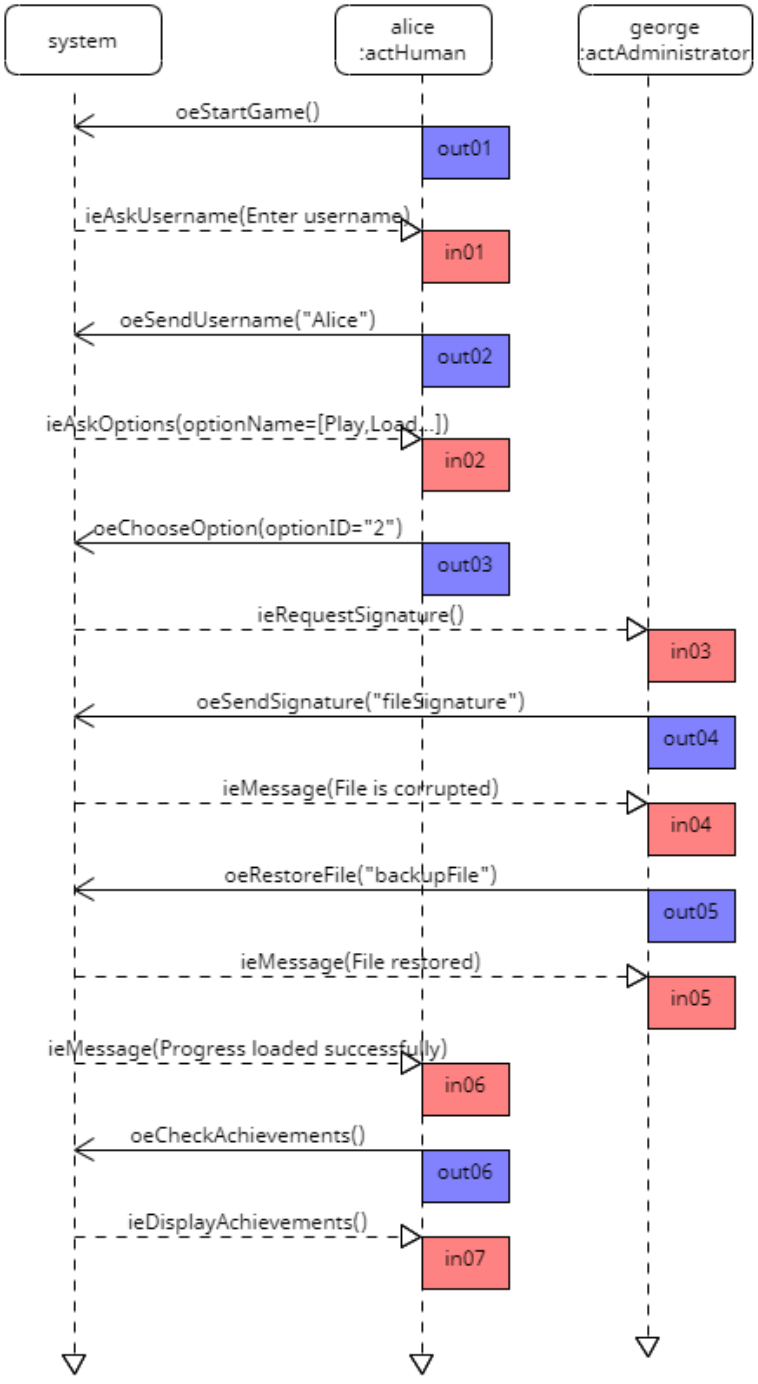

Figure 13: uci-uciugLoadCheckUnlockBan-Part01

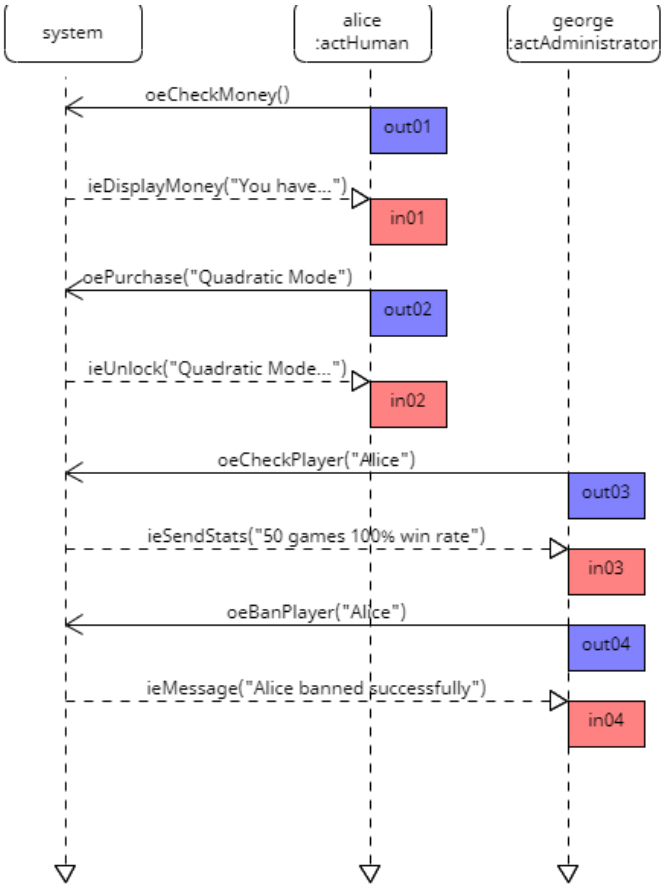

Figure 14: uci-uciugLoadCheckUnlockBan-Part02

3 Environment Model

3.1 Local View 01

Figure 15 shows the local view for the human actor and interfaces

Figure 15: Environment Model - Local View 01. environment model local view

3.2 Local View 02

Figure 16 shows the local view for the administrator actor and interfaces

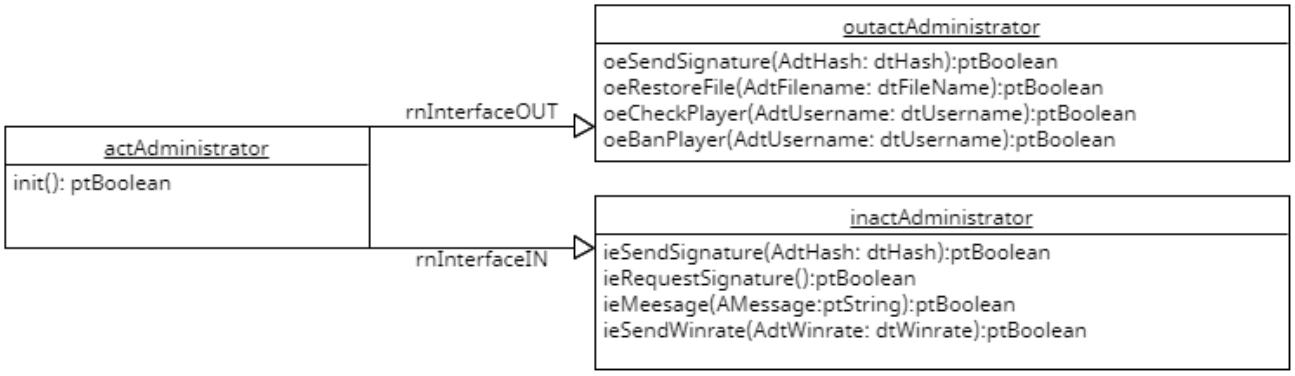

Figure 16: Environment Model - Local View 02. environment model local view

3.3 Global View 01

Figure 17 shows a global view for all actors with their relationships with ctState

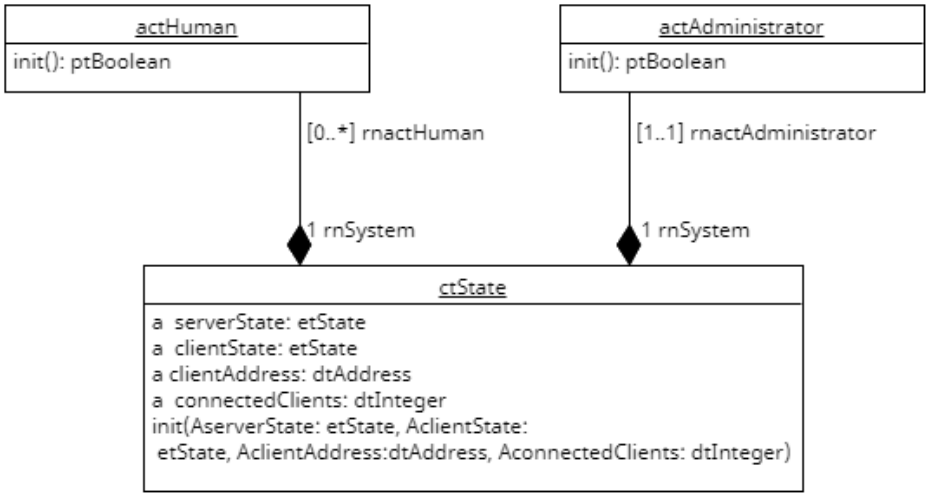

Figure 17: Environment Model - Global View 01. em-gv-01 environment model global view

3.4 Actors and Interface Description

We provide for the given views the description of the actors together with their associated input and output interface descriptions

3.4.1 actHuman Actor

Actor
actHuman represents an actor responsible of playing the game.
OutputInterfaces
OUT 1 **oeStartGame(): ptBoolean** *the player starts the game and connects to the server* OUT 2 **oeExitGame():ptBoolean** *the player stops the game and disconnects from the server* OUT 3 **oeSendUsername(AdtUsername: dtUsername):ptBoolean** *the players sends their username to the server* OUT 4 **oeSendCoeff(AdtCoefficients: dtCoefficients):ptBoolean** *the player sends the polynomial coefficients to the server* OUT 5 **oeChooseOption(AdtOption, etOption):ptBoolean** *the player sends the ID of the chosen option to the server* OUT 6 **oePurchase(AdtGameMode: etMode):ptBoolean** *the player can purchase one of the available game modes* OUT 7 **oeLoadGame(AdtSaveID: etSaveID):ptBoolean** *the player can load their saved progress*

OUT 8 **oeSaveGame():ptBoolea**n *the player can save their game progress* OUT 9 **oeCheckMoney():ptBoolean** *the player can check their in-game money* OUT 10 **oeCheckAchievements():ptBoolean** *the player can check their unlocked achievements* OUT 11 **oeDisplayLockedModes():ptBoolean** *the player can display the locked game modes*
InputInterface
IN 1 **ieAskUsername(): ptBoolean** *the server asks the player for their username* IN 2 **ieEnterCoeff(): ptBoolean** *the server asks the player to input their polynomial coefficients* IN 3 **ieMessage(AMessage: ptString):ptBoolean** *the server sends a message to the client* In 4 **ieAskOption(AdtOptions, etOptions):ptBoolean** *the server asks the player to choose an option* IN 5 **ieDisconnectPlayer():ptBoolean** *the server disconnects the player* IN 6 **ieDisplaySaveIDs(AdtSaveIDs: etSaveID):ptBoolean** *the server displays all save id(s) for the player* IN 7 **ieDisplayAchievements(AdtAchievments: dtAchievements):ptBoolean** *the server displays the achievements to the player* IN 8 **ieDisplayMoney(AdtMoney: dtMoney):ptBoolean** *the server displays the money to the player* In 9 **ieDisplayLockedModes(AdtModes: etModes):ptBoolean** *the server display the locked game modes to the player* IN 10 **ieUnlock(AdtUnlocked: dtunlocked):ptBoolean** *the server unlocks the purchased game mode for the player*

3.4.2 actAdministrator Actor

Actor
actAdministrator represents an actor responsible of administration tasks for the game.
OutputInterfaces
OUT 1 **oeSendSignature(AdtSignature: dtSignature):ptBoolean** *the admin sends the file signatue to the server* OUT 2 **oeRestoreFile(AdtFile: dtFile):ptBoolean** *the admin sends the backup file to the server*

OUT 3 **oeCheckPlayer(AdtUsername: dtUsername):ptBoolean** *the admin checks the statistic for a player* OUT 4 **oeBanPlayer(AdtUsername: dtUsername):ptBoolean** *the admin bans a player*
InputInterface
IN 1 **ieSendSignature(AdtSignature: dtSignature):ptBoolean** *the server sends a file signature to the admin* IN 2 **ieRequestSignature():ptBoolean** *the server requests a file signature from the admin* IN 3 **ieMeesage(AMessage:ptString):ptBoolean** *the server sends a message to the admin* IN 4 **ieSendStats(AdtStats: dtStats):ptBoolean** *the server sends player's statistics to the admin*

4 Concept Model

4.1 PrimaryTypes-Classes

4.1.1 Local view 01

Figure 18 shows the local view on all the primary types class types.

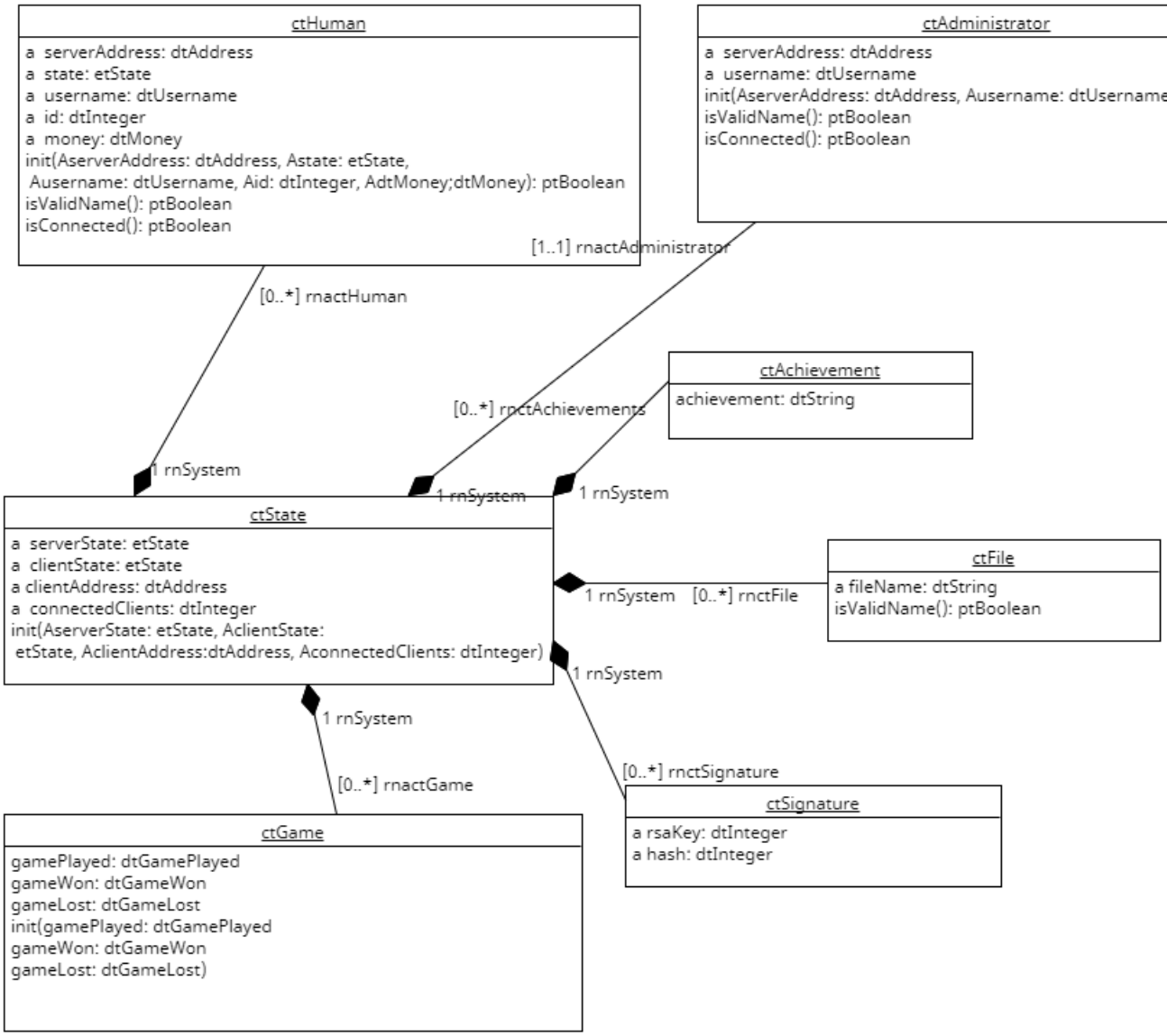

Figure 18: Concept Model - PrimaryTypes-Classes local view 01. Local view of all the primary types class types

4.1.2 Local view 02

Figure 19 shows the local view on the ctState primary types class types.

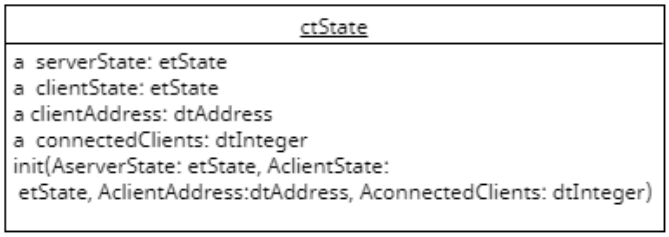

Figure 19: Concept Model - PrimaryTypes-Classes local view 02. local view of the ctState primary type.

4.1.3 Local view 03

Figure 20 shows the local view on the ctHuman primary types class types.

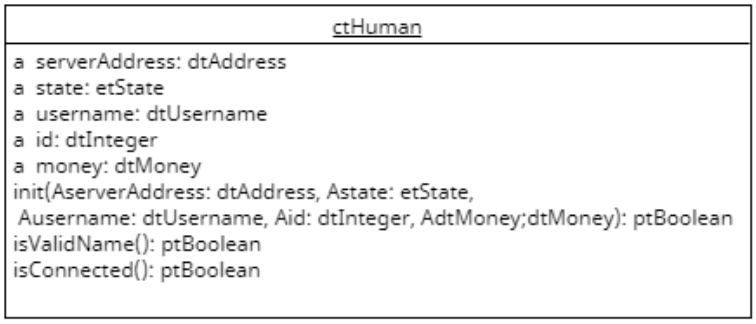

Figure 20: Concept Model - PrimaryTypes-Classes local view 03. local view of the ctHuman primary type.

4.1.4 Local view 04

Figure 21 shows the local view on the ctAdministrator primary types class types.

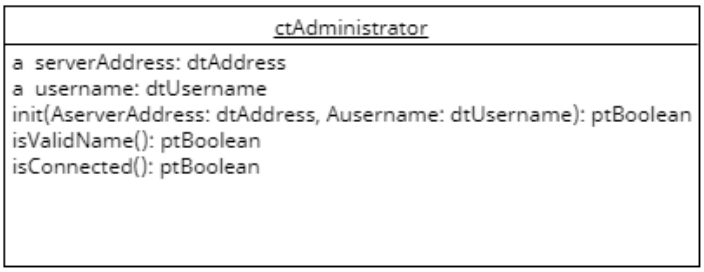

Figure 21: Concept Model - PrimaryTypes-Classes local view 04. local view of the ctAdministrator primary type.

4.1.5 Global view 05

Figure 22 shows the global view on the ctAdministrator associations.

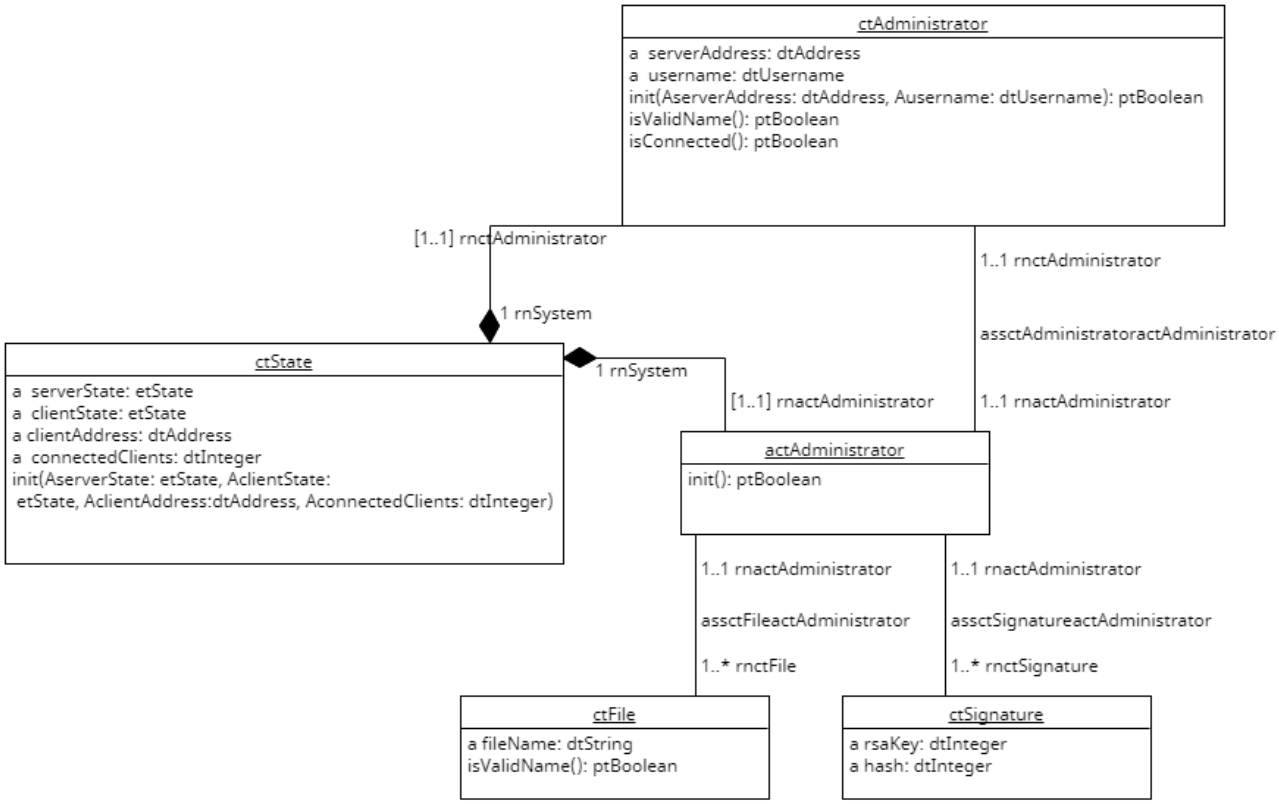

Figure 22: Concept Model - PrimaryTypes-Classes global view 01. Primary types class types global view.

4.1.6 Global view 06

Figure 23 shows the global view on the ctHuman associations.

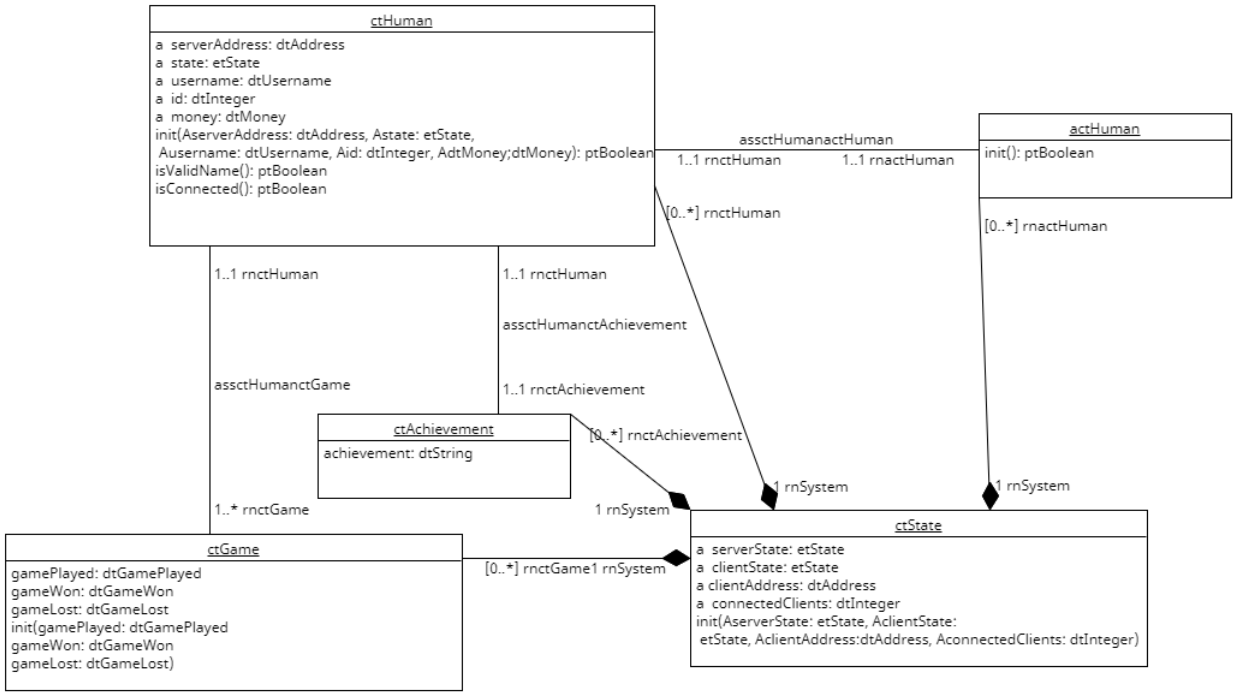

Figure 23: Concept Model - PrimaryTypes-Classes global view 02. Primary types class types global view.

4.2 PrimaryTypes-Datatypes

In this section, a subset of all datatypes will be presented as local views and rest can be seen in the global view

4.2.1 Local view 01

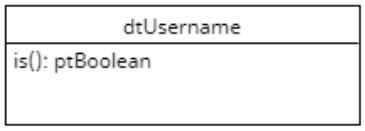

Figure 24: Concept Model - PrimaryTypes-Datatypes local view 01

4.2.2 Local view 02

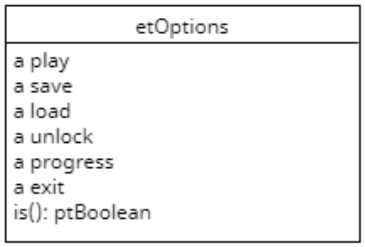

Figure 25: Concept Model - PrimaryTypes-Datatypes local view 02

4.2.3 Local view 03

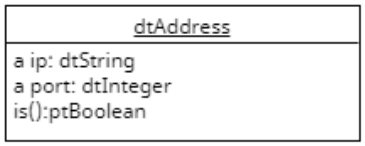

Figure 26: Concept Model - PrimaryTypes-Datatypes local view 03

4.2.4 Global view 01

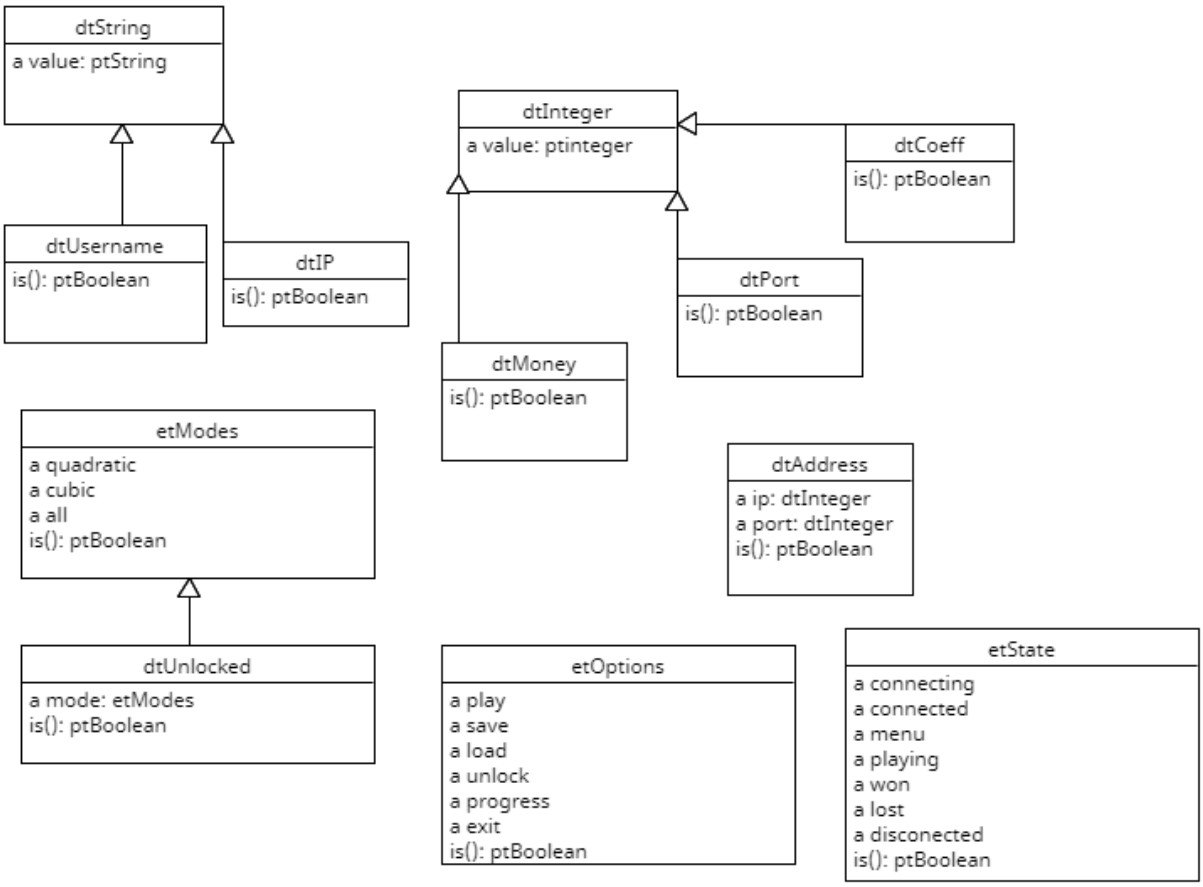

Figure 27: Concept Model - PrimaryTypes-Datatypes global view 01. global view of primary types datatype types

4.2.5 Global view 02

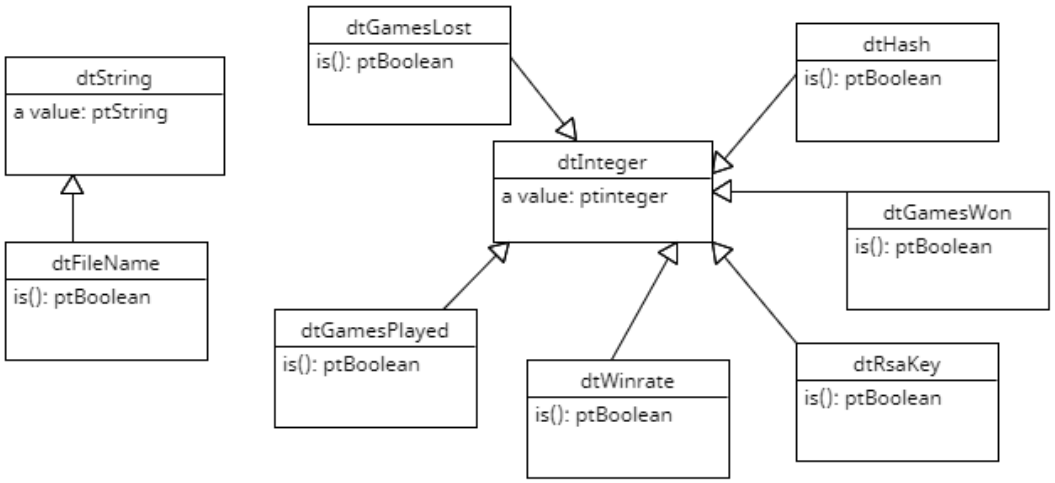

Figure 28: Concept Model - PrimaryTypes-Datatypes global view 02. global view of primary types datatype types1

4.3 SecondaryTypes - Datatypes

4.3.1 Local view 01

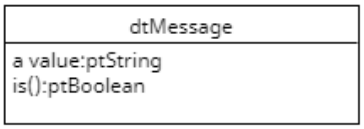

Figure 29: Concept Model - SecondaryTypes-Datatypes local view 01.

4.4 Concept Model Types Descriptions

This section provides the textual descriptions of all the types defined in the concept model and that can be part of the graphical views provided.

4.4.1 Primary types - Class types descriptions

Classes
ctHuman used to characterize the entity that is going to play the game
attribute **serverAddress:dtAddress** *the IP address and port of the server* attribute **state:etState**

based on the player's choices, they will be in one of the defined states
attribute **username:dtUsername**
a name to be used in the game. The name will also be used for save files
attribute **id:dtInteger**
a unique identifier for the player
attribute **money:dtMoney**
representation of the in-game money for a player
attribute **gamesPlayed:dtGamesPlayed**
representation of how many games were played by a player
attribute **gamesWon:dtGamesWon**
representation of how many games were won by a player
attribute **dtGamesLost:dtGamesLost**
representation of how many games were lost by a player
attribute **dtAchievement:etAchievement**
a text representing some achievement earned by a player
operation **init(AserverAddress: dtAddress, Astate: etState, Ausername: dtUsername, Aid: dtInteger, Amoney:dtInteger, Aachievement:etAchievement, AdtGamesPlayed:dtGamesPlayed,AdtGamesWon:dtGamesWon,dtGamesLost:dtGamesLost): ptBoolean**
used to initialize the current object as a new instance of the ctHuman type
operation **isValidName(): ptBoolean**
used to check if the username is a valid string
operation **isConnected(): ptBoolean**
used to check if the player is connected to a server

ctAdministrator
used to characterize the entity that is responsible for administrating the game

attribute **serverAddress:dtAddress**
the IP address and port of the server
attribute **username:dtUsername**
a name of the admin connected to the server
operation **init(AserverAddress: dtAddress, Ausername: dtUsername): ptBoolean**
used to initialize the current object as a new instance of the ctAdministrator type
operation **isValidName(): ptBoolean**
used to check if the username is a valid string
operation **isConnected(): ptBoolean**
used to check if the administrator is connected to a server

ctState
used to model the system.

attribute **serverState:etState**

used to represent the current state of the server. The server can have multiple states depending on the client connected to it attribute **clientState:etState** *the state of each client connected to the server. The client can be in only one state at a time* attribute **clientAddress: dtAddress** *the IP address and port used by the client to connect to the server* attribute **connectedClients: dtInteger** *used to represented the number of clients connected to the server* operation **init(AserverState: etState, AclientState: etState, AclientAddress:dtAddress, AconnectedClients: dtInteger): ptBoolean** *used to initialize the current object as a new instance of the ctState type*
ctFile used to model a collection of files.
attribute **fileName:dtString** *used to represent the name of a file* operation **isValidName():ptBoolean** *used to check if the file name is valid i.e. the file exists.*
ctSignature used to model a collection of file signatures.
attribute **rsaKey:dtInteger** *used to represent an RSA public and private key* attribute **hash:dtInteger** *used to represent a hash value*
ctGame used to model the game statistics of a user.
attribute **gamePlayed:dtGamePlayed** *used to represent the amount of games played* attribute **gameWon:dtGameWon** *used to represent amount of games won* attribute **gameLost:dtGameLost** *used to represent amount of games lost*

4.4.2 Primary types - Datatypes types descriptions

Datatypes
dtUsername A string used to identify a player or administrator
extends dtString operation **is():ptBoolean**

used to determine which strings are considered as valid usernames.
extends dtString operation **is():ptBoolean** used to determine which strings are considered as valid achievements.
dtMoney An integer used to represent the available in-game money for a player
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid amount of money.
dtAddress Used to define the IP address and port number.
attribute **ip: dtString** attribute **port: dtInteger** operation **is():ptBoolean** used to determine which values are considered as valid addresses.
dtIP A string used to represent an IP address.
extends dtString operation **is():ptBoolean** used to determine which strings are considered as valid IP addresses.
dtPort An integer used to represent a port number
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid port numbers.
dtCoeff Polynomial coefficient, which is a natural number
extends dtInteger operation **is():ptBoolean** used to determine which numbers are considered as a valid coefficient.
dtGamesPlayed An integer used to represent the amount of games played by a player.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid amounts of games played.
dtGamesWon An integer used to represent the amount of games won by a player.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid amounts of games won.

dtGamesLost
An integer used to represent the amount of games lost by a player.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid amounts of games lost.
dtWinrate
An integer used to represent an the winrate of a player.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid winrates.
dtHash
An integer used to represent an hash value.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid hash values.
dtRsaKey
An integer used to represent a RSA key.
extends dtInteger operation **is():ptBoolean** used to determine which integers are considered as valid key values.
dtFileName
A string used to represent the name of a file.
extends dtString operation **is():ptBoolean** used to determine which strings are considered as valid file names.

Enumerations
etModes
this type is used to indicate the different game modes
operation **is():ptBoolean**
used to determine which literal belongs to the enumeration.
etOptions
this type is used to indicate the different menu options
operation **is():ptBoolean**
used to determine which literal belongs to the enumeration.
etStates
this type is used to indicate the different states of a client/server
operation **is():ptBoolean**
used to determine which literal belongs to the enumeration.

etAchievement this type is used to indicate the different achievements a player can earn
operation **is():ptBoolean** used to determine which literal belongs to the enumeration.

4.4.3 Primary types - Assocation types descriptions

Associations
assctHumanactHuman A human, who will be playing the game.
assctAdministratoractAdministator The game needs one administrator, who is responsible for integrity of save files and banning cheating players.
assctFileactAdministrator The administrator can have multiple files in his collection of files.
assctSignatureactAdministrator The administrator can have multiple file signatures in his collection.

4.4.4 Primary types - Aggregation types descriptions

There are no aggregation types for the primary types.

4.4.5 Primary types - Composition types descriptions

There are no composition types for the primary types.

4.4.6 Secondary types - Class types descriptions

There are no elements in this category in the system analysed

4.4.7 Secondary types - Datatypes types descriptions

Datatypes
dtMessage A datatype made of a string value used to send textual information to client/server
attribute **value: ptString** operation **is():ptBoolean** used to determine which strings are considered as valid messages.

4.4.8 Secondary types - Association types descriptions

There are no association types for the secondary types.

4.4.9 Secondary types - Aggregation types descriptions

There are no aggregation types for the secondary types.

4.4.10 Secondary types - Composition types descriptions

There are no composition types for the secondary types.

5 Operation Model

5.1 Environment - Out Interface Operation Scheme for actHuman

5.1.1 Operation Model for oeSendUsername

Operation
oeSendUsername The user sends their username to the system
Parameters
AdtUsername:dtUsername a name chosen by the user
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started the game
Pre-condition (functional)
PreF 1 The username is a valid name
Post-condition (functional)
PostF 1 The username is sent to the server PostF 2 A welcome message is sent to the user
Post-condition (protocol)
PostP 1 The user has access to the menu

```
context actHuman: oeSendUsername(AdtUsername:dtUsername):ptBoolean
 # Pre Protocol
 preP{
  let TheState:ctState in
  let ClientState:etState in
  self.rnActor.rnSystem.clientState = ClientState
  and ClientState = Connected
 }
# Pre Functional
 preF{
  let TheHuman:ctHuman in
```

```
 let Username:dtUsername in
 self.TheHuman.Username.size() >= 3 and
 self.TheHuman.Username.size() <= 10
}
# Post Functional
postF{
 let TheHuman:ctHuman in
 let Message: ptString in
 let usernameSent:ptBoolean in usernameSent=self.TheHuman.
   oeSendUsername().hasReturned()
 and usernameSent = true
 and "Welcome" = Message.value
 and self.rnActor.InterfaceIN@post^ieMessage(Message)
}
# Post Protocol
postP{
       let TheHuman:ctHuman in
       self.TheHuman.state = Menu
}
```

5.1.2 Operation Model for oeChooseOption

Operation
oeChooseOption The user is asked to choose from a set of options.
Parameters
AdtOption:etOption one of the available options
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started the game and entered their username Prep 2 The option "load" is available only if there exists a non-empty save file.
Pre-condition (functional)
PreF 1 An option from etOption has been provided
Post-condition (functional)
PostF 1 If the option is "play", the user is sent a list of modes specified in etMode. PostF 2 If the option is "save", the user is sent a message, which indicates the save was successful. PostF 3 If the option is "load", the user is sent a message, which indicates the load was successful. PostF 4 If the option is "unlock", the user is sent a list of game modes. PostF 5 If the option is "progress", the user is sent a list of progress attributes from the etProgressAttributes. PostF 6 if the option is "exit", that instance of ctHuman is removed from the system state.
Post-condition (protocol)
PostP 1 If the previous choice was "play", game modes become available to be played. PostP 2 If the previous choice was"save", the "load" option becomes available.

```
context actHuman: oeChooseOption(AdtOption:etOption):ptBoolean
 # Pre Protocol
 preP{
  let TheState:ctState in
  let TheHuman:ctHuman in
  let ClientState:etState in
  let usernameSent:ptBoolean in
  usernameSent = self.TheHuman.oeSendUsername().hasReturned()
  and usernameSent = true
  and self.rnActor.rnSystem.clientState = ClientState
  and ClientState = Connected
 }

```

```
preF{
 let Option:etOption in
 self.Option -> forAll(e | e.isValid())
}

postF{
 let TheHuman:ctHuman in
 let GameModes: etMode in
 let Message: ptString in
 let Option: etOption in

 # The ieSendModes is defined for the first time here.
 # The ieSendProgressAttributes is defined for the first time here.

 if option = Play then self.rnActor.InterfaceIN@post^ieSendModes(
   AdtMode:etMode)

 if option = Save then "Save Successful" = Message.value and self.
   rnActor.InterfaceIN@post^ieMessage(Message)

 if option = Load then "Load Successful" = Message.value and self.
   rnActor.InterfaceIN@post^ieMessage(Message)

 if option = Unlock then self.rnActor.InterfaceIN@post^ieSendModes(
   AdtMode:etMode)

 if option = Progress then self.rnActor.InterfaceIN@post^
   ieSendProgressAttributes(AdtProgress:etProgress)

 # The ctHuman is killed.
 if option = Exit then self.TheHuman.state = disconected

 endif


}
postP{
        let TheHuman:ctHuman in
        let Option:etOption in
        # PostP 1
        if Option = Play then self.TheHuman.state = Playing
        # PostP 2
        if Option = Save then self.Option.load.isValid() = true

        endif


}
```

5.1.3 Operation Model for oeSendCoeff

Operation
oeSendCoeff The user sends a polynomial coefficient to the system
Parameters
AdtCoeff:dtCoeff a polynomial coefficient
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started playing one of the game modes
Pre-condition (functional)
PreF 1 If the user is playing "linear game mode", they have to put 2 valid polynomial coefficients. PreF 2 If the user is playing "quadratic game mode", they have to put 3 valid polynomial coefficients. PreF 3 If the user is playing "cubic game mode", they have to put 4 valid polynomial coefficients.
Post-condition (functional)
PostF 1 Each valid polynomial coefficient has been sent. PostF 2 If the user guessed correctly, their state of the ctHuman is changed from "playing" to "won". PostF 3 If the user guessed correctly, they receive in-game money and an achievement(s) if they are eligible. After they receive everything the state of that ctHuman is changed from "won" to "menu". PostF 4 If the user guessed incorrectly, they receive a message, indicating they have to wait for their turn. PostF 5 The user, who lost, their ctHuman state is changed from "playing" to "lost". After this their state is changed from "lost" to "menu".
Post-condition (protocol)
PostP 1 If the user guessed incorrectly, they cannot enter new coefficients until it is their turn. PostP 3 The second user is able to enter coefficients.

```
context actHuman: oeSendCoeff(AdtCoeff:dtCoeff):ptBoolean
 # Pre Protocol
 preP{
  let TheState:ctState in
  let ClientState:etState in
  and self.rnActor.rnSystem.clientState = ClientState
```

```
 and ClientState = Playing
}

preF{
 let TheHuman:ctHuman in
 let GameMode:etMode in
 let firstCoeff:dtCoeff in
 let secondCoeff:dtCoeff in
 let thirdCoeff:dtCoeff in
 let fourthCoeff:dtCoeff in


 if GameMode = Linear then firstCoefficient >= -100
          and firstCoefficient <= 100
          and secondCoefficient >= -100
          and secondCoefficient <= 100

 if GameMode = Quadtratic then firstCoefficient >= -100
          and firstCoefficient <= 100
          and secondCoefficient >= -100
          and secondCoefficient <= 100
          and thirdCoefficient >= -100
          and thirdCoefficient <= 100

 if GameMode = Cubic then firstCoefficient >= -100
          and firstCoefficient <= 100
          and secondCoefficient >= -100
          and secondCoefficient <= 100
          and thirdCoefficient >= -100
          and thirdCoefficient <= 100
          and fourthCoefficient >= -100
          and fourthCoefficient <= 100
}


postF{
 let TheHuman:ctHuman in
 let Message: ptString in
 let Achievement:ctAchievement

 Message = self.rnActor.InterfaceIN@pre^ieMessage(Message)

 if Message.value = "You won" then self.TheHuman.state = Won

 if self.TheHuman.state = Won then self.TheHuman.money + 100 and
   Achievement.init(achievement:dtString)
 self.TheHuman.state = Menu

```

```
54   if Message.value = "You guessed incorrectly" then Message.value = "
       Wait for your turn" and  Message = self.rnActor.InterfaceIN@post^
       ieMessage(Message)
55
56
57   if Message.value = "You lost" then self.TheHuman.state = Lost
58   if self.TheHuman.state = Lost then self.TheHuman.state = Menu
59
60   endif
61
62
63 }
64 postP{
65         let TheHuman:ctHuman in
66         let waitingState = Waiting in
67         if self.TheHuman.state != waitingState then self.TheHuman.
       oeSendCoeff().isValid() = true
68         else self.TheHuman.oeSendCoeff().isValid() = false and self.
       TheHuman,state = waitingState
69
70
71
72 }
```

5.1.4 Operation Model for oePurchaseMode

Operation
oePurchaseMode The user can purchase a game mode with in-game money
Parameters
AdtMode:etMode one of the available game modes
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has entered the "unlock" option.
Pre-condition (functional)
PreF 1 The user has entered a valid mode.
Post-condition (functional)
PostF 1 If the user picked "quadratic" mode and had enough money, the price of the mode is subtracted from the money of the user. PostF 2 If the user picked "cubic" mode and had enough money, the price of the mode is subtracted from the money of the user. PostF 3 The user receives a message that they purchased the mode.

PostF 4 If the player does not have enough money, they get error message.
Post-condition (protocol)
PostP 1 The purchased mode is available in the "play" section. PostP 2 The purchased mode is no longer available in the "unlock" section.

```
context actHuman: oePurchaseMode(AdtMode:etMode):ptBoolean
 # Pre Protocol
 preP{
  let TheHuman:ctHuman in
  let Option:etOption in
  self.TheHuman.oeChooseOption(Option) = Unlock
 }

 preF{
  let TheHuman:ctHuman in
  let Mode:etMode in
  self.Mode -> forAll(e | e.isValid())
 }

 postF{
  let TheHuman:ctHuman in
  let GameMode: etMode in
  let Message: ptString in

  Message = "Purchase Successfull"

  if GameMode = Quadratic and self.TheHuman.money >= 200 then self.
    TheHuman.money - 200 and self.rnActor.InterfaceIN@post^ieMessage(
    Message.value)

  if GameMode = Cubic and self.TheHuman.money >= 300 then self.
    TheHuman.money - 300 and self.rnActor.InterfaceIN@post^ieMessage(
    Message.value)

  else Message = "Not enough money" and and self.rnActor.
    InterfaceIN@post^ieMessage(Message.value)
  endif


 }
 postP{
        let TheHuman:ctHuman in
        let Mode:etMode in
        let Option:etOption in
        let purchasedMode = Mode in
        if Option = Play then purchasedMode.isValid() = true
        if Option = Unlock then purchasedMode.isValid() = false
 }
```

5.1.5 Operation Model for oeSaveGame

Operation
oeSaveGame The user saves their game progress
Parameters
n/a
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started the game and entered their username
Pre-condition (functional)
PreF 1 The user has entered the correct option for saving a game
Post-condition (functional)
PostF 1 The system has created a csv file for the user, which contains the fields (name,money,games played, games won, games lost, win-rate,achievements, unlocked modes). PostF 2 Public and private RSA key is generated and saved in a pem file. PostF 3 This signature of the csv file has been created and saved in a file.
Post-condition (protocol)
n/a

```
context actHuman: oeSaveGame():ptBoolean
 # Pre Protocol
 preP{
  let TheState:ctState in
  let TheHuman:ctHuman in
  let ClientState:etState in
  let usernameSent = oeSendUsername().hasReturned() in
  let self.TheHuman = usernameSent
  and usernameSent = True
  and self.rnActor.rnSystem = ClientState
  and ClientState = Connected

 preF{
  let TheHuman:ctHuman in
  let Option:etOption in
  self.TheHuman.oeChooseOption(Option) = Save
 }

 postF{
  let TheHuman:ctHuman in
  let Username:dtUsername
  let File: ctFile in
```

```
 let saveFile:dtFileName in
 let publicKey:dtFileName in
 let privateKey:dtFileName in
 let signatureFile:dtFileName
 let digitalSignature:ctSignature in

 self.File.saveFile = self.TheHuman.Username.concat("_save.csv")
 self.File.publicKey = self.TheHuman.Username.concat("_public.pem")
 self.File.privateKey = self.TheHuman.Username.concat("_private.pem")
 # Byte file
 self.File.digitalSignature = self.TheHuman.Username.concat("
   signature")
}
postP{
 true
}
```

5.1.6 Operation Model for oeLoadGame

Operation
oeLoadGame The user can load their game progress
Parameters
n/a
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started the game and entered their username. PreP 2 A save file, which is not empty, exists.
Pre-condition (functional)
PreF 1 The user has entered the correct option for loading a game.
Post-condition (functional)
PostF 1 Name has been loaded. PostF 2 Money has been loaded. PostF 3 Achievements have been loaded. PostF 4 Unlocked modes have been loaded. PostF 5 Games Played have been loaded. PostF 6 Games Won have been loaded. PostF 7 Games Lost have been loaded.
Post-condition (protocol)
n/a

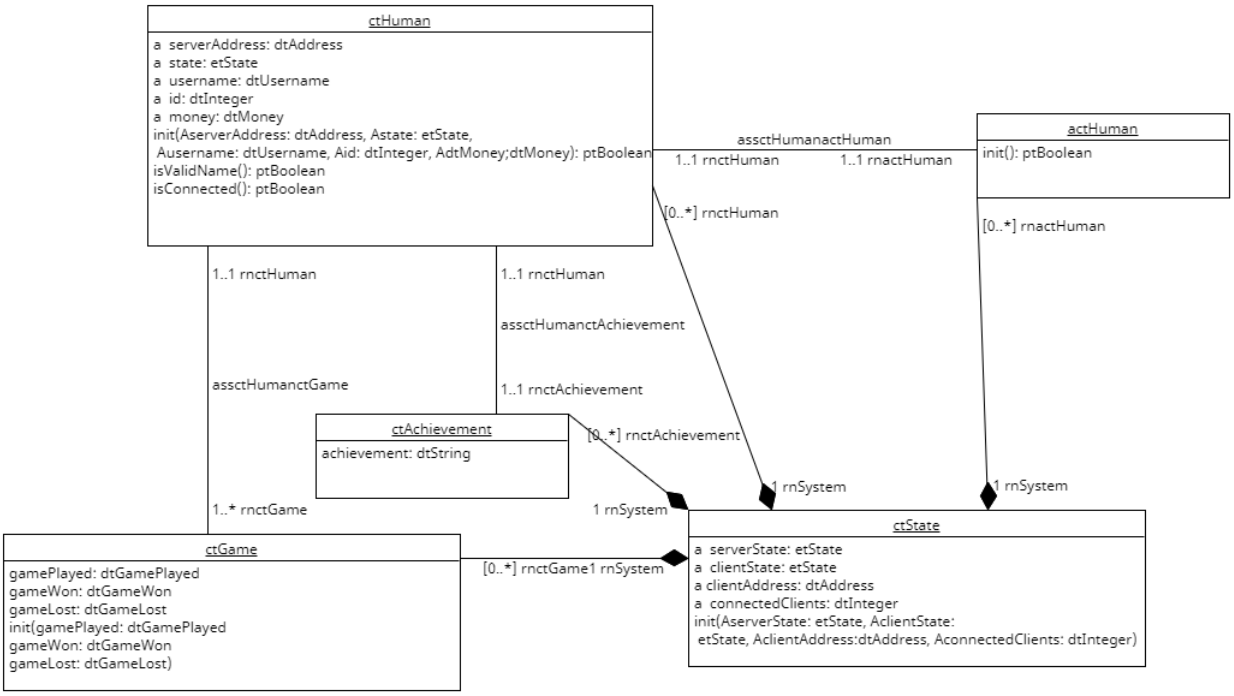

Figure 30: operation scope

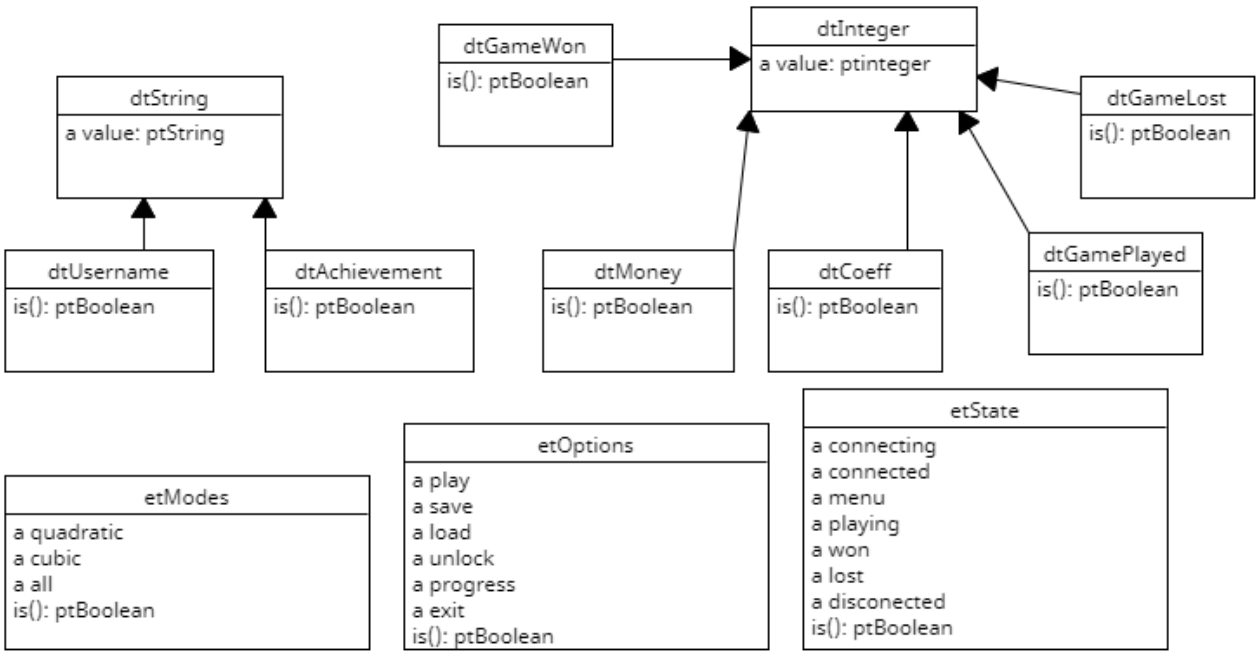

Figure 31: operation scope

5.1.7 Operation Model for oeCheckMoney

Operation
oeCheckMoney The user can check the amount of money they posses
Parameters
n/a
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has entered the the option "progress".
Pre-condition (functional)
PreF 1 The user has entered the entered "money" from the etProgressAttributes.
Post-condition (functional)
PostF 1 The amount of money is sent to the user.
Post-condition (protocol)
n/a

```
context actHuman: oeCheckMoney():ptBoolean
 # Pre Protocol
 preP{
 let TheHuman:ctHuman in
  let Option:etOption in
  Option = Progress

 preF{
  let TheHuman:ctHuman in
  let Option:etProgressAttributes in
  Option = Money
 }

 postF{
  let TheHuman:ctHuman in
  let Money:dtMoney in
  result = self.TheHuman.Money
 }
 postP{

  true
 }
```

5.1.8 Operation Model for oeCheckAchievement

Operation
oeCheckAchievement The user can check their earned achievements
Parameters
n/a
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has entered the option "progress".
Pre-condition (functional)
PreF 1 The user has entered the "achievements" from the etProgressAttributes.
Post-condition (functional)
PostF 1 The earned achievements are sent to the user PostF 2 If the player does not have achievements display nothing
Post-condition (protocol)
n/a

```
context actHuman: oeCheckAchievement():ptBoolean
 # Pre Protocol
 preP{
 let TheHuman:ctHuman in
  let Option:etOption in
  Option = Progress

 preF{
  let TheHuman:ctHuman in
  let Option:etProgressAttributes in
  Option = Achievements
 }

 postF{
  let TheHuman:ctHuman in
  let Achievement:ctAchievement in
  if self.Achievement.achievement <> null then result = self.
    Achievement.achievement
  else self.Achievement.achievement = "Empty"
 }
 postP{
  true
 }
```

5.1.9 Operation Model for oeExit

Operation
oeExit The user can exit the game
Parameters
n/a
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The user has started the game and entered their username.
Pre-condition (functional)
PreF 1 The user has entered the correct option for exiting the game.
Post-condition (functional)
PostF 1 That instance of ctHuman is removed from the system.
Post-condition (protocol)
PostP 1 The game is closed

5.2 Environment - Out Interface Operation Scheme for actAdministrator

5.2.1 Operation Model for oeSendSignature

Operation
oeSendSignature The administrator sends a file signature to the system
Parameters
AdtHash:dtHash a digital signature
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The administrator is connected to the system. PreP 2 A request for sending the signature has been received.
Pre-condition (functional)
PreF 1 A valid hash is entered.
Post-condition (functional)
PostF 1 A digital signature has been successfully sent to the system
Post-condition (protocol)
n/a

```
context actAdministrator: oeSendSignature(AdtHash:dtHash):ptBoolean
 # Pre Protocol
 preP{
  let TheAdmin:ctAdministrator in
  let State:etState in
  self.TheAdministrator.State = Connected
  and self.rnActor.InterfaceIN@pre^ieRequestSignature()

 preF{
  let TheAdmin:ctAdministrator in
  let Signature:ctSignature in
  self.Signature -> forAll(hash | Signature.hash <> null)
  and self.Signature -> forAll(rsaKey | Signature.rsaKey <> null)
 }

 postF{
  let TheAdmin:ctAdmin in
  let digitalSignature:dtHash
  let hashSent = self.TheAdmin.oeSendSignature(digitalSignature).
    hasReturned() and
  hashSent = true
 }
 postP{
```

```
23
24   true
25 }
```

5.2.2 Operation Model for oeRestoreFile

Operation
oeRestoreFile The administrator restores a corrupted save file.
Parameters
AdtFileName:dtFileName the name of the backup file.
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The administrator is connected to the system. PreP 2 The system has indicated the current save file is corrupted.
Pre-condition (functional)
PreF 1 A valid file name has been provided
Post-condition (functional)
PostF 1 The backup file is sent to the system. PostF 2 The corrupted save file is deleted.
Post-condition (protocol)
n/a

5.2.3 Operation Model for oeBanPlayer

Operation
oeBanPlayer The administrator bans a player
Parameters
AdtUsername:dtUsername the name of the player to be banned
Return type
ptBoolean
Pre-condition (protocol)
PreP 1 The administrator is connected to the system.
Pre-condition (functional)
PreF 1 A valid username, which exists in the system, has been entered

Post-condition (functional)
PostF 1 The player's whole progress in the csv file is erased.
Post-condition (protocol)
n/a

5.2.4 Operation Model for oeCheckPlayer

Operation
oeCheckPlayer The administrator views the progress of a player
Parameters
AdtUsername:dtUsername the name of the player
Return type
ptBoolean
Pre-condition (protocol)
Prep 1 The administrator is connected to the system.
Pre-condition (functional)
PreF 1 A valid username is provided, which also exists in the system.
Post-condition (functional)
PostF 1 The win-rate is sent to the administrator. PostF 2 The games played won is sent to the administrator. PostF 3 The games lost is sent to the administrator. PostF 4 The games played is sent to the administrator. PostF 5 The money is sent to the administrator.
Post-condition (protocol)
n/a

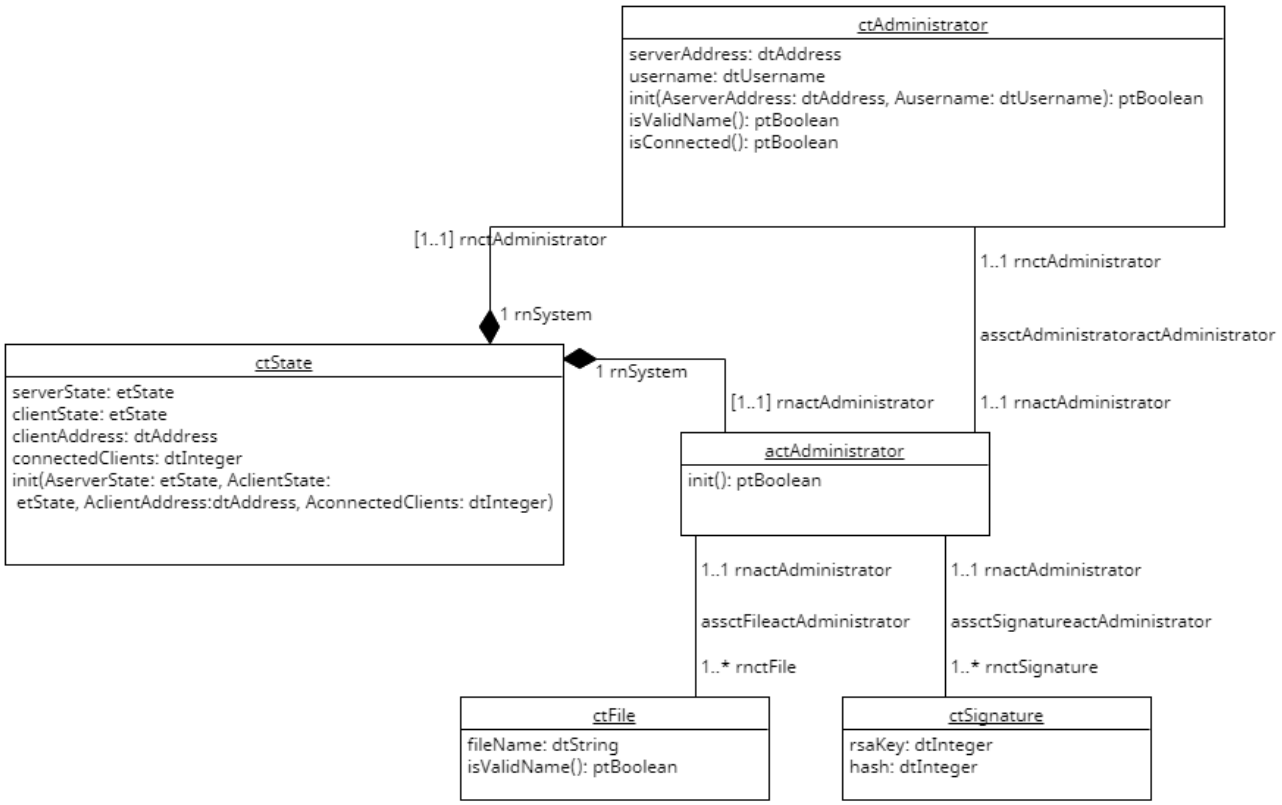

Figure 32: operation scope

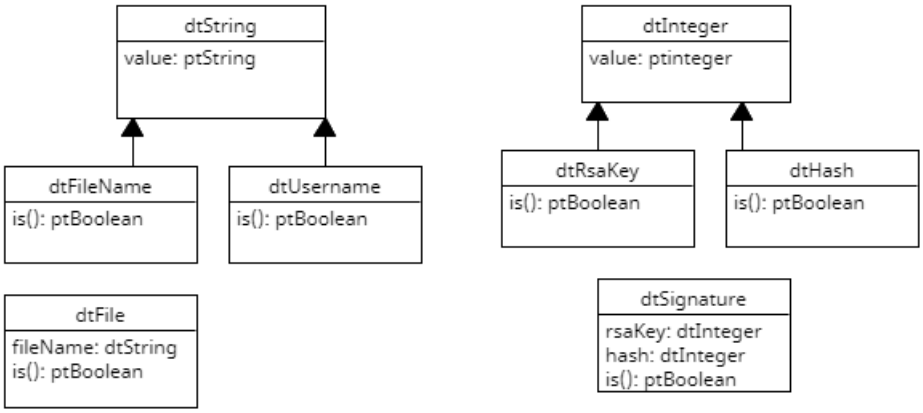

Figure 33: operation scope