INF-5150 Oblig 2 Report

SMSWar

November 26th, 2007

Group 3:

Bjørn Brændshøi

Bjørn Tveter

Kestutis Mintauckis

Petter Risholm

Vincent Demeester

Introduction

This report provides a detailed explanation of the specification and implementation of the obligatory exercise number 2 for the course INF-5150, Unassailable IT-systems, autumn 2007 carried out by group number 3. The security analysis with CORAS diagrams is merged with this document on page 33 and onwards.

Obligatory exercise 2 required that the groups developed a SMS game (Oblig2, 2007) where people can play a "war" game by sending SMS messages to a central server which is able to localize the players through the use of PATS. In short, the requirements of the game were that a user was given a certain number of points and certain strength at the beginning of the game. During the game, players can execute commands by sending SMS's to the system to locate other players, strike other players and increase their own strength (and more).

The report is divided into four major parts where each covers one aspect of the system design and implementation. The two first sections handle the design of the system while the last two focuses on the implementation and functionality. First we present the structure of the system in a composite structure diagram, followed by the behavior of the system with interactions (sequence diagrams) and state machines. Finally, we also present the signals that are used in the system. The last section contains a user guide for the system.

Functional Requirements

The functional requirements for mandatory exercise 2 are stated in (Oblig2, 2007). We have identified one actor, the player. Our system, SMSWar, provides seven functions that make up the game. Our system will only allow one game at a time, and the game can be instantiated by any registered user. This means that we do not define any specific game admin; any user can execute the functionality:

- Announce a game.

Once a game is running, the players have access to the following functionalities:

- Light up an area.
- Increase his strength.
- Strike another player.

General functionalities:

- Register user.
- A KML file for visualization in Google Earth is made automatically by the system while playing. This file could e.g. be placed on a server for the players too download.

Diagram 1 expresses the functional requirements of the system in terms of a use case.

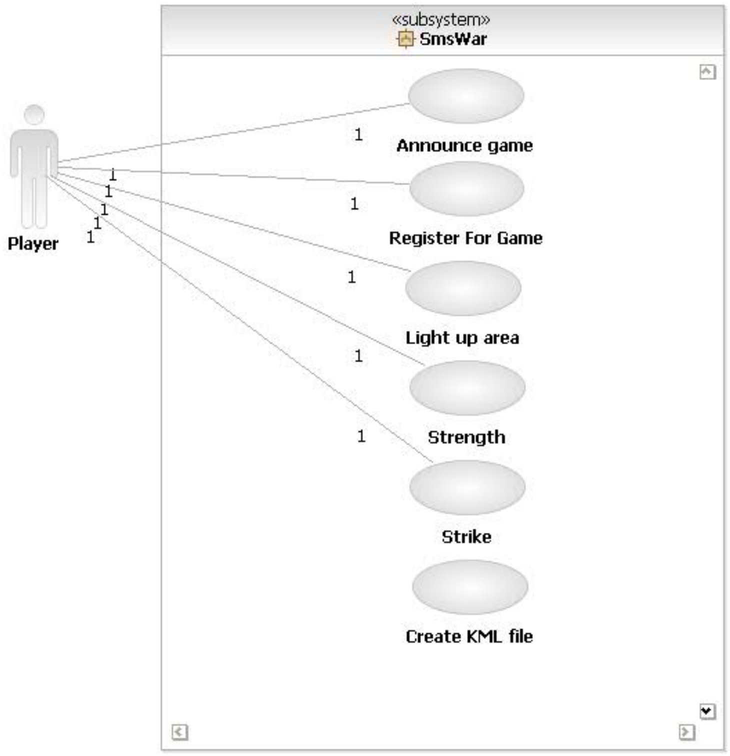


Diagram 1 - The use case diagram

SMSWar System Structure

Diagram 2 shows the classes in the system.

- **SmsWarComp** is the composite structure of our system.
- **SRM_ToUser** is a mediator used to route signals to specific instances of a statemachine.
- **User** and **UserStatus** are two classes used internally in the system.

Diagram 3 shows the composite structure of the developed system. It contains three parts, a main controller, a user controller and a user part and the connections between them. The three parts are all statemachines.

Part - Main Controller (Statemachine)

The main functionality of the main controller is to parse incoming SMS messages and pass them on as appropriate signals to the user controller. Main controller also has these responsibilities (use cases in parantheses):

- Register a new user (“Register for game”)
- Announcing a game (“Announce game”)
- Starting a game.

- Finishing a game.
- Declaring a winner.

It has two incoming ports, **smsIn** and **input** which are incoming SMS's from PATS (users) and incoming messages from the user controller respectively. It also has two corresponding outgoing ports, **smsOut** to send SMSs to the users and **toUserCont** that routes signals to the user controller.

Part - User Controller (Statemachine)

The user controller is responsible for signal distribution within the system. All signals to and from the instances of the User statemachine is routed via the user controller. The user controller uses a SimpleIdRouterMediator (SRM_ToUser) to route signals to single instances of the user statemachine and a MultiCastMediator to route signals to *all* user statemachine instances.

The user controller governs four ports, **toMain** for sending signals to the main controller, **input** for receiving signals from the user statemachines and the main controller as well as the two outgoing ports, **toUser** and **toAllUsers**, that are connected to the user statemachine.

Part - User [*] (Statemachine)

A user statemachine is created for every user that joins a starting game. When there is no active game, there are no instances of the user statemachine. All commands executed by a user via her cell phone, are routed to the user's corresponding statemachine through the MainController and UserController. The main responsibility of the User part is to keep track of a user's current status such as points, strength and position.

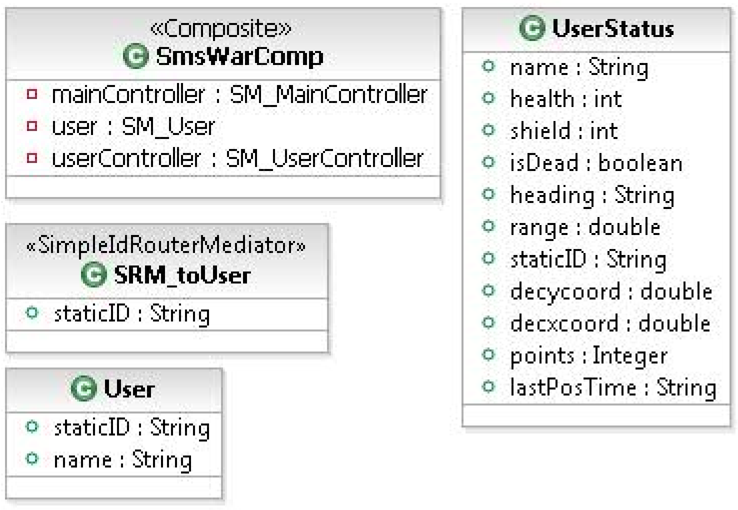


Diagram 2 - Class diagram

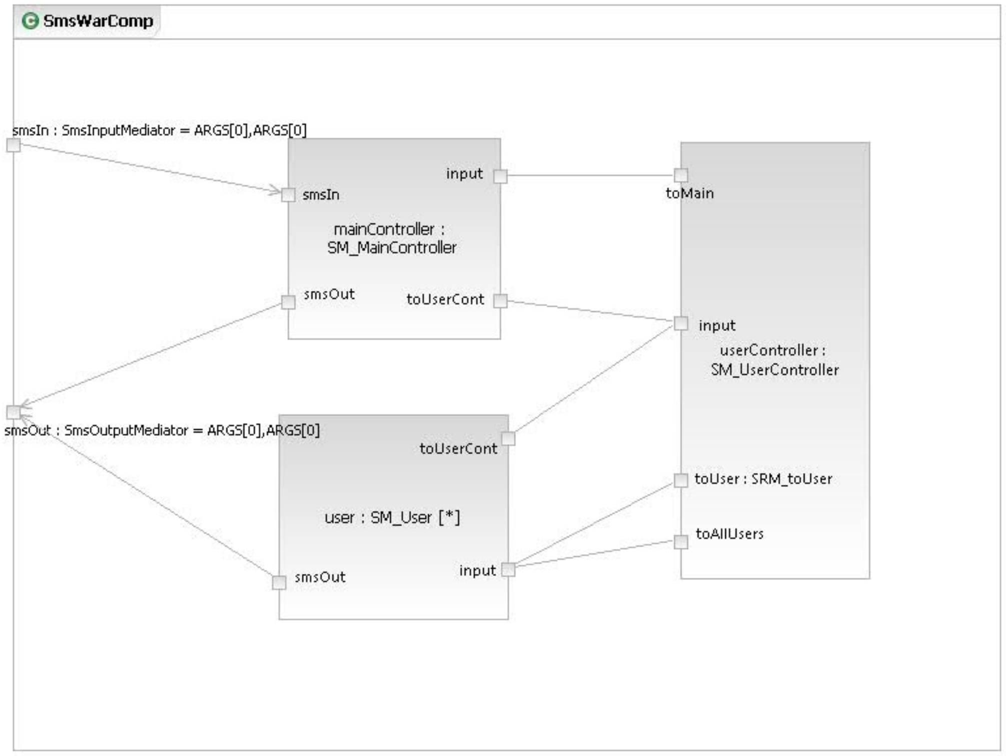

SmsWarComp
smsIn : SmsInputMediator = ARGS[0],ARGS[0]
smsIn
input
mainController : SM_MainController
smsOut
toUserCont
toMain
input
userController : SM_UserController
smsOut : SmsOutputMediator = ARGS[0],ARGS[0]
toUserCont
user : SM_User [*]
smsOut
input
toUser : SRM_toUser
toAllUsers

Diagram 3 - The composite structure

SMSWar System Interactions

Our interactions (sequence diagrams) are drawn in two different levels:

0. This is the outside perspective on the system. The actors are users (mobiles) and the system.
1. This is the inside of our system, showing interaction between the parts of the system.

The messages are written in a pseudo-style without using proper syntax. E.g. the signal Sms has 3 parameters, but we only show the main content of the sms message. Where necessary, we have included more parameters.

Interaction – Register

This diagram shows a user registering in our system.

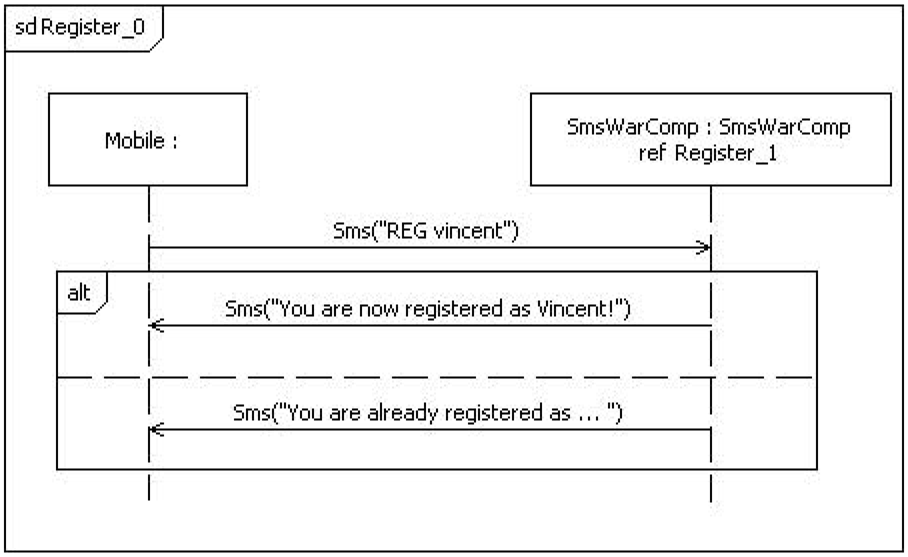

sd Register_0
Mobile :
SmsWarComp : SmsWarComp
ref Register_1
Sms("REG vincent")
alt
Sms("You are now registered as Vincent!")
Sms("You are already registered as ... ")

Diagram 4 - Register new user (level 0)

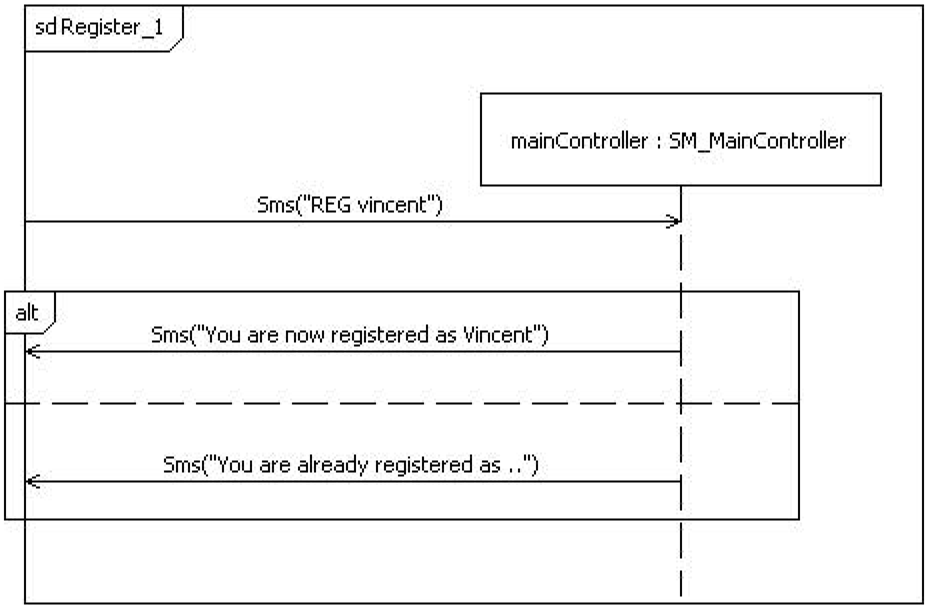

sd Register_1
mainController : SM_MainController
Sms("REG vincent")
alt
Sms("You are now registered as Vincent")
Sms("You are already registered as ..")

Diagram 5 - Register new user (level 1)

Interaction – AnnounceGame

This diagram shows how a new game is announced.

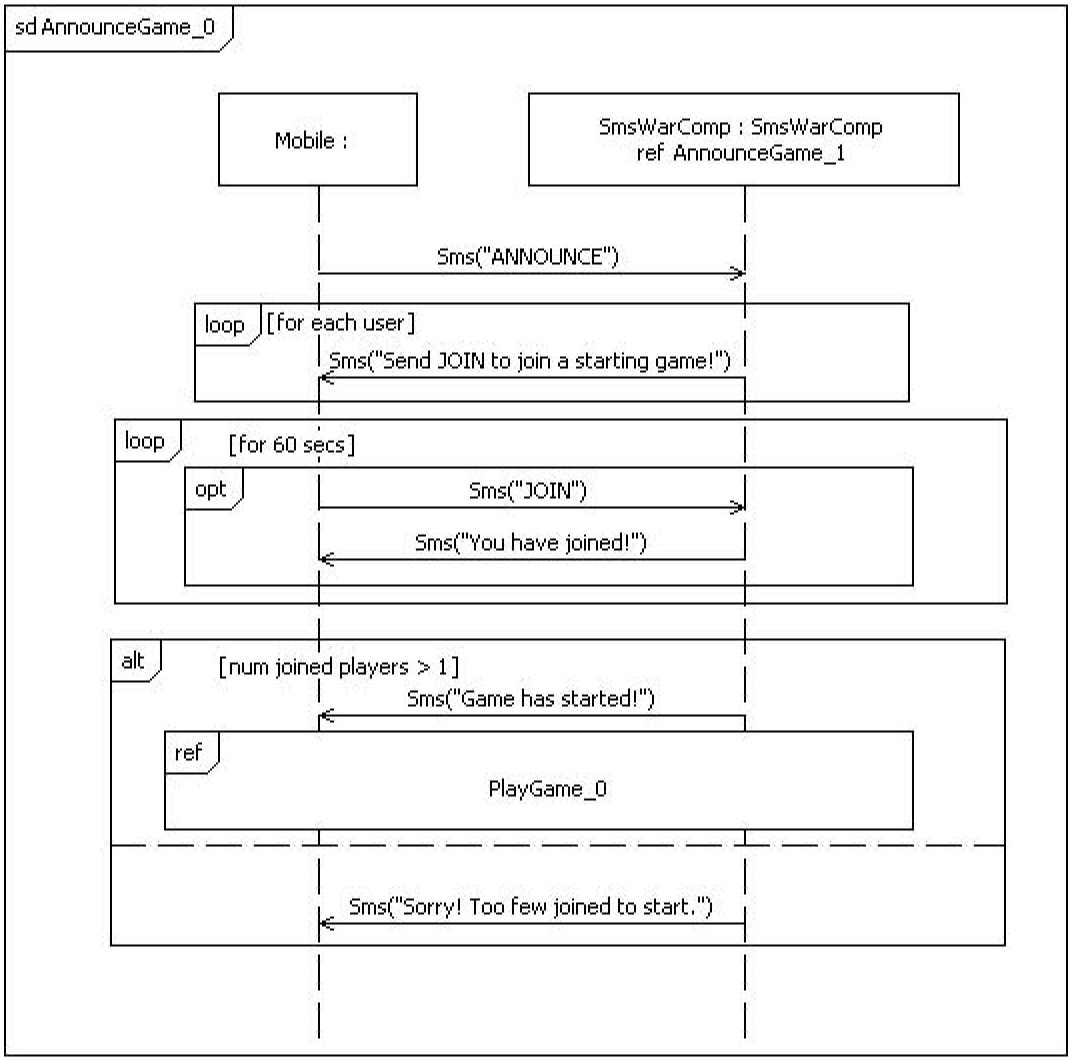


Diagram 6 - A user announcing a new game (level 0)

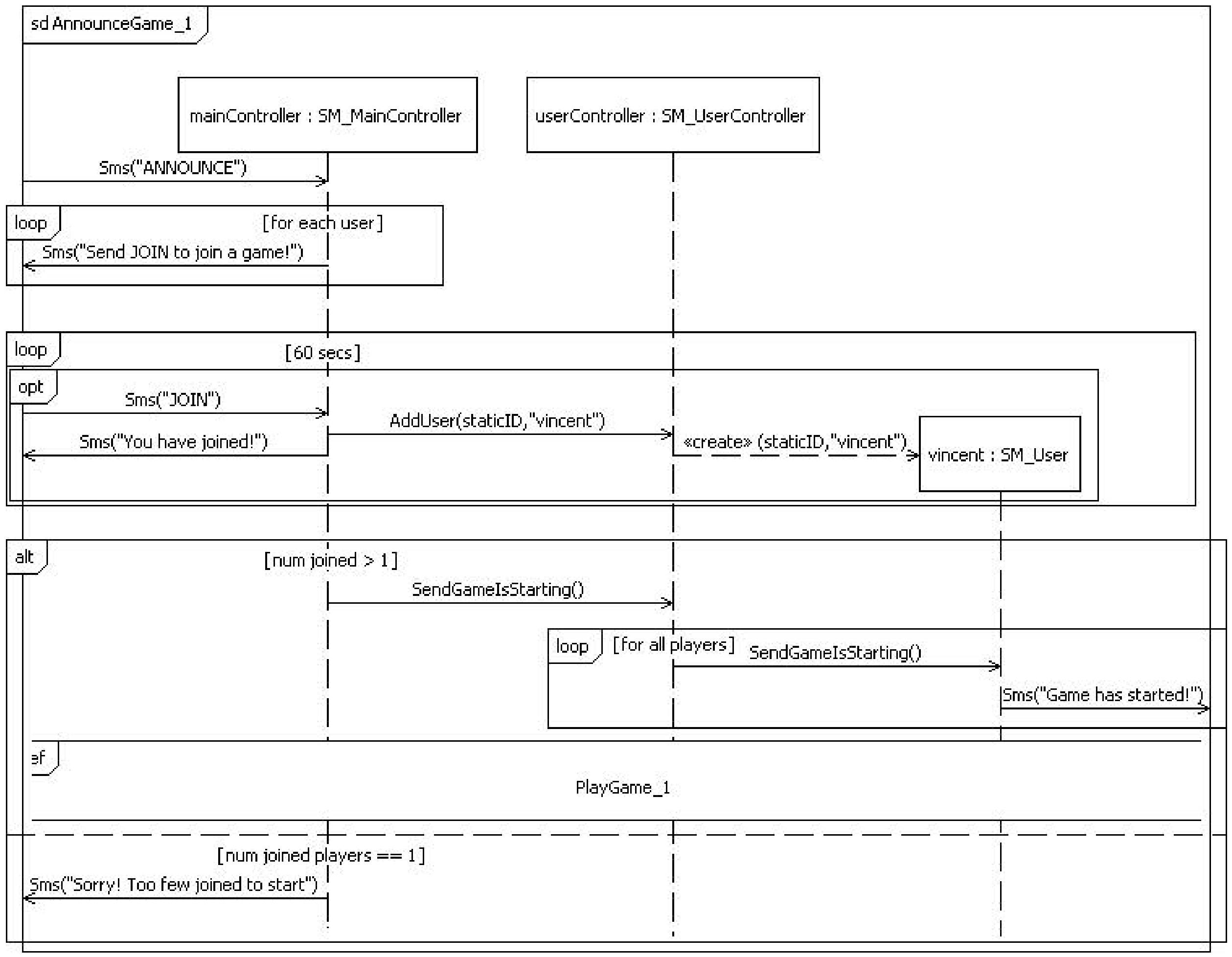


Diagram 7 - A user announcing a new game (level 1)

Interaction – PlayGame

This diagram shows the high level interactions of the various functionalities that might be executed during the general game play. Additionally, the diagram encapsulates the functionality for generating a KML file for positioning of players in Google Earth.

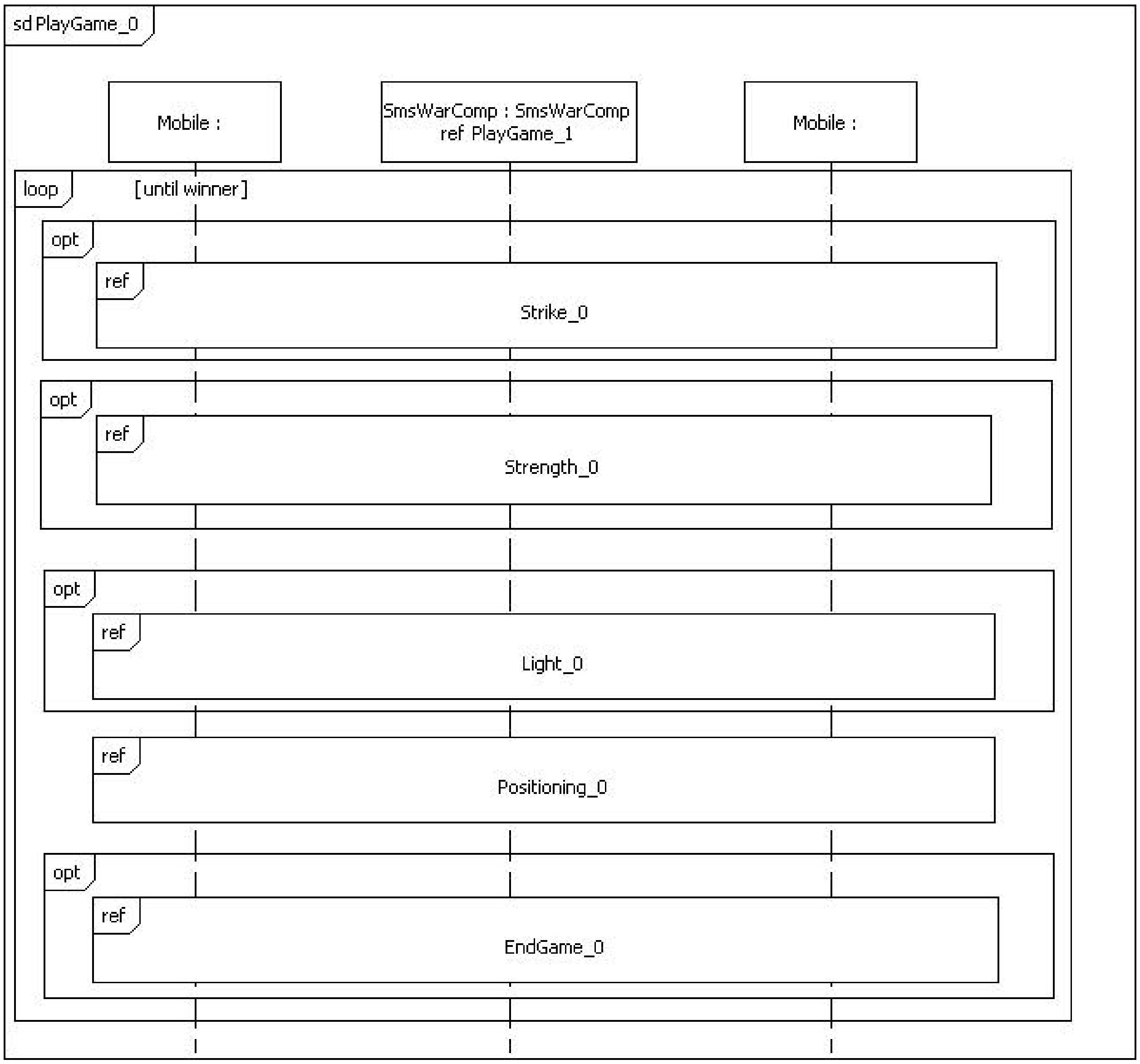


Diagram 8 - Gameplay (level 0)

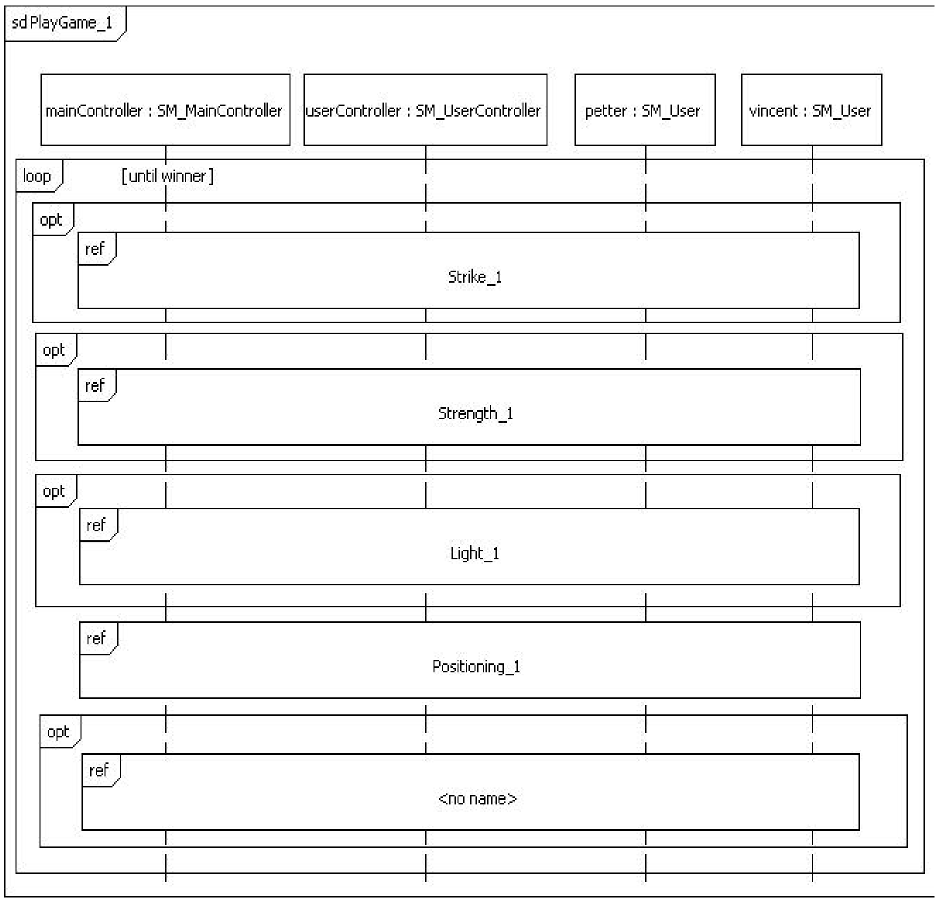


Diagram 9 - Gameplay (level 1)

Interaction – Strike

This diagram shows how the behavior of the **strike** functionality.

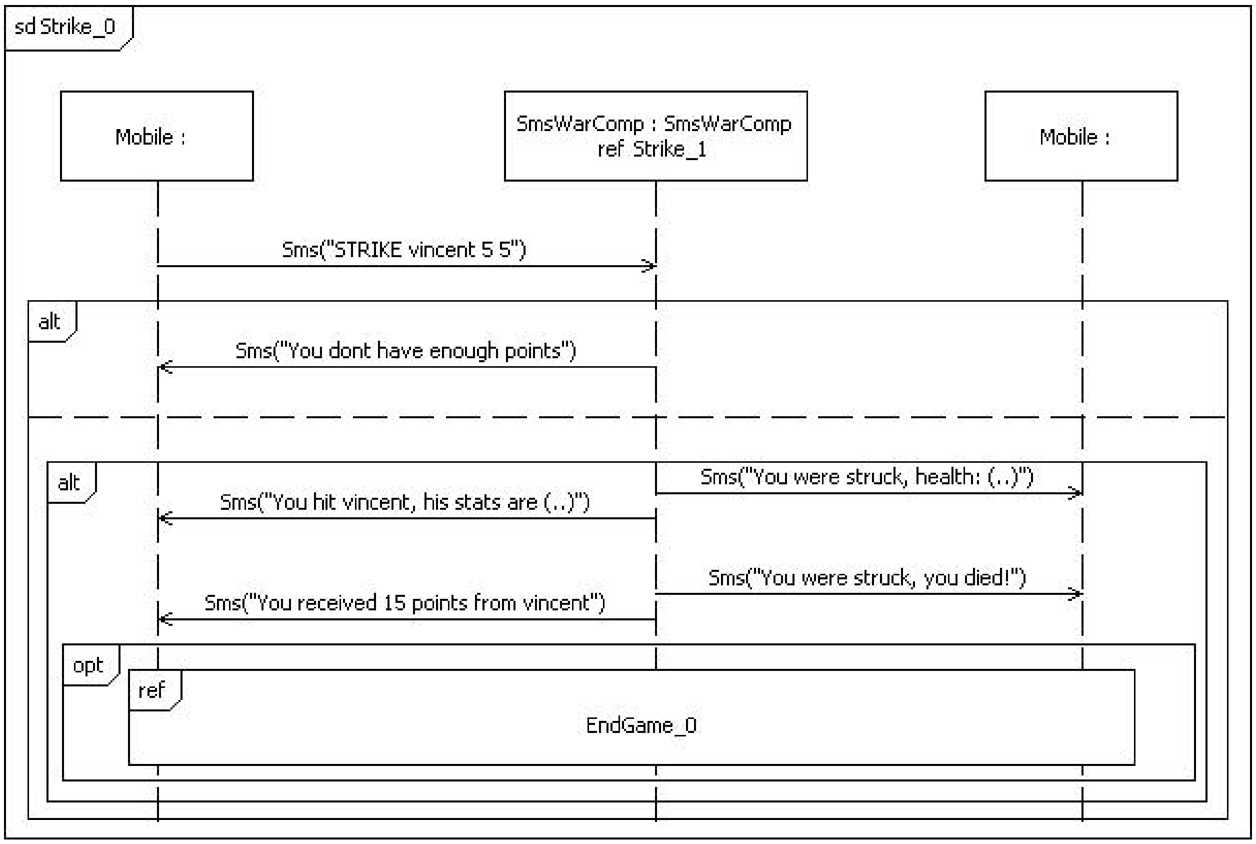

sd Strike_0
Mobile :
SmsWarComp : SmsWarComp
ref Strike_1
Mobile :
Sms("STRIKE vincent 5 5")
alt
Sms("You dont have enough points")
alt
Sms("You were struck, health: (..)")
Sms("You hit vincent, his stats are (..)")
Sms("You were struck, you died!")
Sms("You received 15 points from vincent")
opt
ref
EndGame_0

Diagram 10 - Strike (level 0)

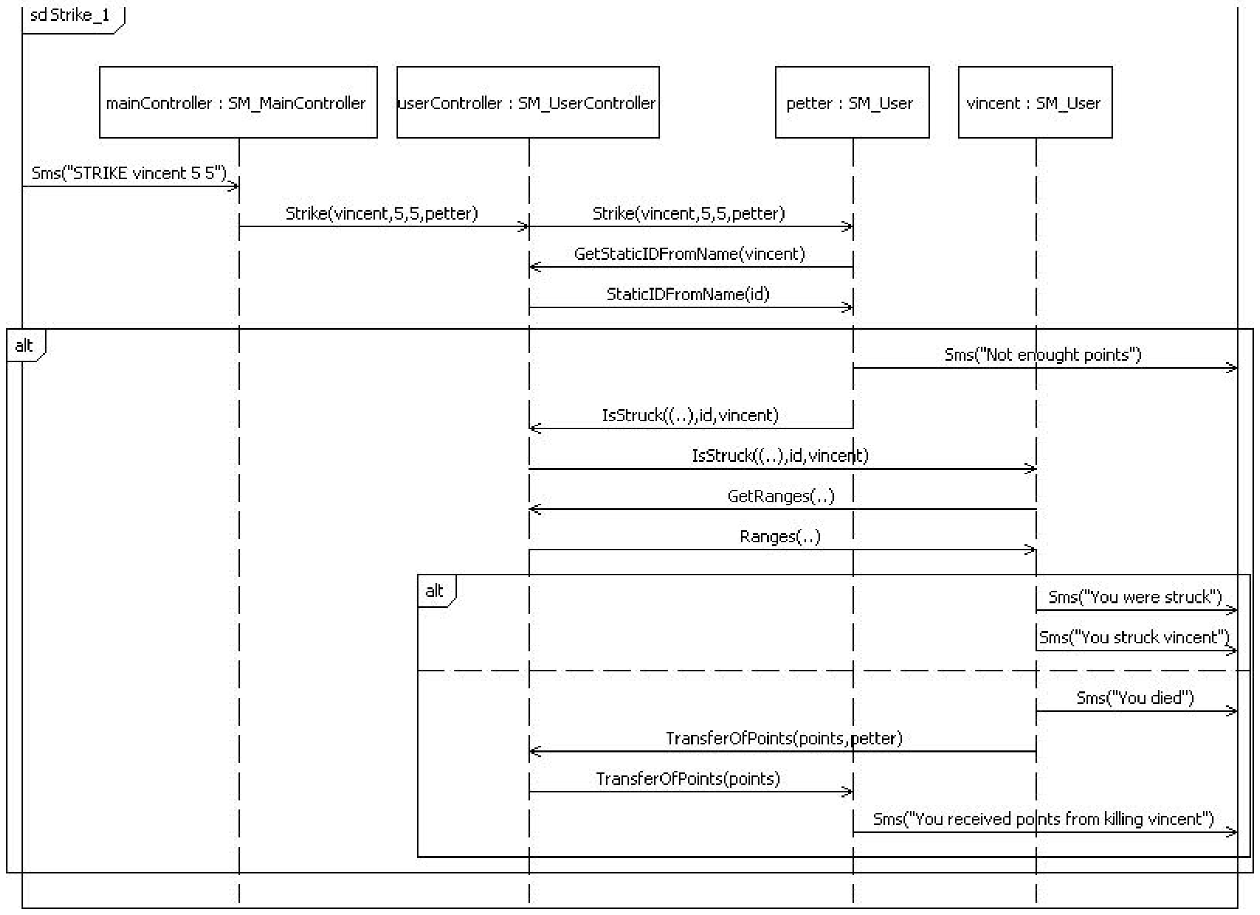

sd Strike_1
mainController : SM_MainController
userController : SM_UserController
petter : SM_User
vincent : SM_User
Sms("STRIKE vincent 5 5")
Strike(vincent,5,5,petter)
Strike(vincent,5,5,petter)
GetStaticIDFromName(vincent)
StaticIDFromName(id)
alt
Sms("Not enought points")
IsStruck((..),id,vincent)
IsStruck((..),id,vincent)
GetRanges(..)
Ranges(..)
alt
Sms("You were struck")
Sms("You struck vincent")
Sms("You died")
TransferOfPoints(points,petter)
TransferOfPoints(points)
Sms("You received points from killing vincent")

Diagram 11 - Strike (level 1)

Interaction – Strength

This diagram shows the behavior of the **strength** functionality.

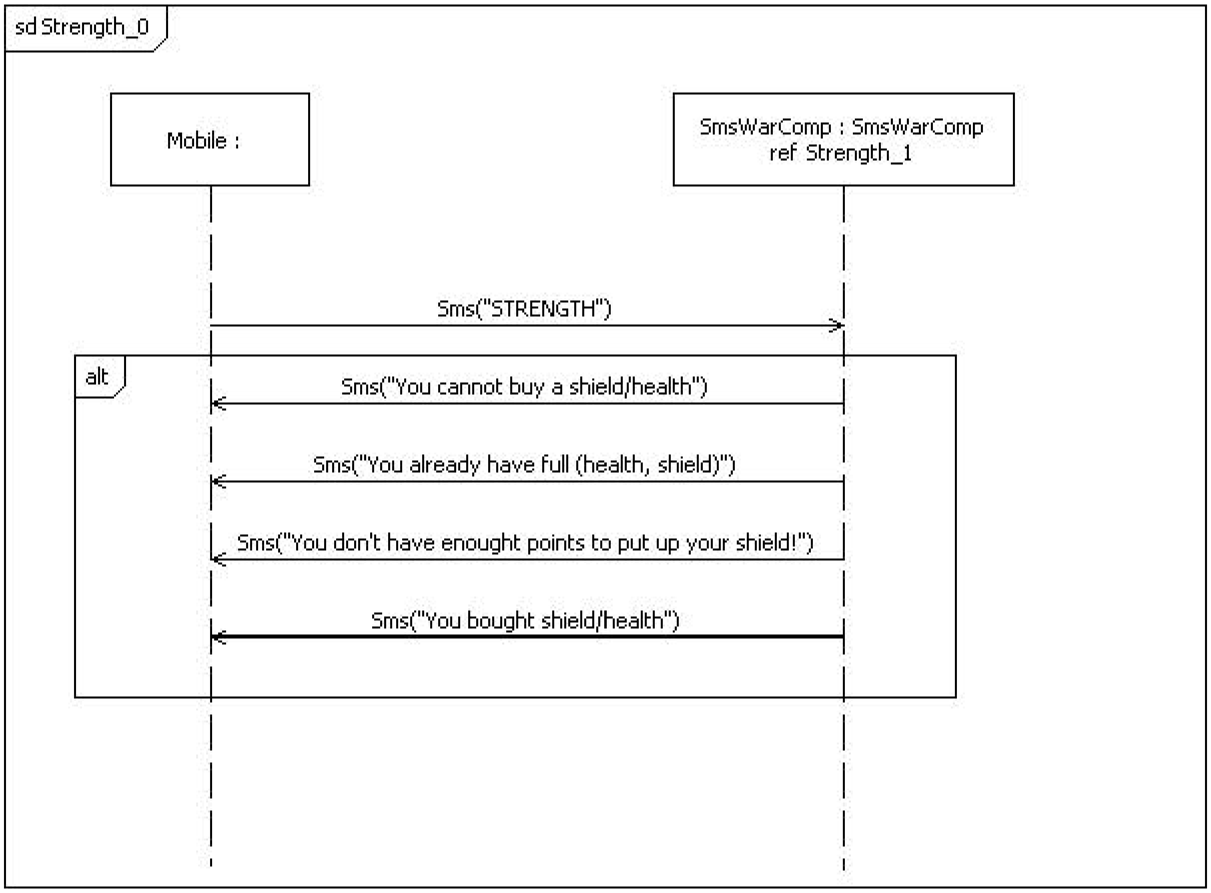

sd Strength_0
Mobile :
SmsWarComp : SmsWarComp
ref Strength_1
Sms("STRENGTH")
alt
Sms("You cannot buy a shield/health")
Sms("You already have full (health, shield)")
Sms("You don't have enought points to put up your shield!")
Sms("You bought shield/health")

Diagram 12 - Strength (level 0)

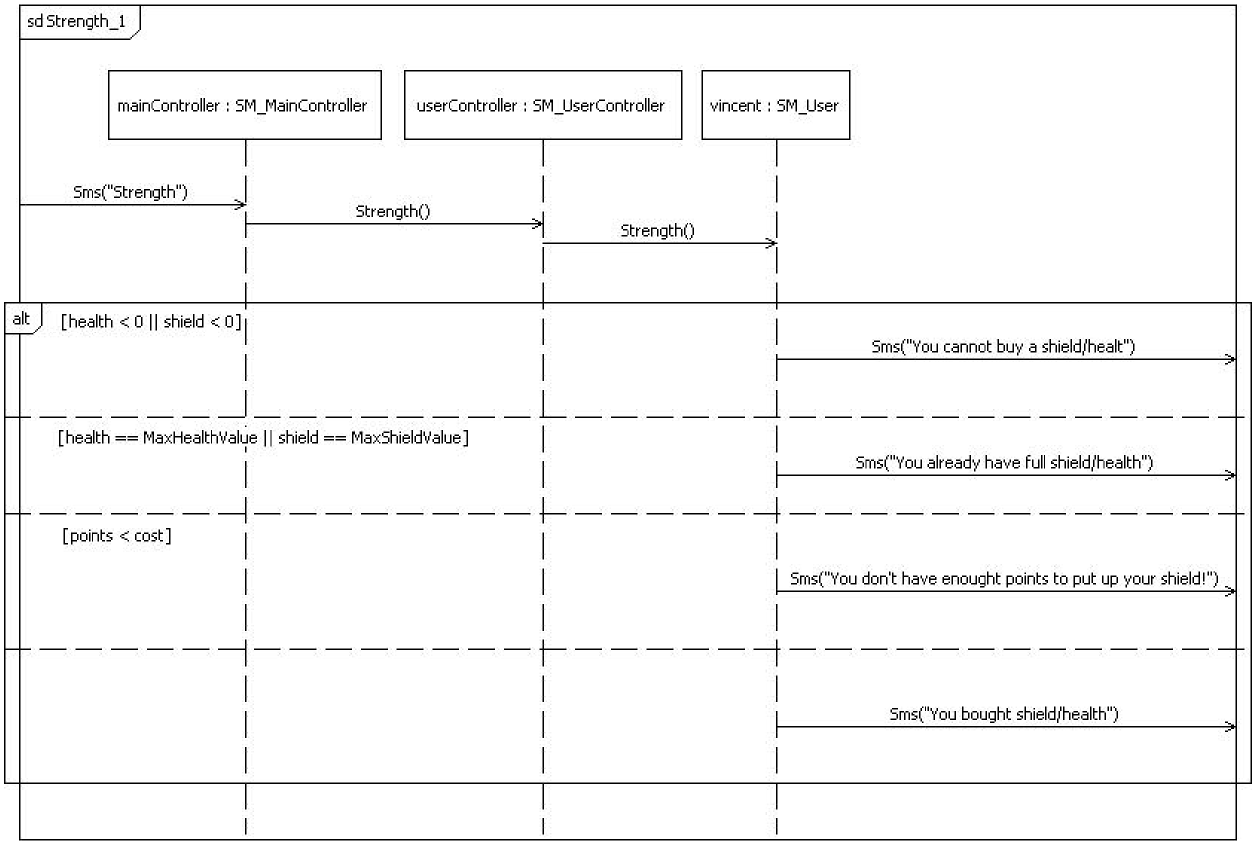

sd Strength_1
mainController : SM_MainController
userController : SM_UserController
vincent : SM_User
Sms("Strength")
Strength()
Strength()
alt
[health < 0 || shield < 0]
Sms("You cannot buy a shield/healt")
[health == MaxHealthValue || shield == MaxShieldValue]
Sms("You already have full shield/health")
[points < cost]
Sms("You don't have enought points to put up your shield!")
Sms("You bought shield/health")

Diagram 13 - Strength (level 1)

Interaction – Light

This diagram shows the behavior of the **light** functionality.

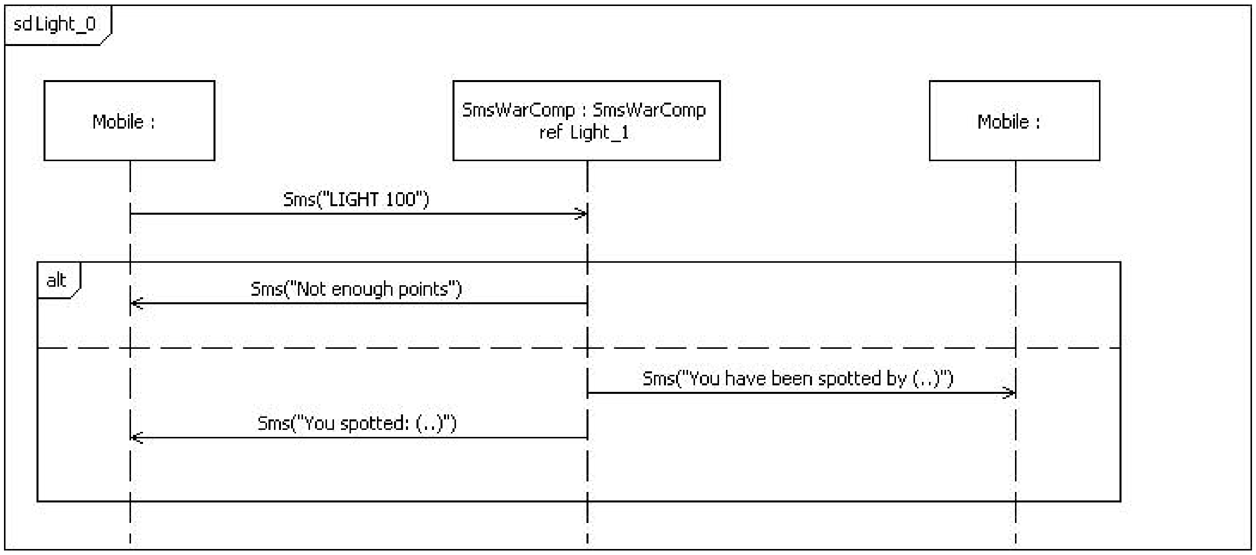


Diagram 14 - Light (level 0)

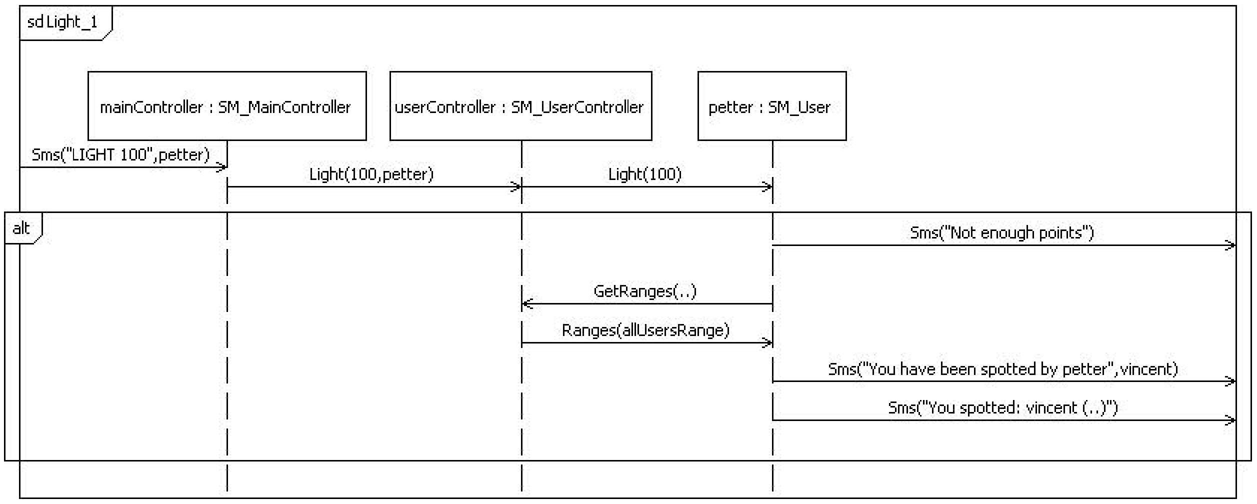

sd Light_1
mainController : SM_MainController
userController : SM_UserController
petter : SM_User
Sms("LIGHT 100",petter)
Light(100,petter)
Light(100)
alt
Sms("Not enough points")
GetRanges(..)
Ranges(allUsersRange)
Sms("You have been spotted by petter",vincent)
Sms("You spotted: vincent (..)")

Diagram 15 - Light (level 1)

Interaction – Positioning

This diagram shows the behavior of the **positioning** functionality.

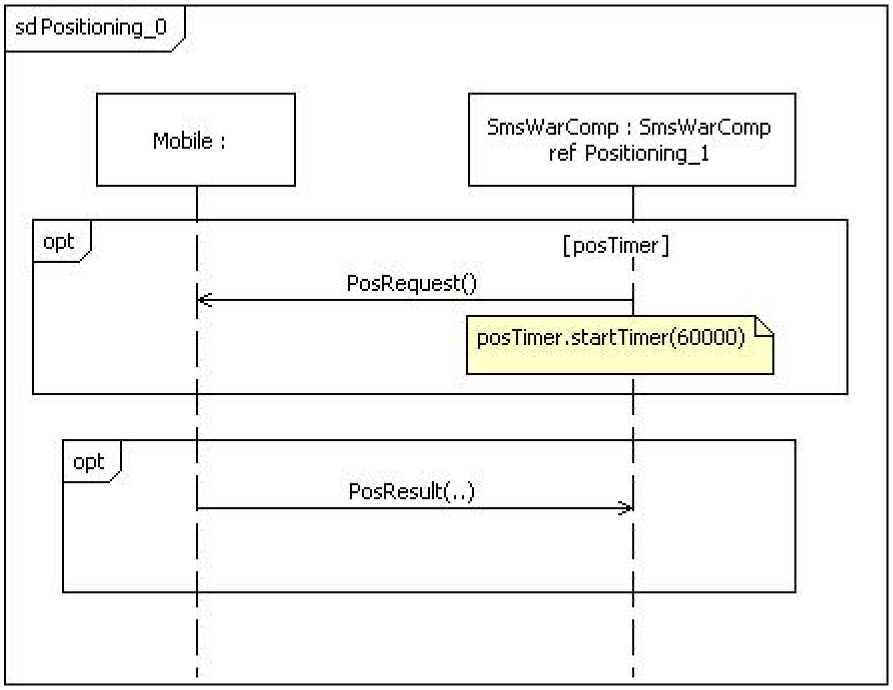


Diagram 16 - Positioning (level 0)

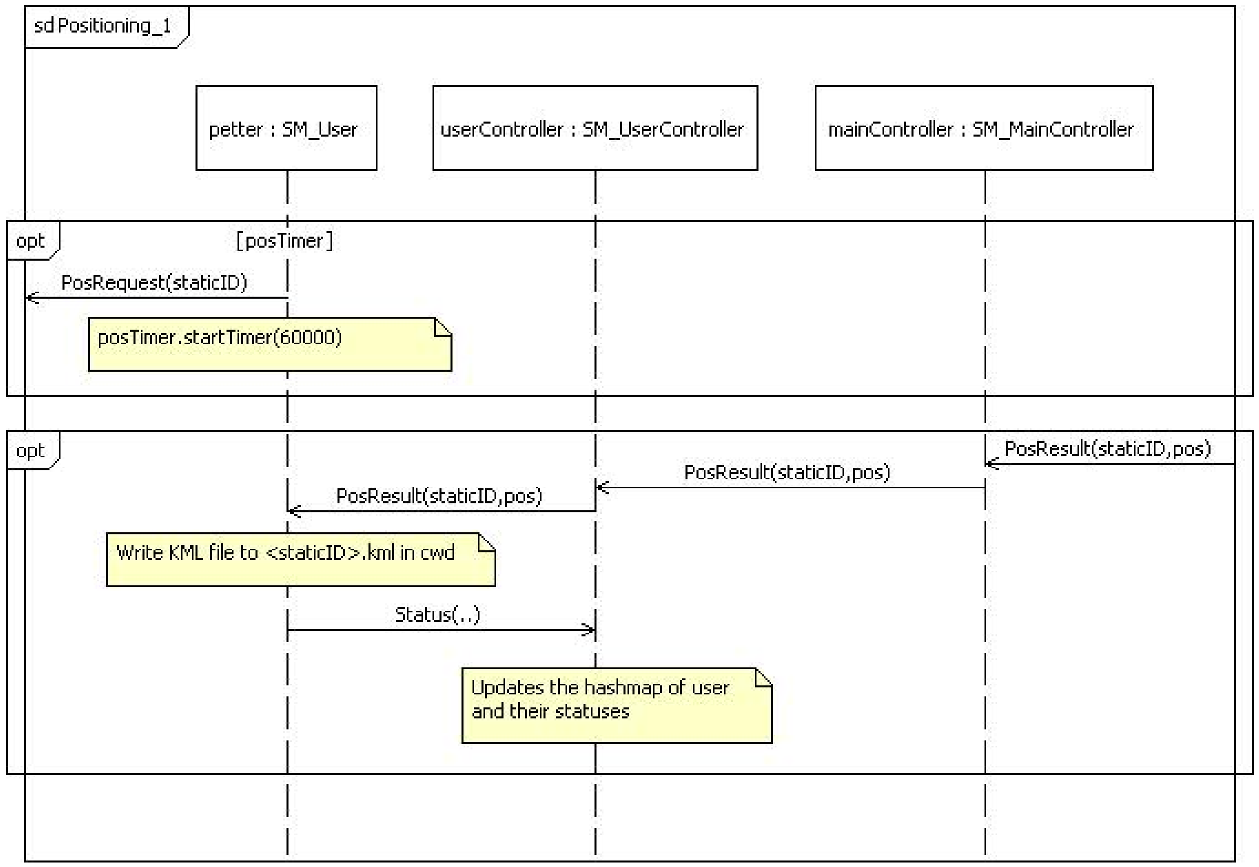

sd Positioning_1
petter : SM_User
userController : SM_UserController
mainController : SM_MainController
opt
[posTimer]
PosRequest(staticID)
posTimer.startTimer(60000)
opt
PosResult(staticID,pos)
PosResult(staticID,pos)
PosResult(staticID,pos)
Write KML file to <staticID>.kml in cwd
Status(..)
Updates the hashmap of user and their statuses

Diagram 17 - Positioning (level 1)

Interaction – EndGame

This diagram shows the behavior of the system when a winner is pronounced and the game ends.

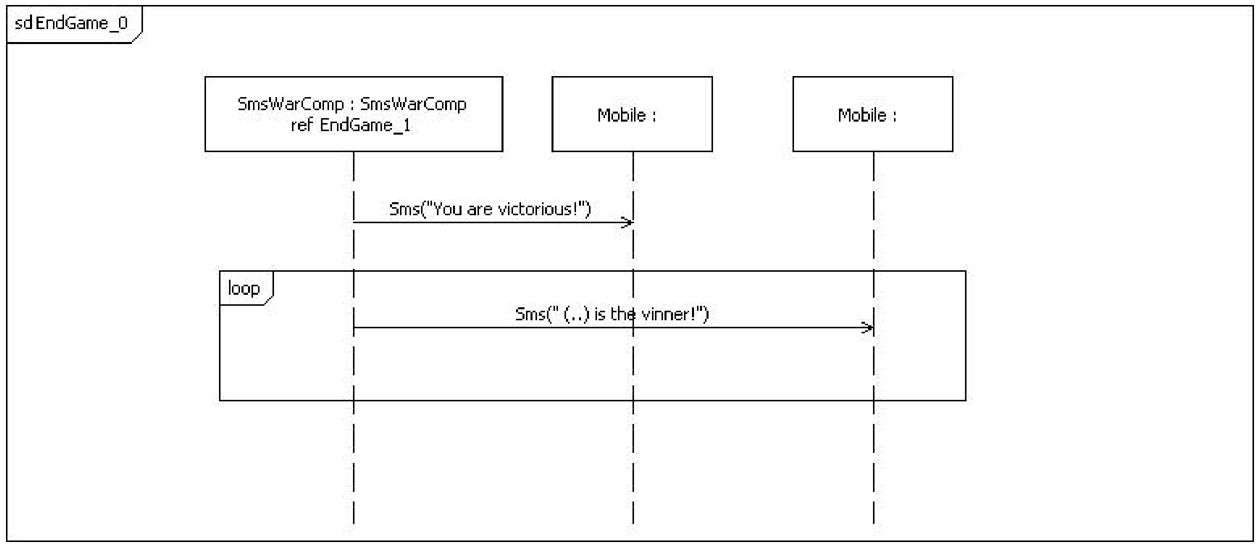


Diagram 18 - End of game (level 0)

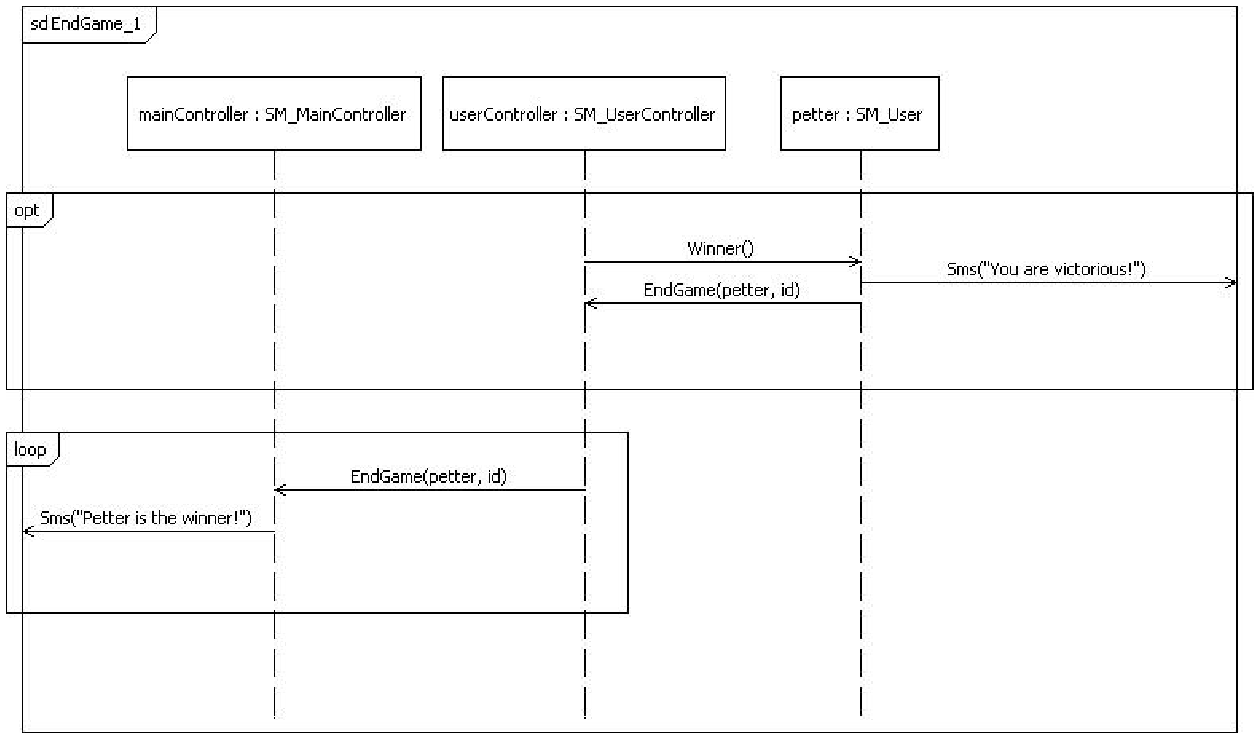

sd EndGame_1
mainController : SM_MainController
userController : SM_UserController
petter : SM_User
opt
Winner()
Sms("You are victorious!")
EndGame(petter, id)
loop
EndGame(petter, id)
Sms("Petter is the winner!")

Diagram 19 - End of game (level 1)

SMSWar Statemachines

The system has three statemachines, as the structure shows.

Statemachine – SM_MainController

See Part - Main Controller (Statemachine) for the overall functionality of this statemachine.

As Diagram 20 - Statemachine SM_MainController shows, the main controller has one **idle** state where it will spend most of its time. It should use minimum time of doing things other than directing responsibilities. It has two submachine states (composite states really), **ParseSms** and **StartGame**. The idle state reacts to the following triggers:

(Please note that in addition to the following functionality, the system always checks for correct syntax for incoming sms messages).

Sms

Parse the SMS, store the command and its parameters in local variables. Then enter **ParseSms** that has an exit point for each available command and one for [else]. It then reacts on the command, if known. The submachine has several checks to verify that the user can enter the command given. See Diagram 21 - Composite state ParseSms.

If the command is **announce**, it will enter the submachine state **StartGame** that handles the functionality of starting a new game and inviting users to join. See Diagram 22 - Composite state StartGame.

Posresult

This signal is forwarded to the user controller.

EndGame

This signal is sent by the user statemachine that has won the game. The system then notifies all players of the winner and kills all instances of the user statemachine and empties the player list of the current game. The system is now ready for a new game.

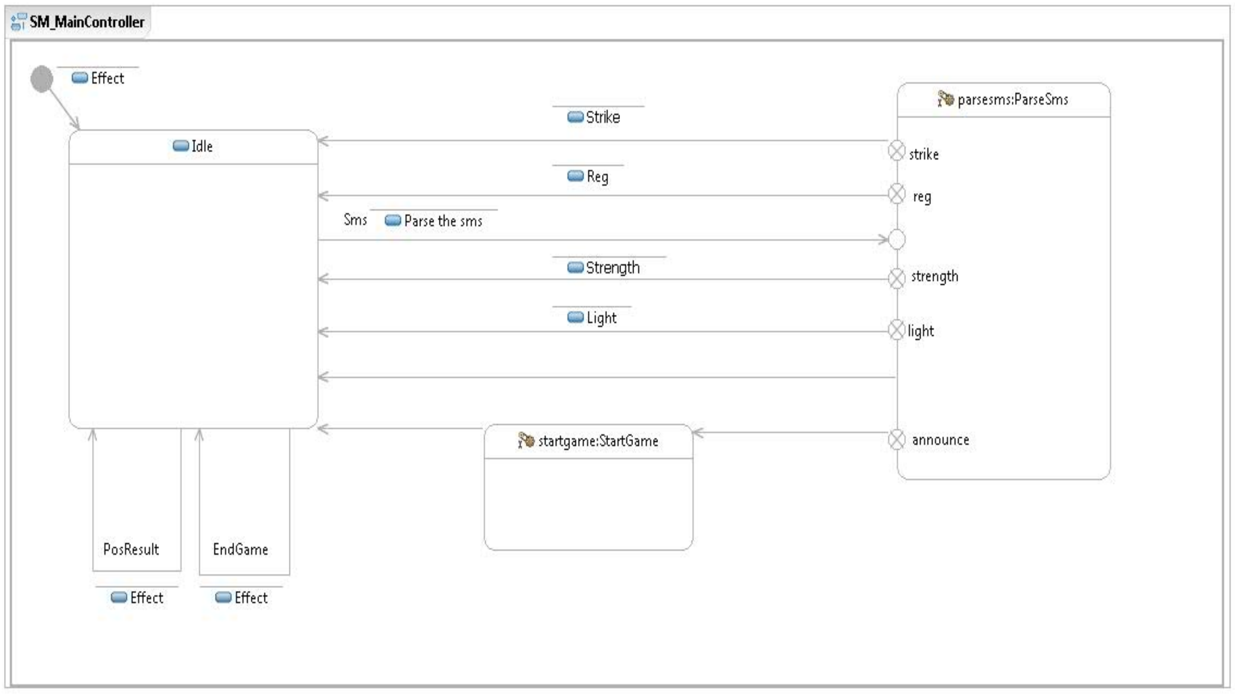
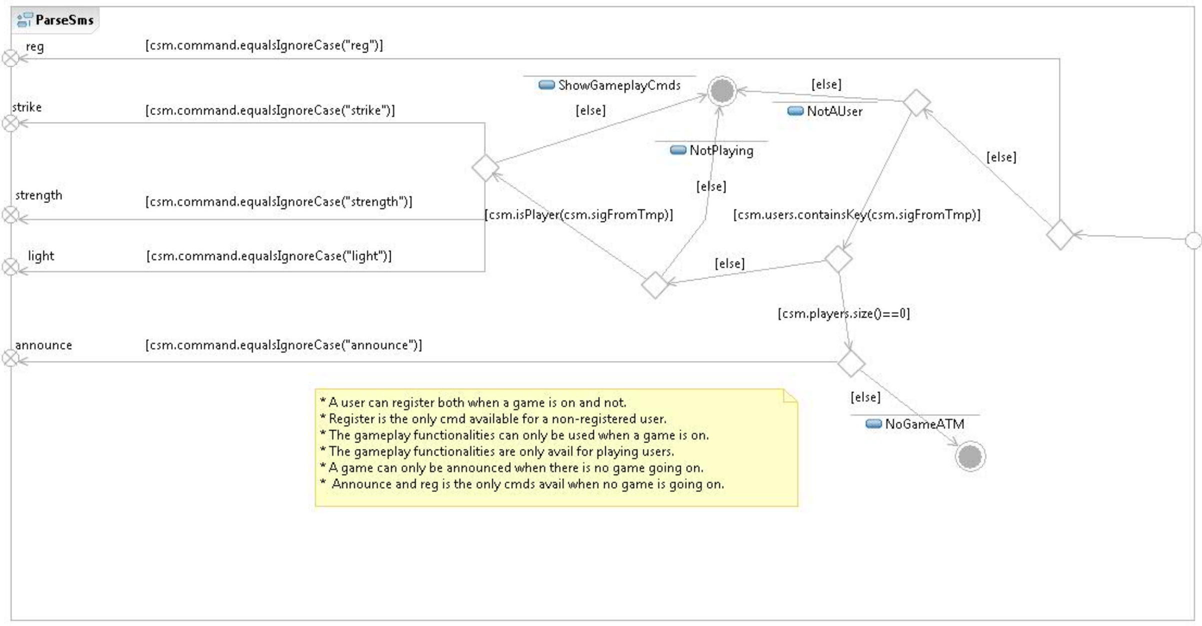


Diagram 20 - Statemachine SM_MainController


ParseSms
reg
[csm.command.equalsIgnoreCase("reg")]
strike
[csm.command.equalsIgnoreCase("strike")]
strength
[csm.command.equalsIgnoreCase("strength")]
light
[csm.command.equalsIgnoreCase("light")]
announce
[csm.command.equalsIgnoreCase("announce")]
ShowGameplayCmds
[else]
[else]
NotAUser
NotPlaying
[else]
[else]
[csm.isPlayer(csm.sigFromTmp)]
[csm.users.containsKey(csm.sigFromTmp)]
[else]
[csm.players.size()==0]
[else]
NoGameATM

* A user can register both when a game is on and not.
* Register is the only cmd available for a non-registered user.
* The gameplay functionalities can only be used when a game is on.
* The gameplay functionalities are only avail for playing users.
* A game can only be announced when there is no game going on.
* Announce and reg is the only cmds avail when no game is going on.

Diagram 21 - Composite state ParseSms

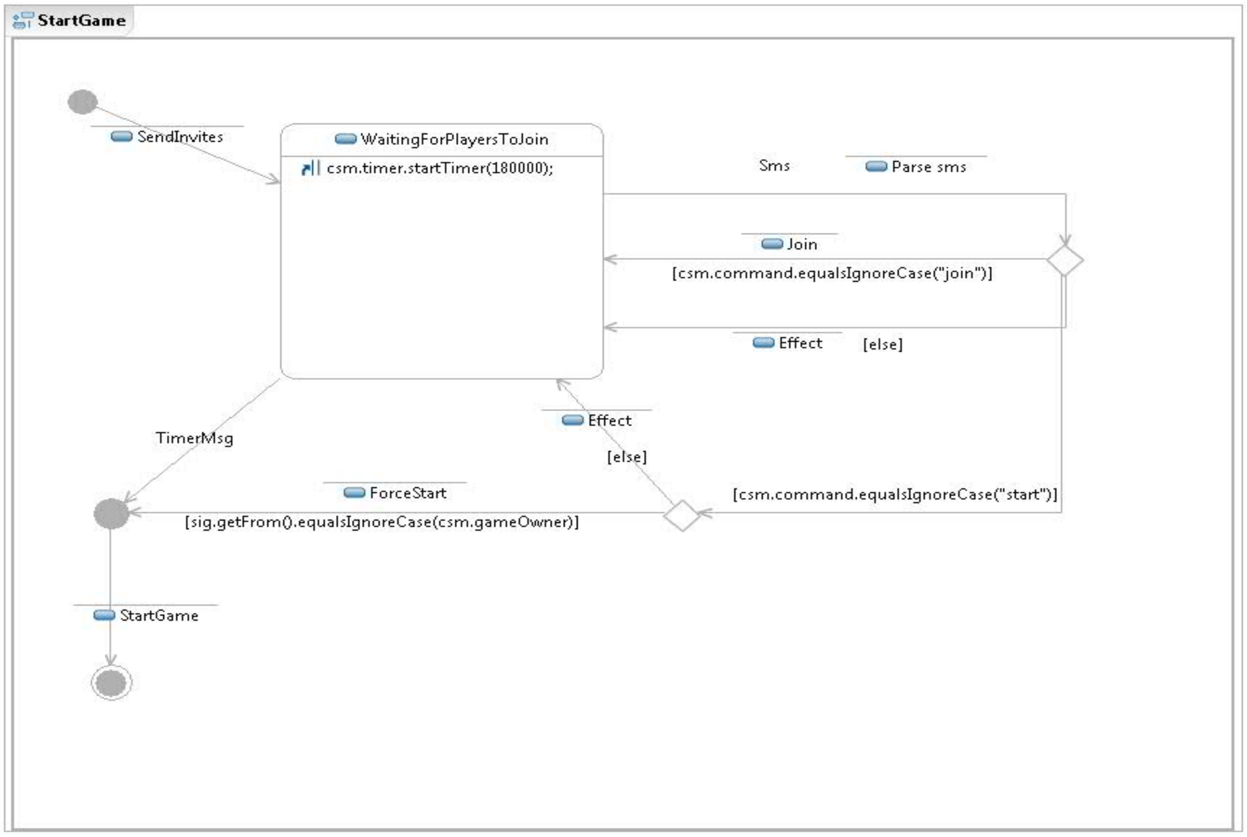

StartGame
SendInvites
WaitingForPlayersToJoin
csm.timer.startTimer(180000);
Sms
Parse sms
Join
[csm.command.equalsIgnoreCase("join")]
Effect
[else]
TimerMsg
Effect
[else]
ForceStart
[sig.getFrom().equalsIgnoreCase(csm.gameOwner)]
[csm.command.equalsIgnoreCase("start")]
StartGame

Diagram 22 - Composite state StartGame

Statemachine – SM_UserController

See Part - User Controller (Statemachine) for overall functionality of this statemachine.

As Diagram 23 - Statemachine SM_UserController shows, the user controller has an **idle** state where it will spend most of its time. It should use minimum time in transitions to maximize the throughput of requests. The state reacts to the following triggers:

AddUser

Received from main controller when a user joins a game. Add a new instance of the user statemachine and update the **idList** of the port (SRM_toUser) with the static id of the user.

Status

Received from user when it has updated its position. At the same time the local table which contains the current status of all the users in the system is updated and the KML file is written to disk. The file is named **SMSWar.kml** and is placed in the current working directory.

GetStaticIDFromName

As our system lets users register with a certain name ("nickname") we need some function that can look up the corresponding static id of a name. This is done by sending this signal to the user controller, the static id is returned in the signal **StaticIDFromName**.

GetRanges

When a user needs to get the range (and direction) to all the other players, the signal GetRanges is used. It returns a hashmap of all other players with their statuses, range and direction relative to the requester.

UserIsDead

Whenever a user is killed, he sends this signal to the user controller. The user controller then checks how many players are still alive. If only one, we have a winner - output the signal **Winner** to the winner's state machine.

GenericRoutingSignalTo…

These three signals are described in SMSWar Signals and are simply forwarding mechanisms to the respective targets. The names are pretty self-explanatory of the target for the signals.

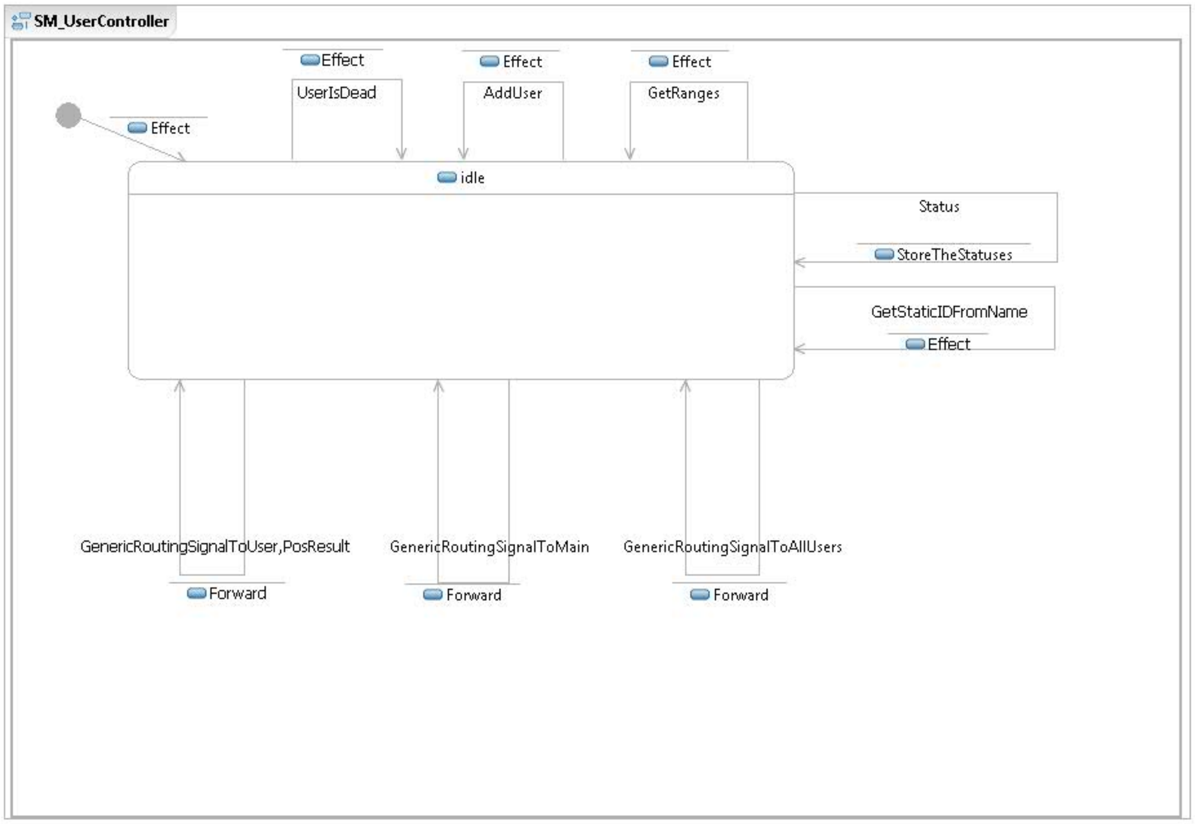


Diagram 23 - Statemachine SM_UserController

Statemachine - SM_User

See Part - User [*] (Statemachine) for overall functionality of this statemachine.

As Diagram 24 - Statemachine SM_User shows, this statemachine is the only one that has the multiplicity [*] in our system. That means that this statemachine will be instantiated many times. It will be instantiated for each player that joins a game. All the instances will be killed when the game is over.

The user statemachine has an **idle** state where it will spend most of its time. It has three composite states; **LightUpArea**, **Strike** and **IsStruck**. It should use minimum time in transitions to maximize the throughput of requests. The signals handled by **idle** are deferred in the submachine states. Also, the submachine states are given a timer to finish, so the game will not freeze if a signal to a submachine state is lost and we previously would have a deadlock. The state reacts to the following triggers:

Strike

Enter **Strike** where the strike functionality will be handled. It outputs the signal **IsStruck()** to the struck users statemachine. See Strike for more information on how this command is used.

IsStruck

Enter **IsStruck** where the functionality of being struck is handled. If the player died, it sends out the signals **UserIsDead()** and **TransferOfPoints()** to the user controller and then enters the state **dead** where it will not respond to any input but the signal **kill** which makes the statemachine go to final state.

Strength

The functionality of strength is handled. See Improve Strength for more information on how the command is used.

Light

Enter **LightUpArea** where the functionality of lightning up is handled. It will find all players that are within the range and notify the players being lit up of who did it and then the player that initiated this functionality with the list of players lit up and their statuses.

TransferOfPoints

This signal is sent from the user statemachine that dies after a strike. It will add the points transferred and then notify the player of the amount of points she got.

SendGameIsStarting

This signal is sent from the main controller when the game has started. It will initialize the user statemachine and notifying the player that the game has started.

PosResult

When the user statemachine receives the positioning result it sends out the status of the user to the user controller via the signal **Status()**.

TimerMsg

This statemachine has two timers:

1. A timer for sending **PosRequest()**.
2. A timer for decreasing the shield value. The shields value will decrement over time.

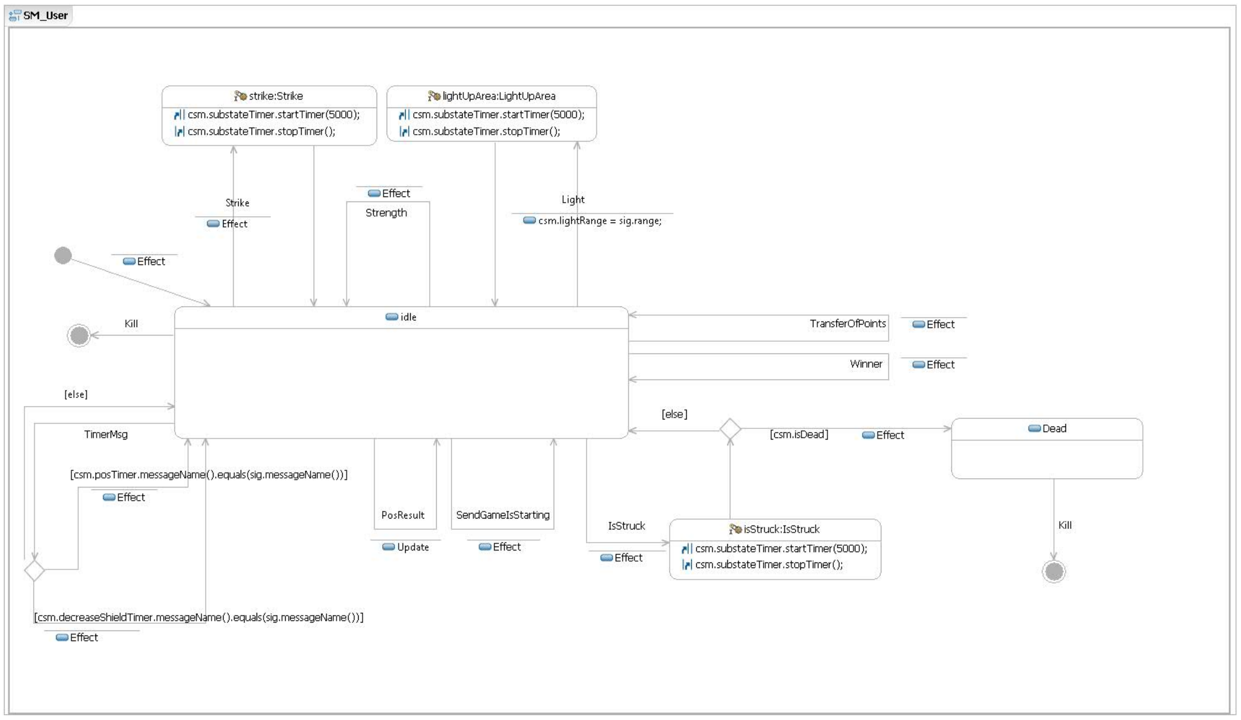

SM_User
strike:Strike
csm.substateTimer.startTimer(5000);
csm.substateTimer.stopTimer();
lightUpArea:LightUpArea
csm.substateTimer.startTimer(5000);
csm.substateTimer.stopTimer();
Strike
Effect
Effect
Strength
Light
csm.lightRange = sig.range;
Effect
Kill
idle
TransferOfPoints
Effect
Winner
Effect
[else]
[else]
TimerMsg
[csm.isDead]
Effect
Dead
[csm.posTimer.messageName().equals(sig.messageName())]
Effect
PosResult
SendGameIsStarting
IsStruck
isStruck:IsStruck
csm.substateTimer.startTimer(5000);
csm.substateTimer.stopTimer();
Kill
Update
Effect
Effect
[csm.decreaseShieldTimer.messageName().equals(sig.messageName())]
Effect

Diagram 24 - Statemachine SM_User

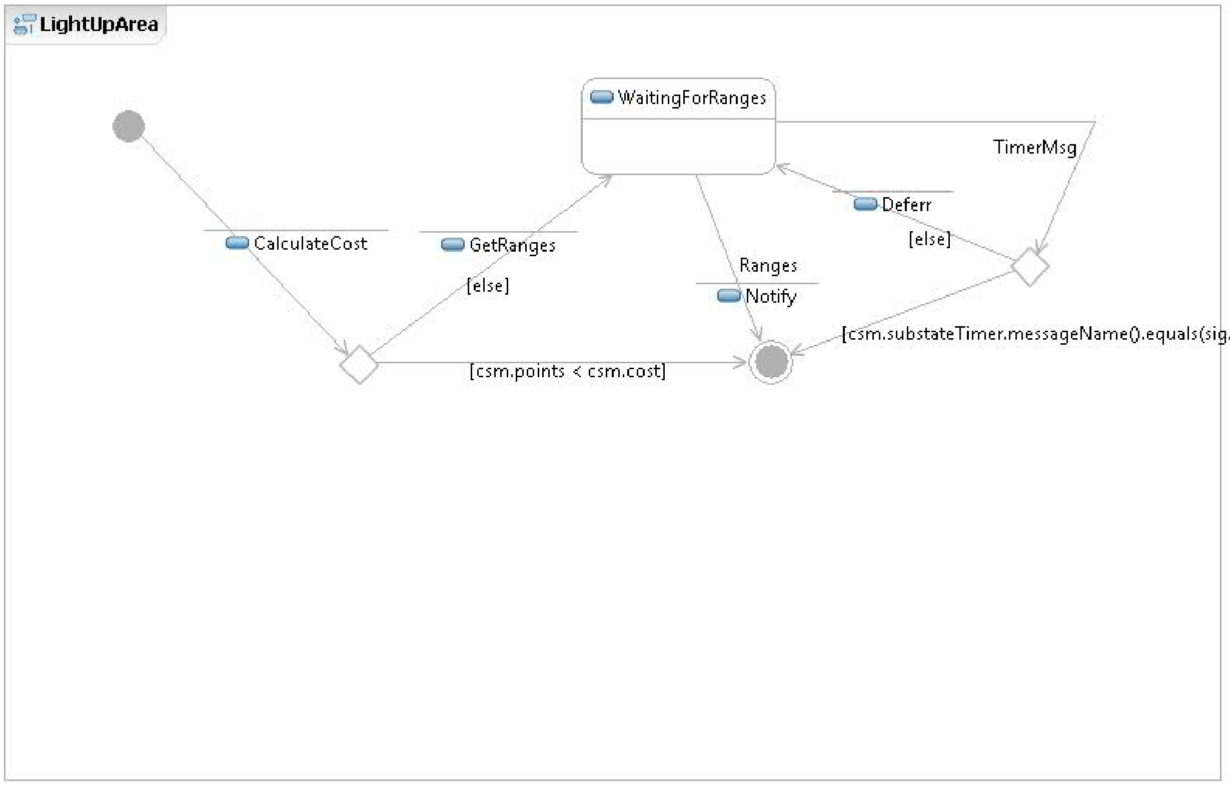


Diagram 25 - Composite state LightUpArea

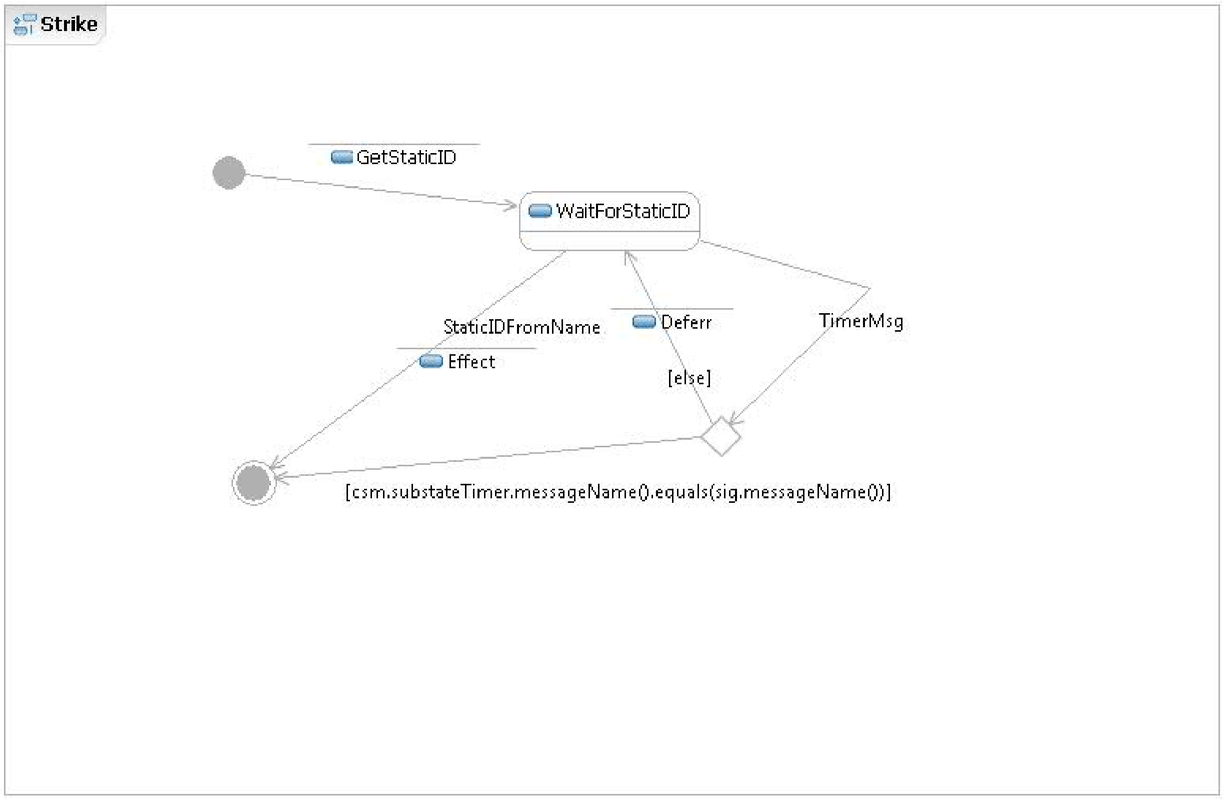


Diagram 26 - Composite state Strike

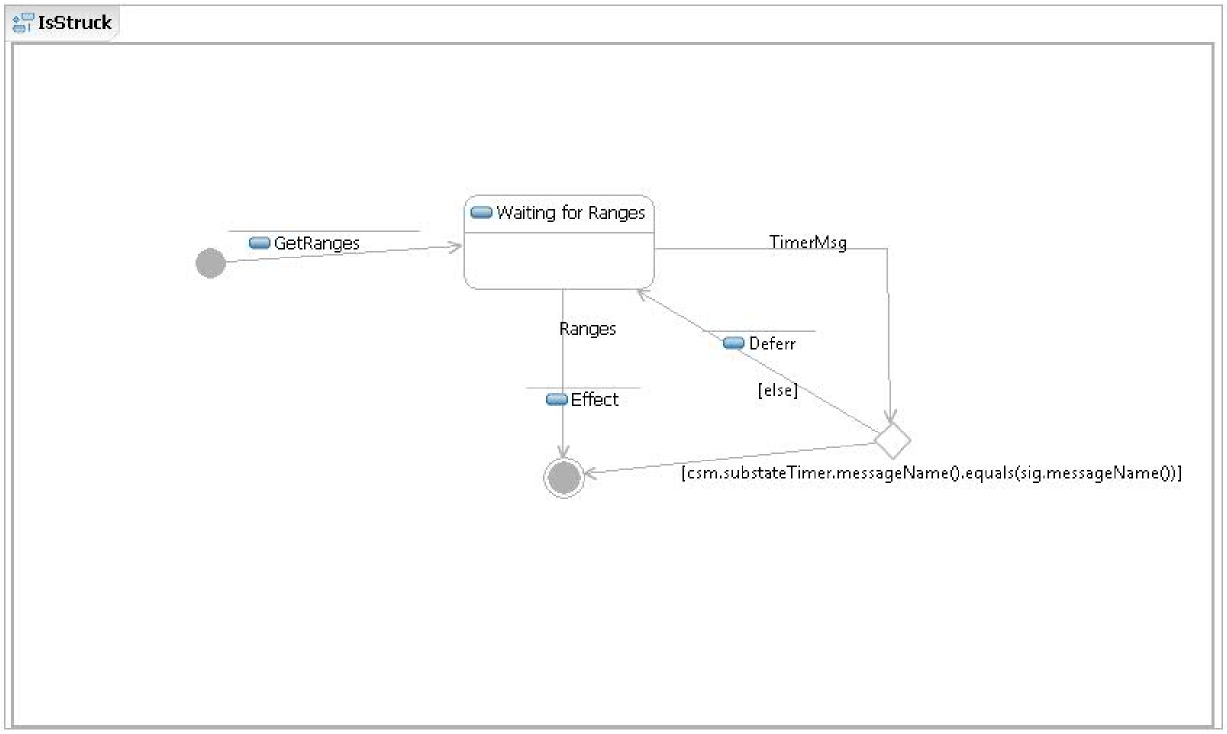


Diagram 27 - Composite state IsStruck

SMSWar Signals

The system has several signals that it uses. We have decided that it is better to have several specialized signals rather than few general ones. It results in less parsing of the signals. We have made some general parent signals to make redundant attributes absent and routing easier:

Signal - Generic

This diagram shows the signals of the system that all contains an attribute called **staticID**. Instead of adding the same attribute to all of those signals, we made a parent signal that the others heritages that attribute from, called **Generic**.

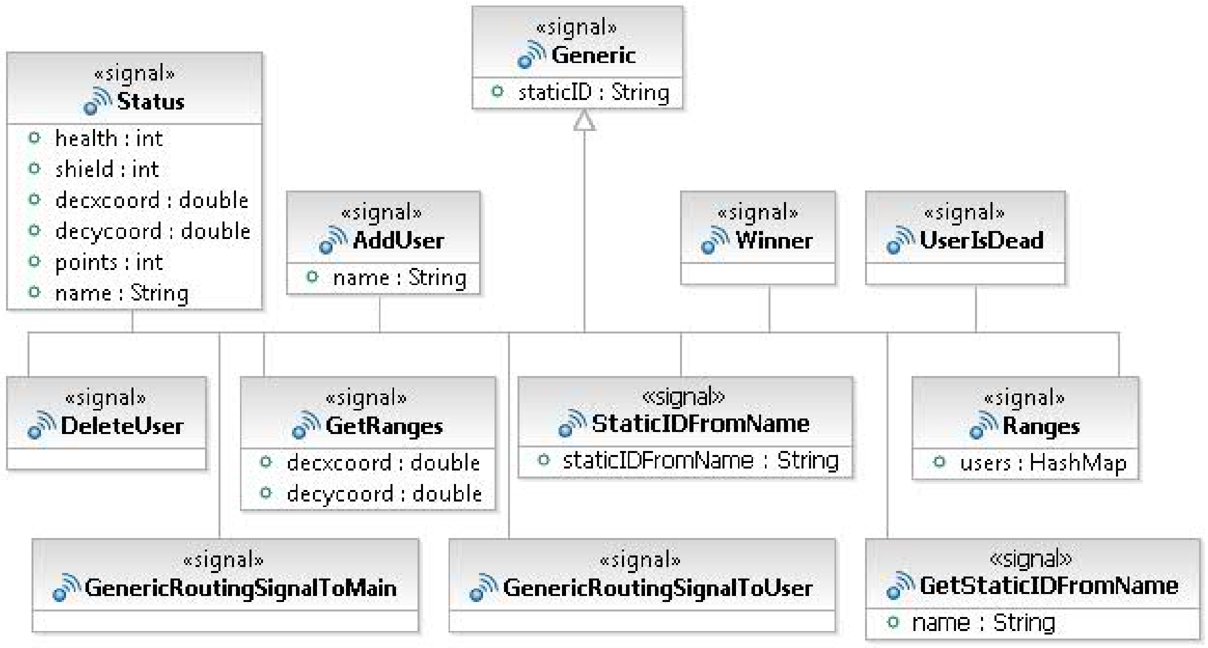


Diagram 28 - Signals with static id as an attribute

Signal – GenericRoutingSignalToAllUsers

The children of this signal will be routed by the user controller to ***all*** instances of the user statemachine.

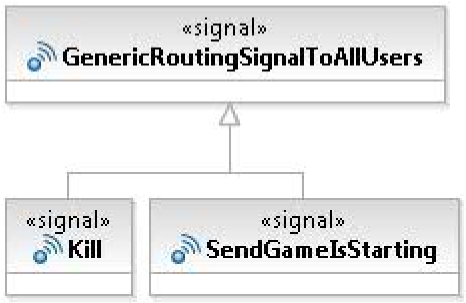

«signal» GenericRoutingSignalToAllUsers
«signal» Kill
«signal» SendGameIsStarting

Diagram 29 - Signals routed to all instances of the user statemachine

Signal – GenericRoutingSignalToUser

The children of this signal will be routed by the user controller to one specific instance of the user statemachine. The routing is done by the port **toUser** which is a SimpleIdRouterMediator.

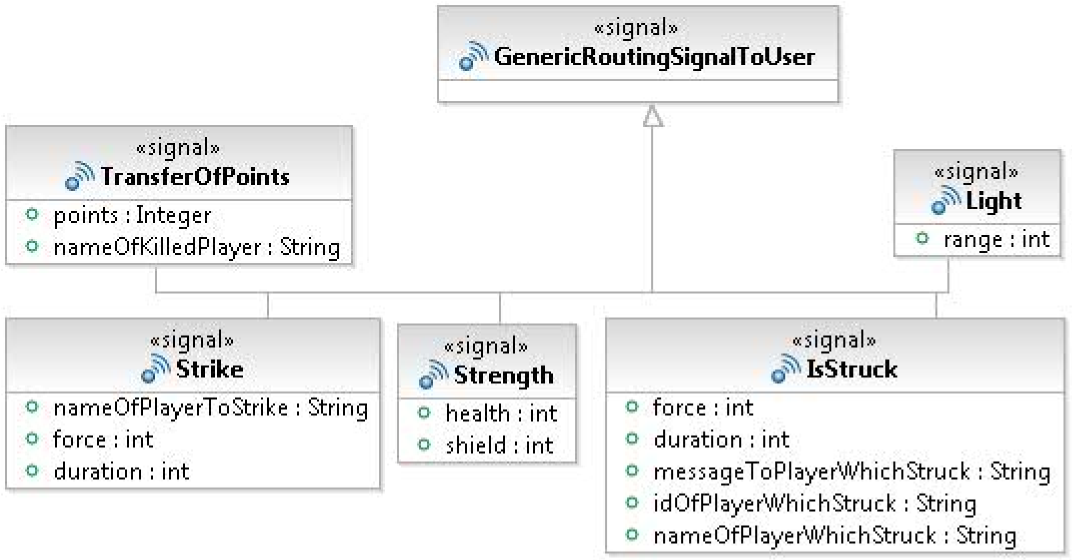


Diagram 30 - Signals routed to one instance of the user statemachine

Signal – GenericRoutingSignalToMain

The children of this signal will be routed by the user controller to the main controller.

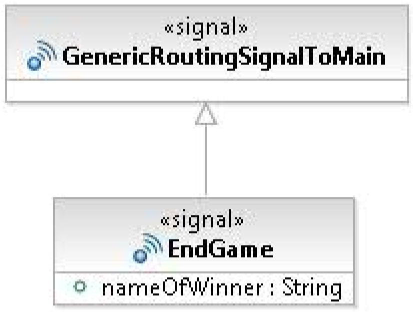


Diagram 31 - Signals routed to the main controller

User Guide

This section contains a short guide for playing SMSWar. It lists all the various commands that a user can send to the system and the effect of the commands and the response of the system. In addition, we have included some other messages which might be sent to users which are not an effect of a command sent from that particular user. E.g. declare the winner of the game.

For all incoming commands

Every incoming SMS to SMSWar will be checked syntactically. The system will respond with a message describing correct syntax for the used command if wrong.

Register

Command

REG <name>

Response (one of the following):

- A message saying that the user was registered with the name <name>.
- A message saying that the user has already registered with a certain name.

Effect

If not previously registered; the user will register in the system with the name <name>.

Strike

Command

STRIKE <name> <force> <duration>

Cost

Cost = <force> + <duration>

Response

To the striker:

1. A message describing the remaining points.
2. A message with details of the result of the strike.

To the struck player:

1. A message which details how much strength the player lost, and his remaining shield, health and points.

If the struck player dies:

2. A message to the striker telling how many points she got by killing him.
3. A message to the struck player telling he's out of the game.

Effect

The user will strike the user with the name <name> with <force> over a time span of <duration>. You cannot hit yourselves.

Force and duration has to be in the interval **[1-10]** to have effect.

Killed users will not be able to play any more until a new game is announced.

If a player is struck, the player will lose points in relation to the distance from the striker, the force and duration of the strike in addition to certain randomness.

Improve Strength

Command
STRENGTH <health> <shield>

Cost
Cost = <health> + <shield>/2

Response
The system will respond with a message containing information about the cost of the operation and the new health and shield parameters.

Effect
This command will increase the user's health and shield according to the parameters given with the command. A player can never get a higher health or shield value than 100. Buying more than this will have no effect. The shield value will decrease over time.

Light

Command
LIGHT <range>

Cost
Cost = (<range>/400)^2

Response
The system will respond with a message notifying the players that were spotted who spotted them. The spotter will be notified about their distance, direction, health, shield and points. In addition, it will include the current points of the player which executed the command.

Effect
This command lights up an area of distance <range> around the player. Players can use this as a tactical advance.

Game specific commands and effects

Announce Game

Command
ANNOUNCE

Response
- A message for every fifth person that joins.
- A message describing the number of players that were invited.

Effect
An invitation will be sent out to all registered users. Those users can respond with the command JOIN to join the starting game. The invitation period is by default 3 minutes. The user starting a game – game owner - (who sent the ANNOUNCE message) can force start the game manually by sending **START**. The game owner will be notified of every fifth player that joins. The game owner does *not* have to send JOIN.

System Messages and Effects

Game Started

Message
This message will be sent out to all players that joined a starting game.

Effect
This message marks the start of the game.

User is Dead

Message
This message will be sent to a player when his health reaches 0.

Effect
The player is out of the game and the system will not respond to any commands from this player anymore during this game.

Winner

Message
A message going out to all players telling them that the game is finished and who won the game.

Effect
The game is finished and the system is now ready for announcement of a new game.

Summary

SMSWar is a game developed as the obligatory exercise 2 in INF-5150 autumn 2007. As the course focus on development of unassailable systems, a big emphasis was put on the design phase of the system development. We used sequence diagrams extensively and modeled the functionality expressed by the sequence diagrams using state machines in RSM. These again were translated into java code using JavaFrame.

The requirements for exercise 2 expressed a number of functional requirements which the system should fulfill. As far as we can see, all of these functions are integrated in SMSWar. In addition, we changed the shield functionality to strength functionality. The strength of a player is defined as a tuplet of the health and shield of a player. If the health of a player reaches zero, a player dies.

Bibliography

INF-5150 Oblig 2. (2007). *UIO.* Retrieved from http://www.uio.no/studier/emner/matnat/ifi/INF5150/h07/undervisningsmateriale/Oblig-2-INF5150-2007.pdf

INF-5150 Oblig 2

CORAS Security Analysis Report

SMSWar

November 26th, 2007

Group 3:

Bjørn Brændshøi

Bjørn Tveter

Kestutis Mintauckis

Petter Risholm

Vincent Demeester

Introduction

This report is part of the obligatory exercise number 2 in the class INF 5150 autumn 2007. The task was to do a security analysis of the system which was designed and developed in the first part of exercise number 2, SMSWar. SMSWar allows people to play a "war" game by sending Sms messages to a central server which is able to localize the players through the use of PATS. In short, the requirements of the game were that a user was given a certain number of points and certain strength at the beginning of the game. During the game, players can execute commands by sending sms to the central server to for instance locate other players, strike other players or increase their own strength. An additional requirement was added to the game, that a player was allowed to buy additional points in the game for a specific amount of money.

A CORAS security analysis is divided into seven steps. The following sections describe the analysis which was performed in the different steps.

Step 1: Introductory Meeting

In step 1, the owner of the SMSWar system presented the system, the target of the analysis and the goal of the analysis. This section provides a summary of this presentation.

Short description of the system

Diagram 32 contains the client drawing of the system in question. In the diagram we see a few players which can communicate with the SMSWar systems through the use of their cell phones. An sms message always go through the pats server where the sms messages are transformed into network packets going over a secure line through the UiO firewall and into the SMSWar system.

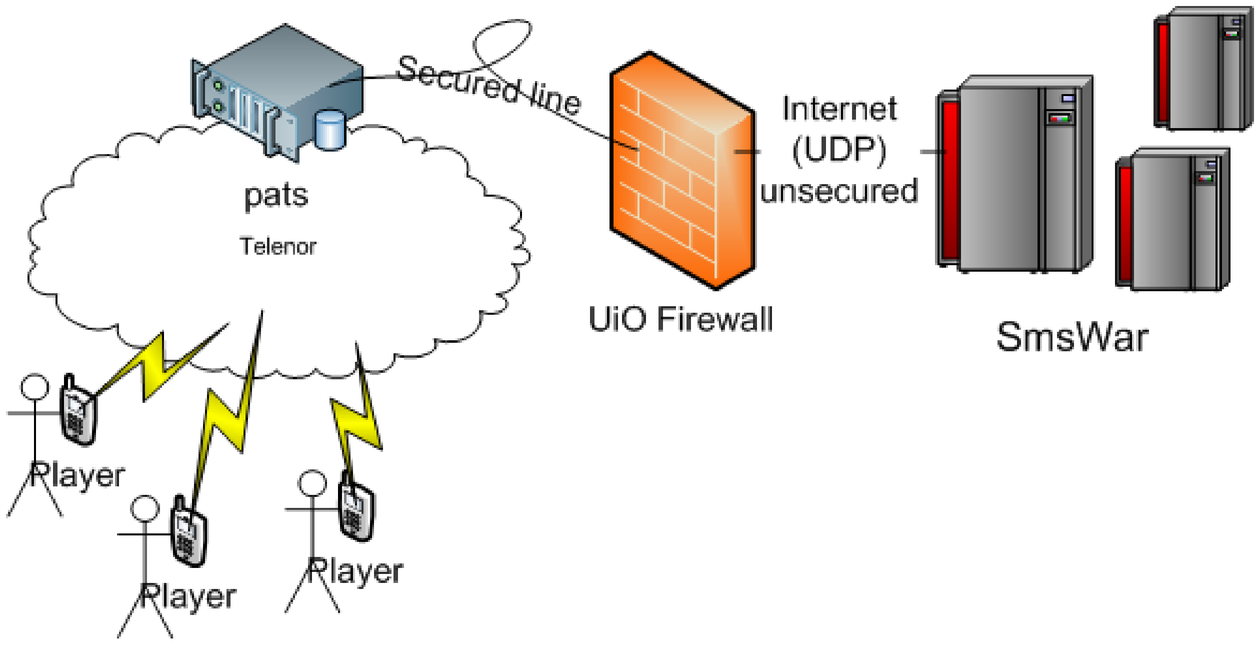


Diagram 32: Highlevel view of the SMSWar system

Stakeholders

The stakeholder in the analysis is the owner of the system which will get the profit from the game.

Goals of the analysis

A few goals were defined by the stakeholder which will be the main guideline for the rest of the analysis:

- Maximize the profit from the system.
- Prevent the system administrator from losing money.
- Maximize user satisfaction/reputation.
- User trust

Target of the analysis

The target of the security analysis will be the SMSWar system. However, we make a few assumptions regarding certain parts of the system which we either assume are safe/secure, or that we don't have access to do any improvements on these parts. The following list contains our assumptions:

- We assume that the communication channels between UIO/pats and Phone and pats are secure.
- We assume that PATS is secure and have 100% up time.
- We assume that money is transferred directly to the bank by the user and this transaction is secure. The SMSWar system receives a message from the bank describing who deposited money and how much money they deposited.

Step 2: High-Level Analysis

In step 2, a high level analysis of the system was carried out where the assets and the main risks were identified. A meeting between the client and the analysts was also held, where the analysts presented their understanding of the system and target of the analysis.

Target as understood by the analysts

Two different UML diagrams were presented to the client to portray the system as the analysts see it.

Static part of the system

The static part of the system, that is hardware which is used in the system such as cell phones, routers and machines, were described in a UML class diagram. This diagram can be seen in Diagram 33.

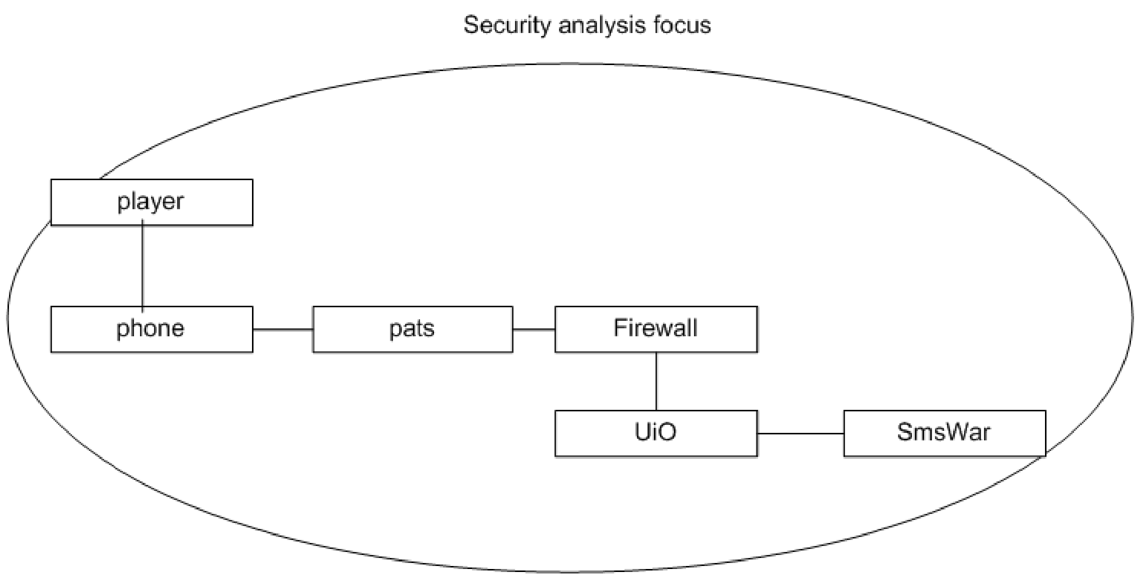


Diagram 33: Class diagram of the static part of the system.

Dynamic part of the system

The dynamic parts of the system such as the information flow through the system were modeled by a UML behavior diagram. This diagram can be seen in Diagram 34.

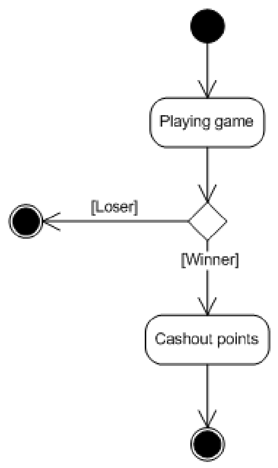


Diagram 34: Dynamic part of the system.

Identification of Assets

In dialog with the client, the following assets were identified for the analysis

- User reputation
- Money
- Points
- Privacy with regards to detailed user information (phone numbers, bank account numbers)

Diagram 35 lists the identified assets and the relations between them. Two of the assets were identified as indirect assets, user reputation and money. User reputation is not affected when either of the two direct assets, user privacy or points are affected. Money is only affected by the points asset and were modeled as an indirect asset since they are in the bank and cannot be reached by any hackers through our system directly.

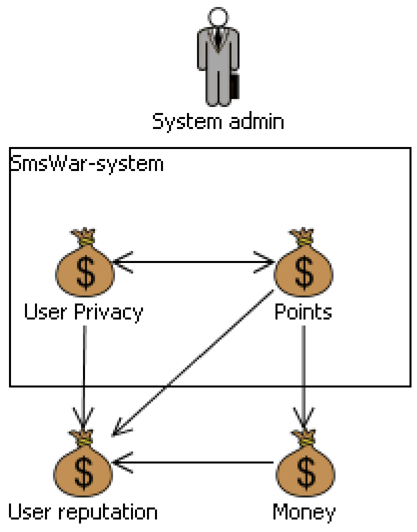


Diagram 35: Asset diagram.

Step 3: Approval

In step 3, the client approved the target descriptions and asset descriptions which were presented by the analysts. Furthermore, the assets were ranked according to importance, likelihood scales were defined and risk evaluation criteria were defined for each asset.

Asset ranking

The following list ranks the assets according to importance:

1. User reputation
2. Money
3. Points
4. Privacy

Consequence Scales

Money

Consequence value (Money)	Description
Catastrophic	>=1 00 EUR is lost
Minor	1 – 99 EUR is lost

Insignificant	0 EUR is lost

Points

Consequence value (Points)	Description
Catastrophic	>=1 000 points is lost
Major	1– 999 points is lost
Insignificant	0 points is lost

Privacy

Consequence value (Privacy)	Description
Catastrophic	Bank account numbers and passwords are lost.
Major	Names of players and phone numbers
Insignificant	Nothing is lost

User reputation

Consequence value (User reputation)	Description
Catastrophic	Complete dissatisfaction with game
Major	Dissatisfaction with game play
Minor	Thinks it's too expensive to play the game.
Insignificant	Satisfaction with game.

Likelihood scale

The following table contains the likelihood scale which were used to describe all likelihoods in the diagram.

Likelihood value	Description
Certain	Ten times or more per 10 games (10-*:10 games)
Likely	Five to nine times per 10 games (5-9:10 games)
Possible	Two to four times per 10 games (2-4:10 games)
Unlikely	One time per 10 games (1:10 games)
Rare	Never

Risk evaluation criteria

Risk evaluation criteria were defined for each asset. These criteria were listed in risk matrices which defines whether a risk is acceptable or not for a specific asset.

Money

		Consequence		
		Insignificant	**Minor**	**Catastrophic**
Frequency	**Rare**	Acceptable	Acceptable	Acceptable
	Unlikely	Acceptable	Acceptable	Must be evaluated
	Possible	Acceptable	Must be evaluated	Must be evaluated
	Likely	Must be evaluated	Must be evaluated	Must be evaluated
	Certain	Must be evaluated	Must be evaluated	Must be evaluated

Points

		Consequence		
		Insignificant	**Major**	**Catastrophic**
Frequency	**Rare**	Acceptable	Acceptable	Acceptable
	Unlikely	Acceptable	Acceptable	Must be evaluated
	Possible	Acceptable	Must be evaluated	Must be evaluated
	Likely	Must be evaluated	Must be evaluated	Must be evaluated
	Certain	Must be evaluated	Must be evaluated	Must be evaluated

Privacy

		Consequence		
		Insignificant	**Major**	**Catastrophic**
Frequency	**Rare**	Acceptable	Acceptable	Must be evaluated
	Unlikely	Acceptable	Must be evaluated	Must be evaluated
	Possible	Must be evaluated	Must be evaluated	Must be evaluated
	Likely	Must be evaluated	Must be evaluated	Must be evaluated
	Certain	Must be evaluated	Must be evaluated	Must be evaluated

User Reputation

	Consequence			
	Insignificant	**Minor**	**Major**	**Catastrophic**

Frequency	**Rare**	Acceptable	Acceptable	Acceptable	Acceptable
	Unlikely	Acceptable	Acceptable	Acceptable	Must be evaluated
	Possible	Acceptable	Acceptable	Must be evaluated	Must be evaluated
	Likely	Acceptable	Must be evaluated	Must be evaluated	Must be evaluated
	Certain	Must be evaluated	Must be evaluated	Must be evaluated	Must be evaluated

Step 4: Risk Identification

Threats, vulnerabilities, threat scenarios and unwanted incidents were defined through a structured brainstorming session. Diagram 36 contains the results of this brainstorming. We have included deliberate actions, accidental actions and non-human threats in the same diagram.

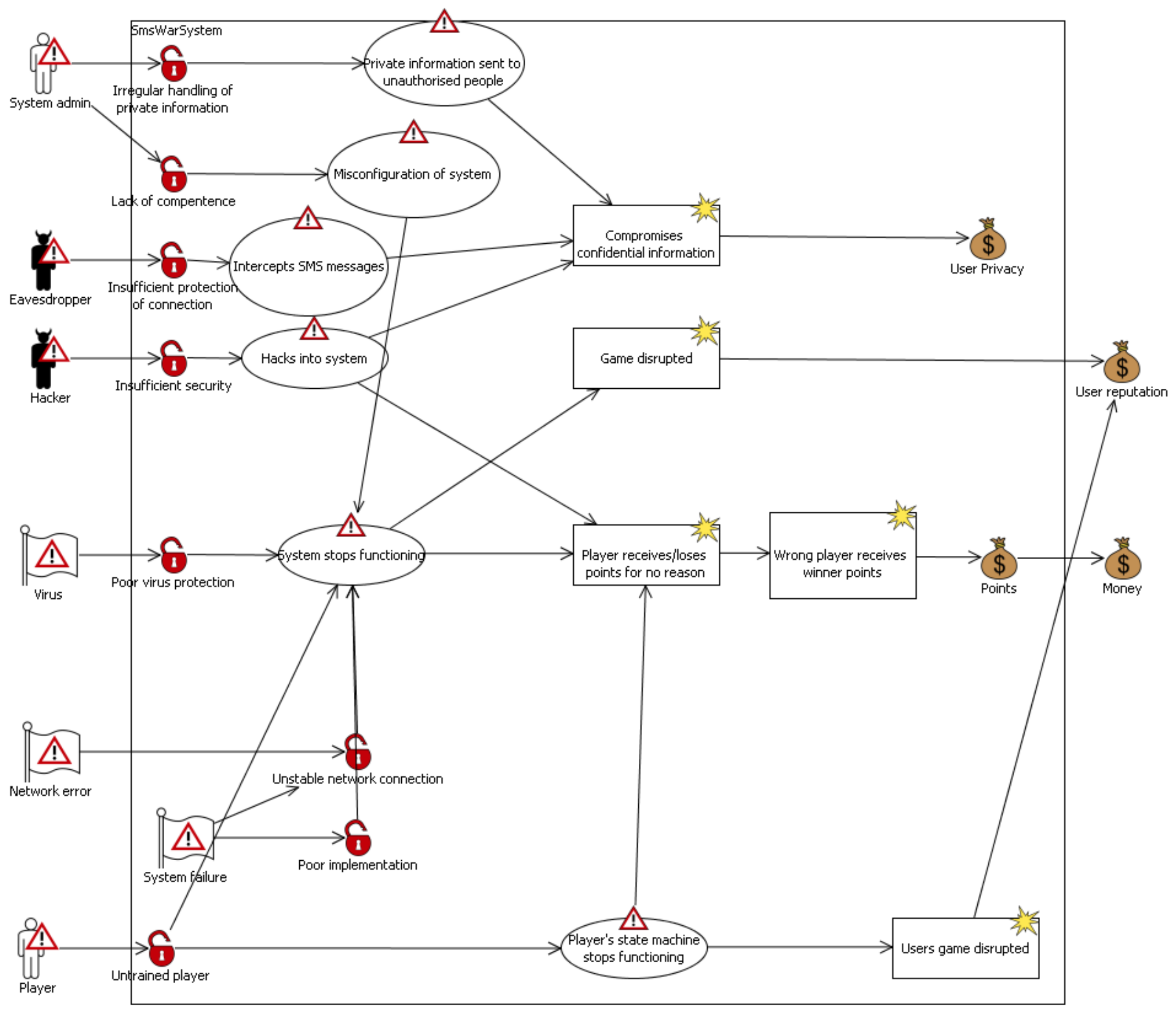

SmsWarSystem
System admin
Irregular handling of private information
Private information sent to unauthorised people
Lack of compentence
Misconfiguration of system
Eavesdropper
Insufficient protection of connection
Intercepts SMS messages
Compromises confidential information
User Privacy
Hacker
Insufficient security
Hacks into system
Game disrupted
User reputation
Virus
Poor virus protection
System stops functioning
Player receives/loses points for no reason
Wrong player receives winner points
Points
Money
Network error
Unstable network connection
System failure
Poor implementation
Player
Untrained player
Player's state machine stops functioning
Users game disrupted

Diagram 36: Threat diagram containing deliberate actions, accidental actions and non-human threats.

Step 5: Risk Estimation

In step 5, likelihood estimates were added to all the threat scenarios, and unwanted incident likelihoods were based on these. Each asset was also given a consequence estimate. Diagram 37 describes the output from step 5, the threat diagram which contains the likelihoods and consequence estimates.

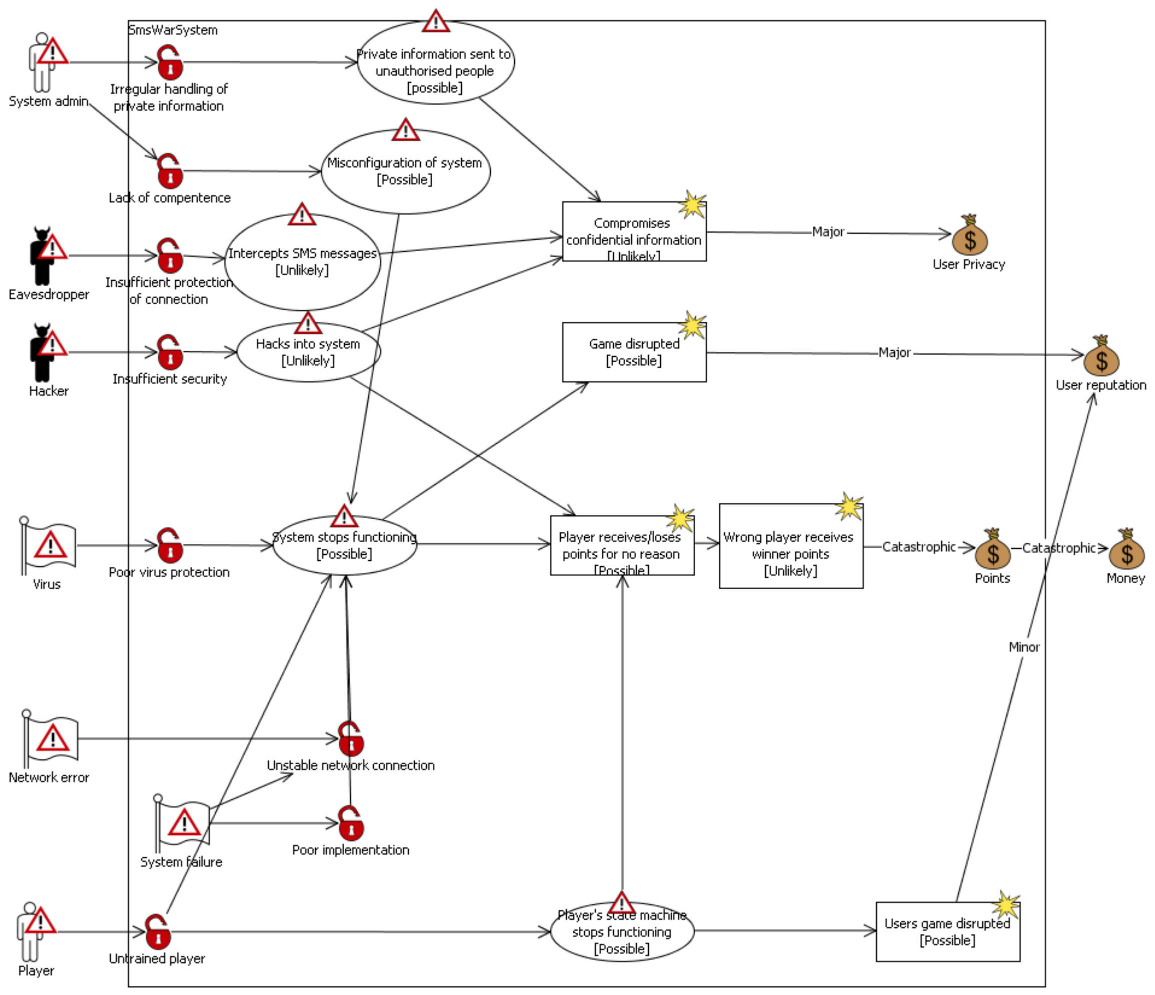

SmsWarSystem
System admin
Irregular handling of private information
Private information sent to unauthorised people [possible]
Lack of compentence
Misconfiguration of system [Possible]
Eavesdropper
Insufficient protection of connection
Intercepts SMS messages [Unlikely]
Compromises confidential information [Unlikely]
Major
User Privacy
Hacker
Insufficient security
Hacks into system [Unlikely]
Game disrupted [Possible]
Major
User reputation
Virus
Poor virus protection
System stops functioning [Possible]
Player receives/loses points for no reason [Possible]
Wrong player receives winner points [Unlikely]
Catastrophic
Points
Catastrophic
Money
Minor
Network error
Unstable network connection
System failure
Poor implementation
Player
Untrained player
Player's state machine stops functioning [Possible]
Users game disrupted [Possible]

Diagram 37: Threat diagram with likelihoods and consequence estimates.

Step 6: Risk Evaluation

In this step we evaluated all the risks to conclude whether they were acceptable or not. Diagram 38 contains an overview of the identified risks and whether they are acceptable or not. Acceptance of the risks were based on the risk evaluation criteria and the relevant risk matrices are given in the following sections.

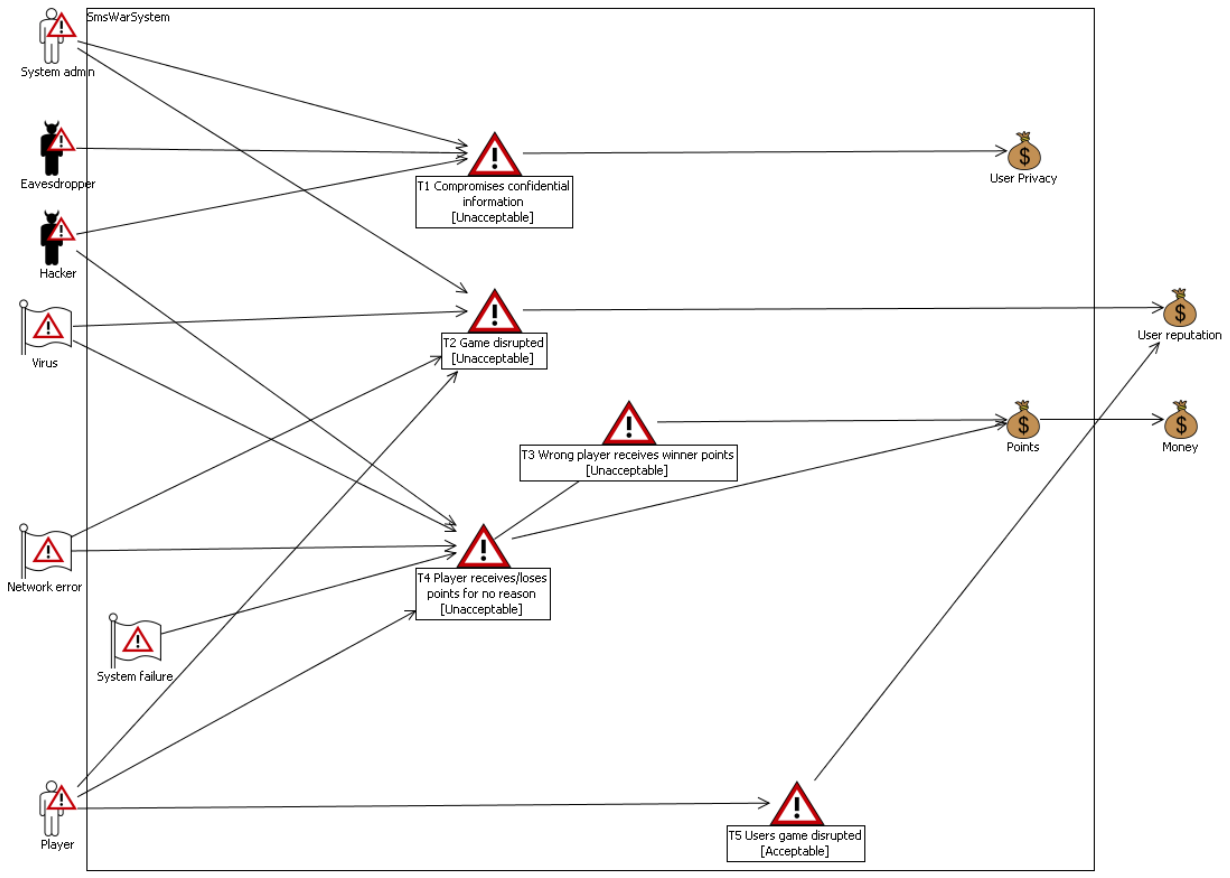

SmsWarSystem
System admin
Eavesdropper
Hacker
Virus
Network error
System failure
Player
T1 Compromises confidential information [Unacceptable]
T2 Game disrupted [Unacceptable]
T3 Wrong player receives winner points [Unacceptable]
T4 Player receives/loses points for no reason [Unacceptable]
T5 Users game disrupted [Acceptable]
User Privacy
User reputation
Points
Money

Diagram 38: Risk overview diagram.

Assets: Money and Points

The following table contains the evaluation of the risks which affect money and points. We chose to use one table since the scales are the same for both assets.

		Consequence		
		Insignificant	Minor	Catastrophic
Frequency	Rare			
	Unlikely			🟥 T3
	Possible		🟥	🟥 T4
	Likely	🟥	🟥	🟥
	Certain	🟥	🟥	🟥

Asset: User Privacy

		Consequence		
		Insignificant	**Major**	**Catastrophic**
Frequency	**Rare**			🟥
	Unlikely		🟥 T1	🟥
	Possible	🟥	🟥	🟥
	Likely	🟥	🟥	🟥
	Certain	🟥	🟥	🟥

Asset: User Reputation

		Consequence			
		Insignificant	**Minor**	**Major**	**Catastrophic**
Frequency	**Rare**				
	Unlikely				🟥
	Possible		T5	🟥 T2	🟥
	Likely		🟥	🟥	🟥
	Certain	🟥	🟥	🟥	🟥

Step 7: Risk Treatment

During a workshop with the client a few treatments were identified and added to the threat diagrams.

Treatments

Diagram 39 shows the various treatments which were identified. The following sections contains a cost/benefit analysis for each of them.

Better procedures for handling private data

This would entail sending the administrator to a training program to enlighten him on aspects of private data. Such a program is quite cheap and would be a one time cost. Hence it has a high cost/benefit ratio and should be implemented.

Training program

This would also entail sending the administrator to a training program to strengthen his competence on system administrator tasks to for instance heighten the security of the system. Such a program is quite cheap and would be a one-time cost. Hence it has a high cost/benefit ratio and should be implemented.

Upgrade firewalls

Upgrading a firewall can be quite expensive but it is also a one-time cost. Even though a new firewall would improve security, the current firewall is still quite secure. Hence the cost/benefit ratio would be limited. This treatment won't be implemented.

Upgrade virus protection

Upgrading the virus protection is quite inexpensive and leads to a heightened security level of the system. Since, new viruses occur very often, it is very important to update the virus protection regularly. This treatment will have a high cost/benefit ratio and will be implemented.

Improve source code

It is very unclear how much this would actually improve the source code and the unassailability of the system. Hence, it is not very clear how much benefit one would get from using a certain amount of money on improving the source code. Hence we assume the cost/benefit ratio won't be very high and will not be implemented.

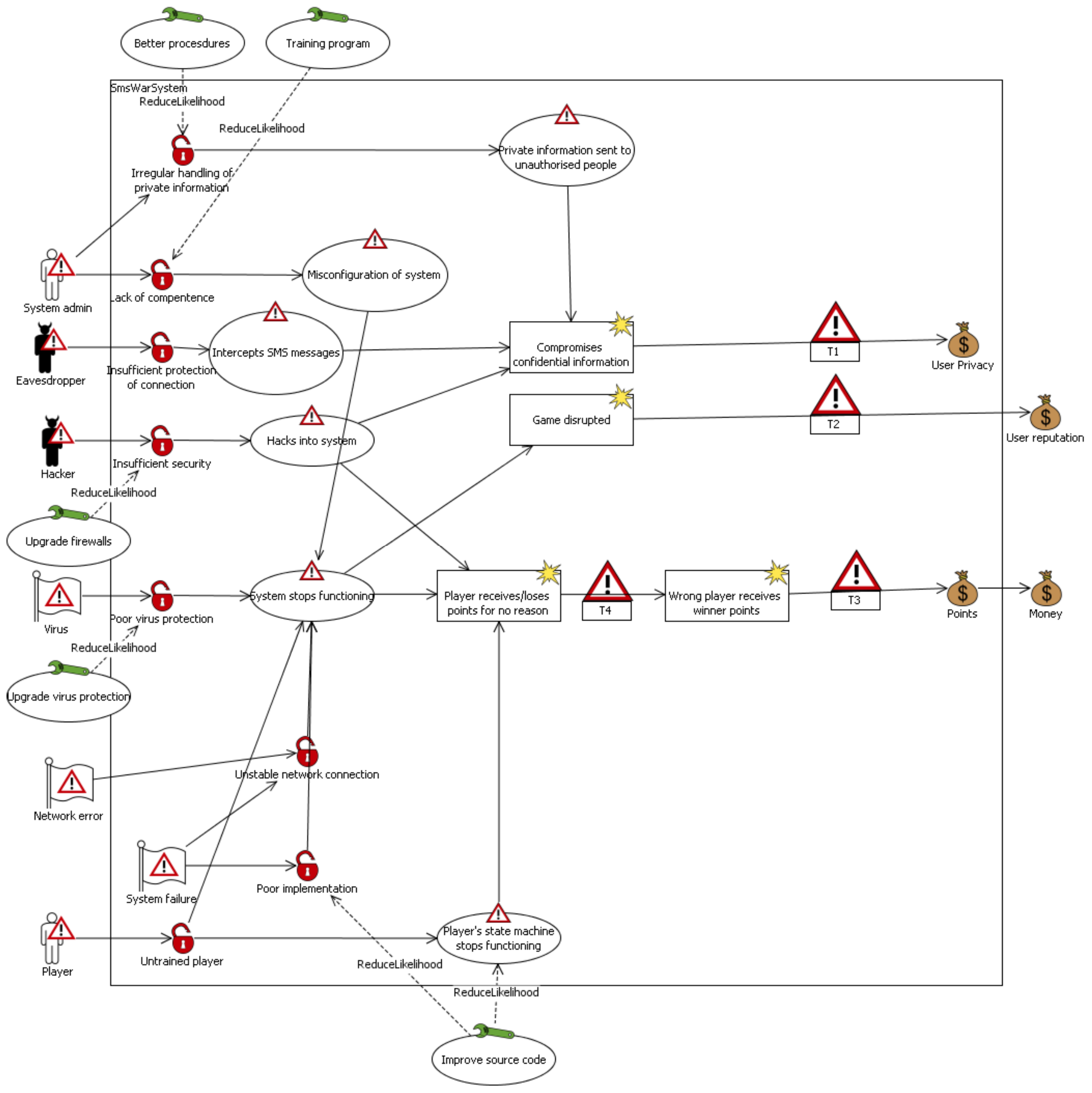


Diagram 39: Treatment diagram.

Treatment Overview

Diagram 40 shows an overview of the treatments and can be used when the treatments should be presented to the clients or other interested parties.

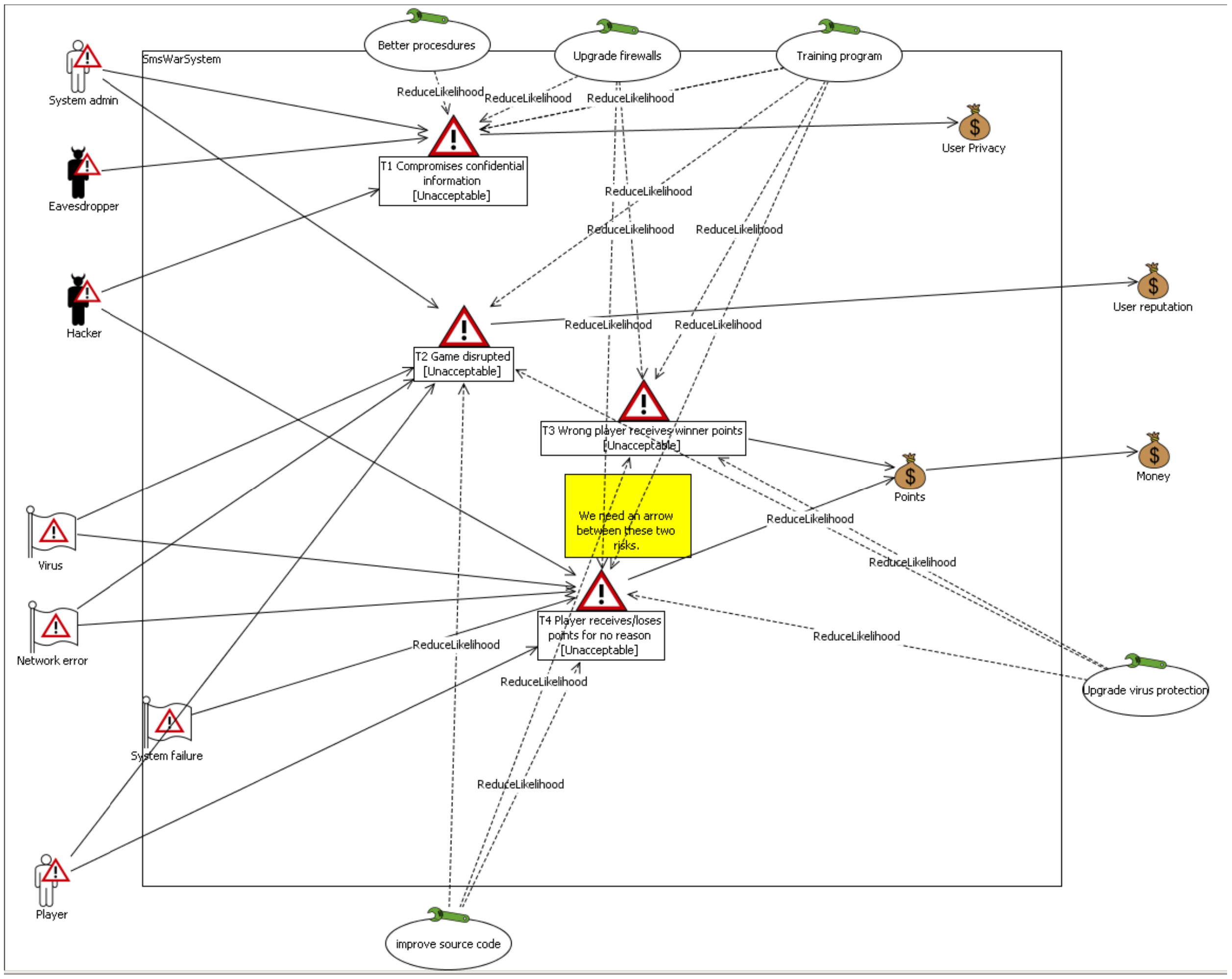

SmsWarSystem
Better procesdures
Upgrade firewalls
Training program
System admin
Eavesdropper
Hacker
Virus
Network error
System failure
Player
improve source code
Upgrade virus protection
User Privacy
User reputation
Points
Money
T1 Compromises confidential information [Unacceptable]
T2 Game disrupted [Unacceptable]
T3 Wrong player receives winner points [Unacceptable]
T4 Player receives/loses points for no reason [Unacceptable]
We need an arrow between these two risks.
ReduceLikelihood

Diagram 40: Treatment overview diagram.

Summary

This report contains a security analysis of the SMSWar system developed in the class INF 5150. A CORAS security analysis approach was used. The previous sections contain the output from each of the seven steps which has to be carried out in a CORAS analysis. We ended up with recommending implementing three treatments to improve the security of the system:

- Upgrade virus protection software
- Send the administrator to a class to learn about handling sensitive data.
- Send the administrator to a class to increase his system administrator skills.