ივ. ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

მორის ვარდუაშვილი

მრავალმომხმარებლიანი სოციალური თამაშების პლატფორმა

Multiplayer Social Gaming Platform

სამაგისტრო ნაშრომი შესრულებულია მაგისტრის აკადემიური ხარისხის მოსაპოვებლად ინფორმაციულ ტექნოლოგიებში

ხელმძღვანელი: ტექ. მეცნ კანდიდატი მანანა ხაჩიძე

თბილისი, 2014

ანოტაცია

ნაშრომში წარმოდგენილია მრავალმომხმარებლიანი სოციალური თამაშების ახალი პლატფორმა. განხილულია ის მიზეზები, რამაც ახალი პლატფორმის შექმნა განაპირობა და ასევე მისი გამოყენების სფეროები. აღწერილია ის ძირითადი საკითხები, რომლებსაც აუცილებელია, რომ ახალი პლატფორმა აკმაყოფილებდეს. ნაჩვენებია მრავალმომხმარებლიანი თამაშების შემუშავების ორი განსხვავებული მოდელი და ამ მოდელებიდან სოკეტებზე დამყარებული მოდელის უპირატესობაა დამტკიცებული. ახალი პლატფორმის, არამარტო თამაშებში, არამედ სხვა სფეროებში გამოყენების მაგალითებია მოყვანილი, თუმცა აღნიშნული პლატფორმის პროფილი უშუალოდ სოციალური თამაშებია. ნაშრომში გაანალიზებულია ბაზარზე არსებული მსგავსი ტიპის სხვა პროდუქტები. შედარებულია მათი პროგრამული უზრუნველყოფის სიმარტივე და მოქნილობა თამაშებისთვის, ასევე მათი სალიცენზიო ღირებულებაც, რომელმაც საჭიროება წარმოშვა, შექმნილიყო უფასო პლატფორმა, რომელიც ვფიქრომ უფრო პოპულარული გახდება მცირებიუჯეტიანი პროექტებისთვის.

ნაშრომში აღწერილია სოციალური თამაშების ახალი პლატფორმის საბაზისო ელემენტები, რომლებიც აუცილებელია მისი მუშაობის პრინციპის გასაგებად. პლატფორმის პროგრამული უზრუნველყოფა შეიცავს წინასწარ დამუშავებულ იმ ელემენტებს, რომლებიც ბევრად მარტივს ხდის უშუალოდ თამაშების დეველოპმენტს. ასევე გათვალისწინებულია ის საკითხები, რომლებიც ან არ აქვს სხვა პროდუქტებს, ან ძალიან მოუხერხებელია გამოსაყენებლად.

აღნიშნული პლატფორმის გამოყენებით, ნაშრომს თანდართული აქვს სადემონსტრაციო მაგალითი, სადაც გამოყენებულია თამაშებისთვის დამუშავებული სხვადასხვა ელემენტები. არსებული მაგალითით ნაჩვენებია ახალი პლატფორმის უპირატესობა სხვა პლატფორმებთან შედარებით.

ANNOTATION

The article presents a new platform of multiplayer social gaming. The article discusses the reasons which led to the creation of new platforms and also its application fields. There is described the main issues that need to comply with the new platform. The article shows two different models of development of multiplayer games and is approved the advantage of the models based on the model sockets. Here are given examples applicable not only in gaming but also in other fields, but the profile of the mentioned platform is directly the Social Games. The article analyzes the market of the same type other products. It is compares their software simplicity and the flexibility for games, also value of their license, what created the necessity of the creation of free platform, and I think this one will become more popular in low-budget projects.

The article describes the basic elements of social games for new platforms, which are necessary understanding its working principle. Platform software contains such pre-treated elements, which makes games development simpler. Also it envisages issues that either other products do not have them, or are too inconvenient to use.

The demonstrative example is attached on the article with using the mentioned platform, where is used various kind of processed elements. Given platform shows the advantages of the new platform compared to other platforms.

სარჩევი

შესავალი

მრავალმომხმარებლიანი სოციალური თამაშები დღითი-დღე უფრო პოპულარული და მოთხოვნადი ხდება მსოფლიო ბაზარზე. იგი თანაბრად პოპულარულია თითქმის ყველა ასაკობრივი, თუ სოციალური ჯგუფის ადამიანებში, რასაც მსოფლიო ყოველწლიური მზარდი სტატისტიკა ადასტურებს.

ინფორმაციული ტექნოლოგიების თანდათანობითმა დახვეწამ და არსებულმა მიღწევებმა ბევრად გაამარტივა სოციალური თამაშების წარმოება და ბევრად პოპულარული ხდება თანამედროვე სპეციალისტებში. ამ სფეროს მაღალმა მოთხოვნადობამ განაპირობა, ის რომ სოციალური თამაშები ბოლო წლებში დამოუკიდებელ მიმართულებად ჩამოყალიბდა ინფორმაციული ტექნოლოგიების სფეროში, რომელიც მსოფლიოს არაერთ წამყვან უნივერსიტეტებში, როგორც ინფორმაციული ტექნოლოგიების სპეციალობის ცალკე ქვემიმართულებადაა წარმოდგენილი.

ვირტუალური სოციალური თამაშების ისტორია ჯერ კიდევ 1970 წლიდან იწყება. თუმცა ძირითადი გავრცელება უკვე 1990 წლიდან დაიწყო და განსაკუთრებული პოპულაროზა მოიპოვა. ინტერნეტის საყოველთაო გავრცელებამ კი უკვე შექმნა საჭიროება მრავალმომხმარებლიანი ონლაინ თამაშების შექმნის. ამ პერიოდიდან უკვე გამოჩნდნენ ბაზარზე ნოვატორებიც, რომლებიც მომხმარებლებს სხვადასხვა ტიპის თამაშებს სთავაზობდნენ და დაიწყო ბუმი: მომხმარებლები ყიდულობდნენ დისკებს და ა.შ. ეს აძლევდა ადამიანებს ვირტუალურ ინდივიდუალურობას და ასევე ერთად ონლაინ რეჟიმში საერთო ამოცანების, თუ დავალებების გადაჭრის საშუალებას.

ვირტუალური თამაშების კონტექსტში ნელ-ნელა დაიწყო სხვადასხვა ტექნოლოგიების დახვეწა, რომლებიც აქტიურად გამოიყენება ინფორმაციულ ტექნოლოგიებში ბევრი სხვა მიმართულების სამუშაოს სრულყოფილად შესასრულებლად და მაქსიმალურად ხარისხიანი შედეგის მისაღებად.

1. ვირტუალური სოციალური სამყარო

ვირტუალური სოციალური სამყარო წარმოადგენს ციფრულ გარემოს, რომელიც მსგავსია რეალური სამყაროსი. მრავალ მომხმარებელს შეუძლია შევიდეს ამ ვირტუალურ სამყაროში, განახორციელოს სხვადასხვა ტიპის აქტივობები. მაგ: დაათვალიეროს ქალაქი, დაუკავშირდეს სხვა მომხმარებლებს, აწარმოოს მიმოწერა, ერთად განახორციელონს სხვადასხვა იდეები, დაეხმარონ თამაშში და ა.შ. ეს ყველაფერი აერთიანებს ვირტუალურ სამყაროს. თუმცა პრაქტიკაში ძირითადად ზემოთჩამოთვლილი მაგალითები ცალ-ცალკეა რეალიზებული, ან ლოგიკურად ჯგუფ-ჯგუფად, თუმცა მათი იდეა და მუშაობის პრინციპი, ვირტუალური სოციალური სამყაროს ძირითად კონცეფციაში ჯდება.

1.1 გამოყენების სფეროები

ვირტუალური სოციალური სამყაროს გამოყენების რამდენიმე ძირითად სფერო შეგვიძლია გამოვყოთ:

ვირტუალური სამყარო ბიზნესისთვის

ვირტუალური სამყარო იძლევა საშუალებას მომხმარებლების რეალურ რეჟიმში დაკავშირებისა. შესაძლებელია მომხმარებლებმა გაცვალონ რეალურ რეჟიმში ერთმანეთში ინფორმაცია, აწარმოონ მიმოწერა, ჩაატარონ ონლაინ კონფერენცია ან სხვადასხვა შეკრება, რომელიც მსოფლიოს სხვადასხვა წერტილში მყოფ ადამიანებს ერთმანეთთან რეალურ რეჟიმში შეხვედრის საშუალებას მისცემს.

ვირტუალური სამყარო განათლებისთვის

ვირტუალური სამყარო სტუდენტებს და მოსწავლეებს აძლევს საშუალებას ჩაატარონ სხვადასხვა ტიპის მოდელირებული ექსპერიმენტები. მათ შეუძლიათ ჩაატარონ ვირტუალური ცდები, რომლებიც ბევრად ნაკლები რესურსის გამოყენებით, უფრო მაღალ

შედეგს იძლევა. ასევე სტუდენტებმა კავშირი იქონიონ რეალურ რეჟიმში მასწავლებელთან, გაცვალონ დავალებები, ან თუნდაც დისტანციურ რეჟიმში განახორციელონ სწავლება.

ვირტუალური სამყარო თამაშებისთვის

ვირტუალური სამყარო მომხმარებლებს აძლევს საშუალებას ეთამაშონ ერთმანეთს რეალურ რეჟიმში სხვადასხვა ტიპის თამაშები. ერთად განახორციელონ მისიები, ან ერთად ეთამაშონ სხვა გუნდს. მათ შეუძლიათ თამაშებში სხვადასხვა ტიპის გარემო შეიძლება შექმნან და მოირგონ საკუთარ თავზე.

2. ვირტუალური სოციალური თამაშების ტექნიკური რეალიზაცია

ვირტუალური სოციალური თამაშები ძირითადად ითვალისწინებს სხვადასხვა მომხმარებლების ერთმანეთთან დაკავშირებას. დაკავშირებისთვის სხვადასხვა მეთოდების გამოყენებაა შესაძლებელი და ასევე თვითონ თამაშების რეალიზაციისთვის სხვადასხვა მოდელები შეიძლება გამოვიყენოთ.

2.1 ქსელი, პაკეტები და პროტოკოლები

2.1.1 ქსელი

ვირტუალური სოციალური თამაშებიშთვის მომხმარებელთა დასაკავშირებლად აუცილებელია კომპიუტერული ქსელი, რომელიც შედგება კავშირის არხით ერთმანეთთან დაკავშირებული ციფრული მანქანებისგან: ჰოსტებისგან და მარშრუტიზატორებისგან. ჰოსტი არის კომპიუტერი რომელზეც გაშვებულია აპლიკაცია როგორც ვებ ბრაუზერი, ან მონაცემთა გაზიარების პროგრამა და ა.შ. მარშრუტიზატორი არის მანქანა, რომლის

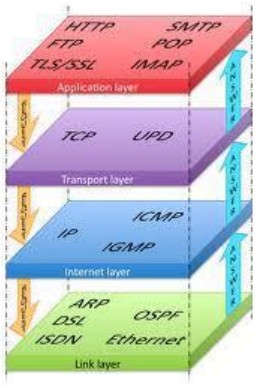
დანიშნულებაცაა ან გაატაროს, ან გადაამისამართოს ინფორმაცია ერთი კავშირის არხიდან სხვა არხზე. კავშირის არხით ხდება ბაიტების გაცვლა ერთი ჰოსტიდან მეორეზე.

2.1.2 პაკეტები

ინფორმაცია არის ბაიტების გარკვეული სახის მიმდევრობა. ამ ინფორმაციის დამუშავება ხდება ქსელში გადასაცემად, იგი იყოფა პაკეტებად. თითოეული პაკეტი შეიცავს გადა-საცემ ინფორმაციას, საწყისი წერტილის შესახებ ინფორმაციას და დანიშნულების წერტილს. როუტერი კი კითხულობს პაკეტის საკონტროლო ინფორმაციას და შესაბამისად ამისამართებს ქსელში.

2.1.3 პროტოკოლები

პროტოკოლი წარმოადგენს გარკვეული წესების ერთობლიობას, რომლის მიხედვითაც ხდება ქსელში ინფორმაციის გაცვლა. პროტოკოლი აწესრიგებს, თუ როგორ უნდა იყოს კონსტრუირებული პაკეტი. სად უნდა ეწეროს პაკეტში დანიშნულების შესახებ ინფორმაცია, მისი ზომა და ა.შ. პროტოკოლები სხვადასხვა ტიპის არსებობს და თითოეულს კონკრეტული დანიშნულება აქვს. აქედან ზოგიერთს ქვემოთ განვიხილავთ.

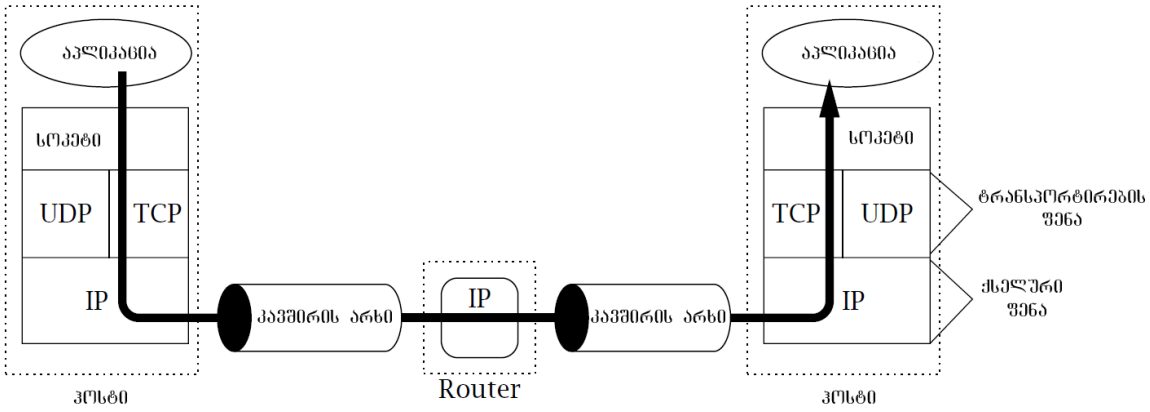


TCP პროტოკოლი

ქსელში ინფორმაციის გადაცემა ხორციელდება პაკეტების საშუალებით, მათ მარშრუტიზაციაზე ქსელში ზრუნავს IP, იგი ყოველთვის ცდილობს მონახოს ოპტიმალური გზა პაკეტის დანიშნულების ადგილამდე გადასცემად. თუმცა ქსელში შესაძლოა გადაცემული პაკეტების მიმდევრობიდან, ზოგმა შესაძლოა დანიშნულების ადგილამდე ვერ

მიაღწიოს, ან გზაში დაზიანდეს, ამიტომ გადაწყდა პროტოკოლების სტეკში IP ფენის ზემოთ განთავსებულიყო პროტოკოლის სხვა ფენა TCP(Transmission Control Protocol), რომელიც კავშირის ბოლოს დაადასტურებს პაკეტის მიღებას, ან მოითხოვს მის ხელახლა გადმოცემას. გარდა ამისა TCP პაკეტების სწორი თანმიმდევრობით მიღების საშუალებას იძლევა. IP და TCP გამოყენება ძირითადად ხდება ერთად, როგორც TCP/IP, ანუ TCP არის მაღალი დონის პროტოკოლი, რომელიც იყენებს უფრო დაბალი დონის IP პროტოკოლს.

სქემაზე ნაჩვენებია ოთხფენიანი მოდელი, რომელიც ხშირად გამოიყენება. თითოეული ფენის მოდელი არის სხვადასხვა დონის აბსტრაქცია. ორ აპლიკაციას შორის კავშირი ქსელის საშუალებით ამ ფენების გავლით ხორციელდება. ინფორმაციის გადაცემისას ფენების ჩავლა ხდება ზემოდან ქვემოთ. IP შემდგომ უკვე ფიზიკური კავშირის არხით და მიღება კი უკუ მიმართულებით ქვემოდან ზემოთ და აპლიკაცია უკვე იღებს საწყის ფორმაში დაბრუნებულ ინფორმაციას.



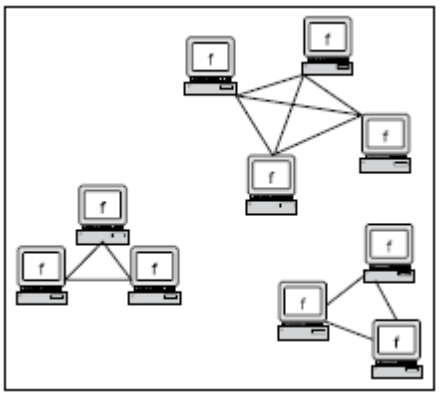
UDP პროტოკოლი

რადგან TCP პროტოკოლის დროს ხდება შემოწმება პაკეტი ხომ არ დაიკარგა და მისი ხელახალი მოთხოვნა, შესაბამისად ეს უფრო ნელი გადაცემაა და ზოგ შემთხვევაში არახელსაყრელია. შეიძლება ინფორმაციის გადაცემის სიჩქარე ბევრად უფრო მნიშვნელოვან

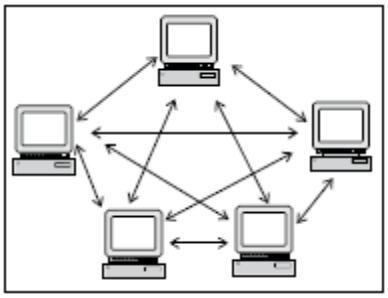
იყოს, ვიდრე რომელიმე პაკეტის დაკარგვა. მაგალითად, როდესაც ხდება აუდიო ან ვიდეო ნაკადის გადაცემა, ამ შემთხვევაში კრიტიკული მნიშვნელობა არ აქვს TCP პროტოკოლში არსებულ შემოწმებებს და ამიტომ გამოიყენება UDP (User Datagram Protocol) პროტოკოლი. მნიშვნელოვანია ის ფაქტორები, რომ ეს პროტოკოლი არ იძლევა გარანტიას რომ პაკეტი აუცილებლად მიაღწევს დანიშნულების წერტილს და ასევე პაკეტი გადაცემული იქნება თუ არასწორი მიმართულებით. UDP არ უზრუნველყოფს დაკარგული პაკეტის ხელახალ გადაცემას და არც გადამცემისთვის შეტყობინებას გადაცემული პაკეტის დაკარგვის შესახებ.

Peer-to-Peer პროტოკოლი

Peer-to-Peer საშუალებით უზრუნველყოფს კლიენტების ურთიერთკავშირს ქსელში. ასეთ ქსელში ყოველი კლიენტი წარმოადგენს თანაბრამნიშვნელოვან კვანძს, რომელსაც შეუძლია მიიღოს და გასცეს ინფორმაცია. ასეთი მოდელი ჰგავს კლიენტ-სერვერულ მოდელს, მაგრამ ამ შემთხვევაში მესამე კომპიუტერი აღარ არის საჭირო, კლიენტებს შორის ინფორმაციის გასაცვლელად, არამედ უშუალოდ კლიენტებს შორის ხდება ინფორმაციის გაცვლა. ამ პროტოკოლით კავშირის გამოყენება ძალიან მოსახერხებელია მრავალმოხმარებლიან სისტემებში, სადაც მომხმარებლები სხვადასხვა არც ისე დიდ ჯგუფებად იყოფიან.

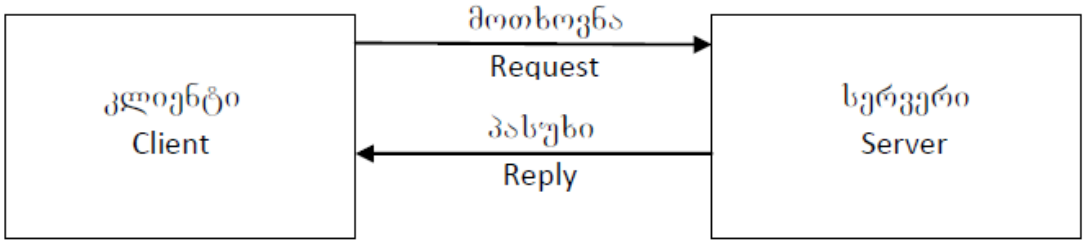


ასეთი კავშირის შემთხვევაში მთლიანი ლოგიკა განთავსებული უნდა იყოს ყველა კომპიუტერზე, რაც არც თუ ისე სახარბიელოა. თითოეულ კლიენტს წვდომა უჩნდება კრიტიკულ მონაცემებზე, რომელთა შეცვლაც შეუძლიათ და სხვა მომხმარებლისთვის მცდარი ინფორმაციის მიწოდებაც შესაძლებელია, რაც უკვე ქმნის უსაფრთოხების კუთხით სერიოზულ პრობლემას.



2.2 კლიენტ-სერვერული მოდელი

კლასიკურ კლიენტ-სერვერულ მოდელში, კლიენტი აგზავნის მოთხოვნას სერვერზე, სერვერი ამუშავებს ამ მოთხოვნებს და შემდეგ უბრუნებს პასუხს ისევ კლიენტს. კავშირის ინიციატორი არის კლიენტი, ხოლო სერვერი პასიურ რეჟიმში ელოდება კლიენტის მოთხოვნას. პირობები მოთხოვნისა და პასუხის დამოკიდებულია სხვადასხვა პარამეტრებზე. კლიენტ-სერვერული მოდელის მარტივი მაგალითია, როდესაც ბრაუზერი აგზავნის მოთხოვნას ვებსერვერზე, ვებსერვერი ამუშავებს მოთხოვნას და შესაბამისი პასუხის გენერაციას ახდენს, შემდეგ ეს პასუხი უბრუნდება კლიენტს და მისი ასახვა ხდება უკვე ბრაუზერში. განსხვავებით სერვერისგან კლიენტმა უნდა იცოდეს დასაკავშირებლად სერვერის მისამართი და პორტი, მაგრამ არა პირიქით.



გარდა ამისა, იმ სერვერების კლასიფიკაცია, რომლებიც კლიენტისგან იღებენ მოთხოვნებს შეიძლება ორ საბაზისო ტიპად მოვახდინოთ. სერვერებირომელიც მხოლოდ ერთ კლიენტს ემსახურება რეალურ დროში. მაგრამ თუ რამდენიმე კლიენტი უგზავნის მოთხოვნას პარალელურ რეჟიმში, ერთსა და იმავე დროში, მაშინ რიგ-რიგობით ხდება მათი მომსახურება.სერვერები, რომლებიც ერთსა და იმავე დროში რამდენიმე კლიენტს ემსახურება. როგორც წესი, ეს ხდება ახალი პროცესის შექმნის საშუალებით სერვერზე მოთხოვნის მიღებისას – ორიგინალი პროცესი უბრუნდება ახალი მოთხოვნის მომლოდინე რეჟიმს.

გარდა კლიენტ-სერვერული მოდელისა ხშირ შემთხვევაში გვხვდება შეტყობინების გასაგზავნი სერვისები, რომლის დროსაც ინფორმაციისი გადაცემა ხდება ორი კლიენტს შორის (მაგ: ჩატები და ა.შ.)ამას ეწოდება peer-to-peer (P2P) გადაცემა. როგორც წესი ორი კლიენტის ერთმანეთთან დაკავშირება ხდება ისევ სერვერის საშუალებით, ხოლო შემდგომ ხდება უკვე ინფორმაციის P2P-ს საშუალებითმიმოცვლა.

2.3 მისამართები

სანამ პროგრამა დაუკავშირდება სხვა პროგრამას, საჭიროა მოხდეს დასაკავშირებელი პროგრამის ქსელში იდენტიფიცირება. TCP/IP დროს პროგრამის იდენტიფიცირებისათვის საჭიროა მისამართი, რომელსაც იყენებს IP და პორტის ნომერი.

ინტერნეტ მისამართი არის ბინარული რიცხვი. ორი სტანდარტი არსებობს: IPv4 და IPv6. IPv4მისამართი არის 32 ბიტიანი. მას შეუძლია 4 მილიარდამდე სხვადასხვა მისამართის

მოცემა, რაც დღეს უკვე აღარ არის საკმარისი და ამიტომ მსოფლიო აქტიურად გადადის IPv6 სტანდარტზე, რომელიც 128 ბიტი სიგრძისაა.

IPv4 ჩაწერა ხდება ათობითი რიცხვებით ოთხ ჯგუფად, რომლებიც დაყოფილია წერტილებით (მაგ. 10.1.2.3) თითოეული ნაწილი ჩანაწერის იკავებს ოთხ ბაიტს და ამიტომ მასში ჩაწერილი რიცხვი არის 0-დან 255-მდე. IP36 ჩაწერა ხდება თექვსმეტობითი სისტემით და მთლიანად იკავებს 16 ბაიტს (მაგ: 2000:fdb8:0000:0000:0001:00ab:853c:39a1), ხოლო თითოეული ნაწილი კი 2 ბაიტს.

2.4 Pooling

ამ ტიპის მოდელით სოციალური თამაშის ან ზოგადად პროგრამის რეალიზაცია ხდება შემდეგნაირად. კლიენტის მხრიდან პერიოდულად სერვერთან იგზავნება მოთხოვნა, იმის შესახე, ხომ არ შეიცვალა სერვერზე რამე, რაიმე მონაცემი ხომ არ განახლდა და შესაბამისად სერვერიდან უბრუნდება პასუხი ან შეიცვალა ან არა.

როდესაც იქმნება თამაში შესაბამისად ეს მოდელი ძალიან არახელსაყრელია, რადგან მუდმივად მოთხოვნის გზავნა სერვერზე იკავებს დიდ დროს, ტრაფიკს, ზრდის სერვერის დატვირთვას. ასევე როდესაც კლიენტის მხრიდან შეიძლება რამდენიმე სერვერზე კავშირი გვჭირდებოდეს, ეს უკვე კლიენტის მხარეს ლოგიკას დაკავშირებებისას ძალიან ართულებს

2.5 სოკეტზე დამყარებული კავშირი

Pooling-სგან განსხვავებით სოკეტითი დამყარებული კავშირი უზრუნველყოფს, რომ კავშირი იყოს მუდმივად გახსნილი, არ სჭირდება სერვერზე პერიოდულად მოთხოვნების გაგზავნა იმის გასაგებად შეიცვალა სერვერზე მონაცემები თუ არა, არამედ სერვერზე მომხდარი ცვლილების შესახებ, სერვერი თვითონვე აგზავნის სოკეტ კავშირის საშუალებით ინფორმაციას კლიენტთან, რის მიხედვითაც კლიენტი ცვლის შესაბამის სურათს. ასეთ შემთხვევაში რაღა თქმა უნდა სერვერზე გაგზავნილი მოთოხვნების რაოდენობა არის

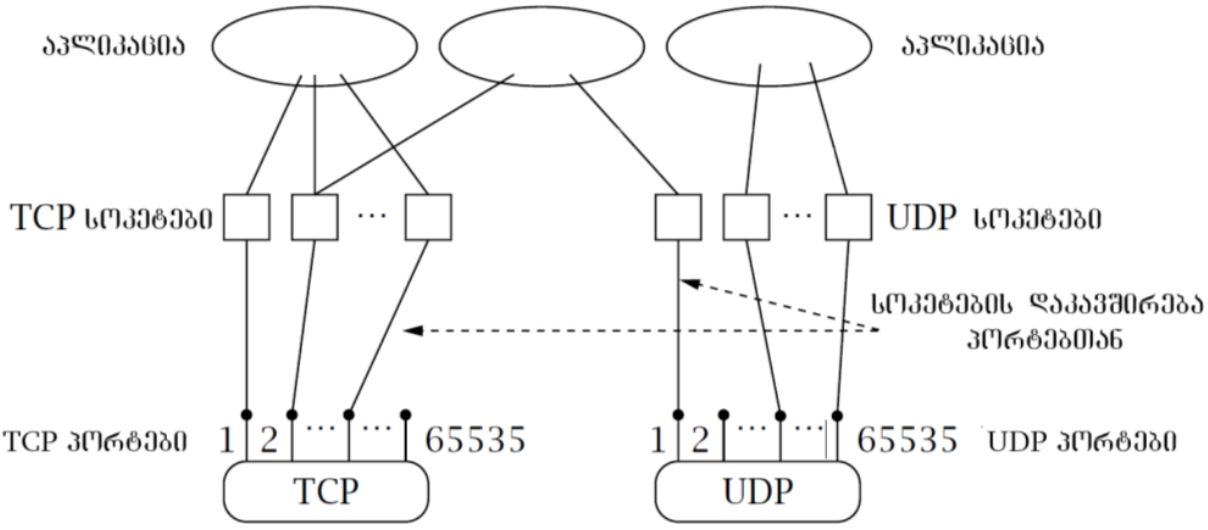
შედარებით ნაკლები, მოთხოვნის გაგზავნის და სერვერიდან პასუხის დროც არის ბევრად ნაკლები და სერვერის დატვირთვაც არასაჭიროებისამებრ არ ხდება.

2.6 სოკეტების გამოყენება ვირტუალურ თამაშებში

2.6.1 სოკეტი

სოკეტი არის აბსტრაქცია, რომლის საშუალებითაც აპლიკაციას შეუძლია ქსელის საშუალებით დაუკავშირდეს სხვა აპლიკაციას დაგააგზავნოს ან მიიღოს მონაცემები. მონაცემების ჩაწერა ხდება სოკეტში, შემდეგ ხდება მათი ბუფერიზაცია, გაგზავნა. ანალოგიურად ხდება მისი წაკითხვა.

სოკეტების სხვადასხვა ტიპის არსებობას განაპირობებს ტრანსპორტირების სხვადასხვა პროტოკოლების არსებობა.



SOCK_STREAM (TCP) ნაკადების გადაცემა ხდება წინასწარ შექმნილი კავშირით.მონაცემთა გადაცემის არხი უზრუნველყოფს ინფორმაციის საიმედოდ გადაცემას, სადაც ფრაგმენტები გაგზავნილი ბლოკის არ იკარგება, არც დუბლირდება და არც გადანაცვლება არ ხდება. ამიტომ სოკეტი ეს ტიპი არის ყველაზე გავრცელებული.ის იყენებს TCP/IP პროტოკოლს. TCP უზრუნველყოფს ინფორმაციის დაუზიანებლად გადაცემას, IP მარშრუტიზაციას ინტერნეტში, ხოლო ინფორმაციის დაუზიანებლობაზე პასუხს არ აგებს.

SOCK_STREAM (UDP) მონაცემთა გადაცემა ხდება ცალკეული შეტყობინებების სახით დატაგრამების. არ საჭიროებს წინასწარი კავშირის დამყარებას, იმიტომ რომ უბრალოდ ხდება პაკეტის აგება, IP ჰედერის ფორმირება მიმღების ინფორმაციით და პაკეტის გაგზავნა. ინფორმაციის გაცვლა ხდება უფრო სწრაფად, მაგრამ არასაიმედოდ. შესაძლებელია დაიკარგოს რაღაც ნაწილი, ან მიმდევრობა შეიცვალოს, ან დადუბლირდეს და ა.შ.იყენებს UDP/IPპროტოკოლს.

SOCK_RAW ეს ტიპი მიეკუთვნება დაბალდონიან სოკეტს. იგი სხვა სოკეტებისგან იმით განსხვავდება, რომ პროგრამას შეუძლია აიღოს თავის თავზე ფორმირება სხვადასხვა ჰედერის, რომელიც ემატება შეტყობინებას. დაბალსაფეხურიანი სამუშაოსთვის IP, ISMP პროტოკოლებთან.

2.6.2 სოკეტის მისამართი

სანამ გადავცემთ მონაცემებს სოკეტის საშუალებით, ჯერ უნდა მოხდეს მისი გამოძახება მისამართით. მისამართი შედგება IP მისამართისა და 16 ბიტიანი პორტის კომბინაციით. IP განსაზღვრავს ჰოსტს ქსელში, პორტი კი კონკრეტულ სოკეტს ამ ჰოსტზე.

2.6.3 სოკეტების გამოყენება

როგორც ზემოთ აღვნიშენთ სოკეტების ორი ტიპია SOCK_STREAM (TCP) და SOCK_STREAM (UDP). ორივე შემთხვევაში საჭიროა კლიენტისა და სერვერის პროგრამის

შექმნა. კლიენტი უზრუნველყოფს სერვერისთვის საჭირო ინფორმაციის მიწოდებას, ხოლო სერვერი გარკვეული პირობებისა და დამუშავების შემდეგ პასუხის გაცემას. განვიხილოთ ორივე ტიპის სოკეტი და თითოეულისთვის კლიენტისა და სერვერის პროგრამა.

2.6.4 TCP სოკეტები

ჯერ განვიხილოთ სოკეტ სერვერი, რომელის მიერაც ხდება მოთხოვნების მიღება და მათი დამუშავება. სერვერის პროგრამა მოიცავს ხუთ ძირითად ეტაპს:

1. ServerSocket ობიექტის შექმნა

   ServerSocket პარამეტრად მიეწოდება პორტის ნომერი და შესაბამისად სერვერი ელოდება კლიენტისგან დაკავშირებას ამ პორტის მიხედვით.

   ServerSocket servSock = new ServerSocket(1234);

2. სერვერი მომლოდინე რეჟიმში გადაყვანა

   სერვერი ელოდება კლიენტისგან დაკავშირებას. ის ამას ახორციელებს ServerSocket კლასის მეთოდით accept() რომელიც აბრუნებს Socket ობიექტს, რომმლითაც კავშირი დამყარდა.

   Socket link = servSock.accept();

3. ინფორმაციის შემავალი და გამომავალი ნაკადები

   Socket კლასის getInputStream და getOutputStream მეთოდებით ხდება შემავალი და გამომავალი ნაკადების წაკითხვა. შემავალი ნაკადის საშუალებით ხდება კლიენტიდან მოსული ინფორმაციის ამიღება, ხოლო გამომავალ ნაკადში კი იწერება კლიენტთან გასაგზავნ ინფორმაცია. ეს ნაკადების ობიექტი შეიძლება გარდაიქმნას სხვა ობიექტად, რომ უფრო მარტივი გახდეს მათგან ინფორმაციის წაკითხვა.

DataInputStream socketIn = new DataInputStream(link. getInputStream());
DataOutputStream socketOut = new DataOutputStream(link. getOutputStream());

4. ინფორმაციის გაგზავნა და მიღება ხდება გამავალ ნაკადში ჩაწერით DataOutputStream klasis write() meTodiT. socketOut.write("SEND DATA");
   ინფორმაციის მიღება კი ხდება შემომავალი ნაკადიდან DataInputStream klasis read() მეთოდის გამოყენებით data = socketIn.read();

5. კავშირის დახურვა კავშირის დახურვა ხდება Socket კლასის close() მეთოდით
   link.close();

სერვერისათვის ინფორმაციის მიწოდება ხდება კლიენტის მიერ, ამიტომ საჭიროა კლიენტის პროგრამის შექმნაც. განვიხილოთ კლიენტის პროგრამა:

1. სერვერთან კავშირის დამყარება
   სერვერთან კავშირის დასამყარებლად იქმნება Socket ობიექტი ორი არგუმენტით:
   სერვერის IP მისამართი (InetAddress saSualebiT)
   პორტის ნომერი, თუ რომელ სერვისს ვუკავშირდებით
   Socket link = new Socket(InetAddress.getLocalHost(),1234);

2. შემავალი და გამომავალი ნაკადები
   Socket კლასის getInputStream და getOutputStream მეთოდებით ხდება შემავალი და გამომავალი ნაკადების წაკითხვა. შემავალი ნაკადის საშუალებით ხდება კლიენტიდან მოსული ინფორმაციის ამოღება, ხოლო გამომავალი ნაკადში კი იწერება კლიენტთან გასაგზავნი ინფორმაცია.

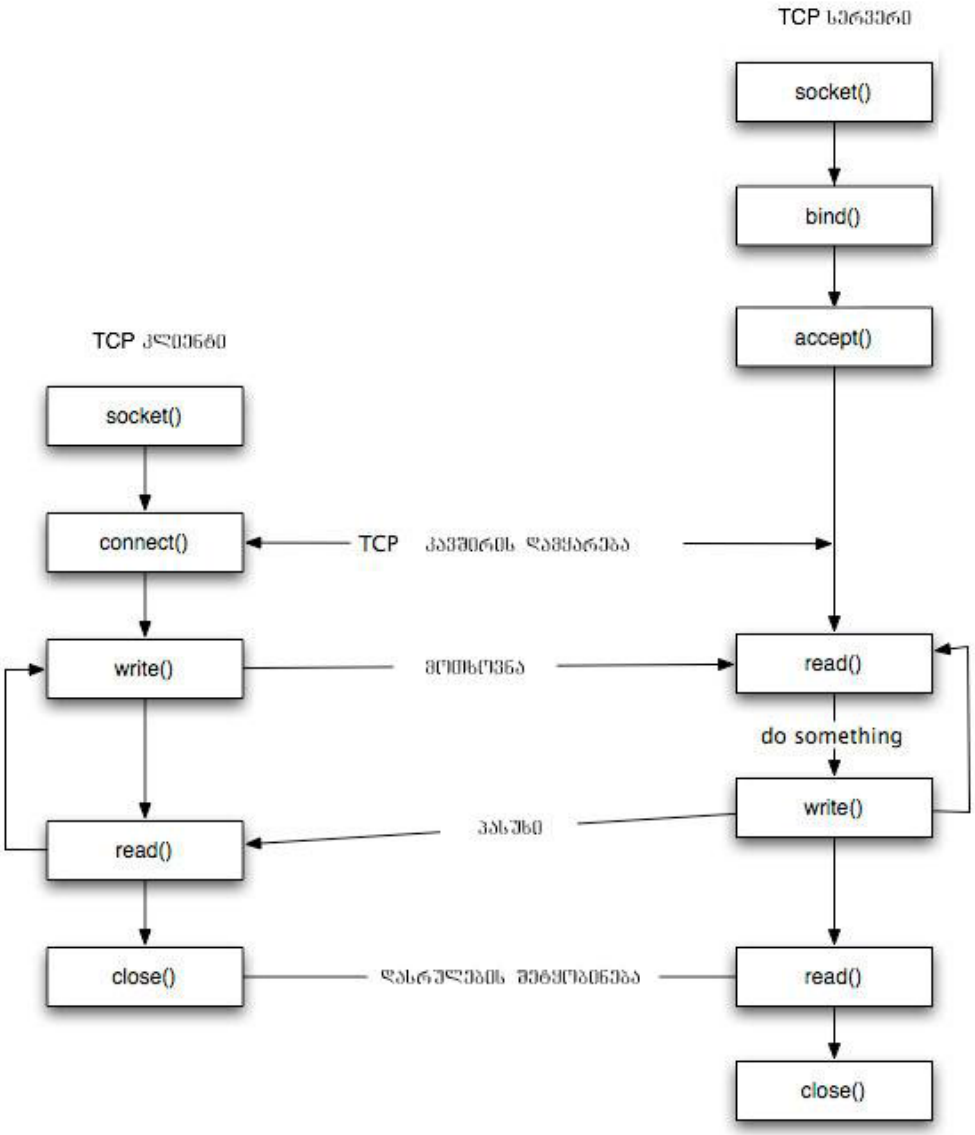
3. ინფორმაციის გაგზავნა და მიღება
   ეს ნაკადების ობიექტები შეიძლება გარდაიქმნას სხვა ობიექტებად, რომ უფრო მარტივი გახდეს ინფორმაციის მათგან წაკითხვა
   DataInputStream socketIn = new DataInputStream(link. getInputStream());

DataOutputStream socketOut = new DataOutputStream(link. getOutputStream());

4. კავშირის დახურვა

   კავშირის დახურვა ხდება Socket კლასის close() მეთოდით

   link.close();



2.4.5 UDP სოკეტები

ზემოთ უკვე განხილული იყო თუ როგორ ხდება ინფორმაციის UDP პროტოკოლის საშუალებით გაცვლა. იგივე ვრცელდებს ამ ტიპის სოკეტებზეც რადგან ისინი ამ პროტოკოლს იყენებენ. ისევე როგორც TCP სოკეტების შემთხვევაში, აქაც საჭიროა კლიენტისა და სერვერის პროგრამის შექმნა.

განვიხილოთ სერვერის პროგრამა:

1. DatagramSocket ობიექტის შექმნა

   DatagramSocket პარამეტრად მიეწოდება პორტის ნომერი და შესაბამისად სერვერი ელოდება კლიენტისგან დაკავშირებას ამ პორტის მიხედვით.

   ```
   DatagramSocket datagramSocket = new DatagramSocket(1234);
   ```

2. ბუფერის შექმნა შემომავალი დატაგრამებისთვის

   ეს ხდება ბაიტების მასივის შექმნით

   byte[] buffer = new byte[256];

3. DatagramPacket ობიექტის შექმნა შემომავალი დატაგრამებისგან

   მიეწოდება ორი პარამეტრი ეს არის წინასწარ შექმნის ბუფერი – ბაიტების მასივი და ბუფერის სიგრძე.

   DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);

4. შემომავალი დატაგრამების მიღება

   DatagramSocket კლასის receive() მეთოდით ხდება დატაგრამების მიღება

   datagramSocket.receive(inPacket);

5. პაკეტიდან გამომგზავნის მისამართისა და პორტის ამოღება

   DatagramPacket ობიექტის getAddress() და getPort() მეთოდებით

   InetAddress clientAddress = inPacket.getAddress();

   int clientPort = inPacket.getPort();

6. ინფორმაციის დაბრუნება ბუფერიდან

   პარამეტრებად მიეწოდება

   ბაიტების მასივი, საწყისი პოზიცია ამ მასივში, ბაიტების რაოდენობა

   String message = new String(inPacket.getData() 0, inPacket. getLength());

7. პასუხად დასაბრუნებელი დატაგრამის შექმნა

   იქმნება DatagramPacket ობიექტი, რომელსაც პარამეტრება ეწოდება

   ბაიტების მასივი, რომელიც უნდა დაბრუნდეს, ზომა პასუხის, კლიენტის მისამართი, კლიენტის პორტი

   DatagramPacket outPacket = new DatagramPacket(response. getBytes(), response.length(), clientAddress, clientPort);

8. პასუხის დატაგრამის გაგზავნა

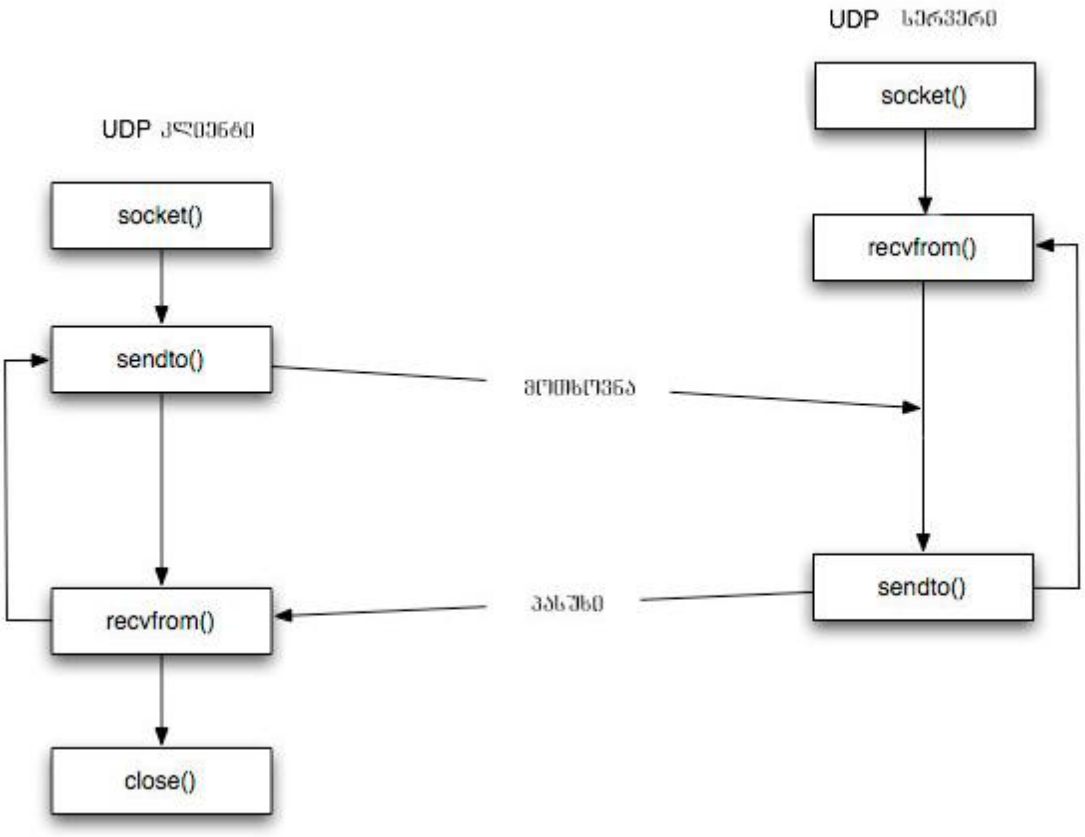
   DatagramSocket ობიექტის send მეთოდით ხდება გაგზავნა

   datagramSocket.send(outPacket);

9. DatagramSocket დახურვა

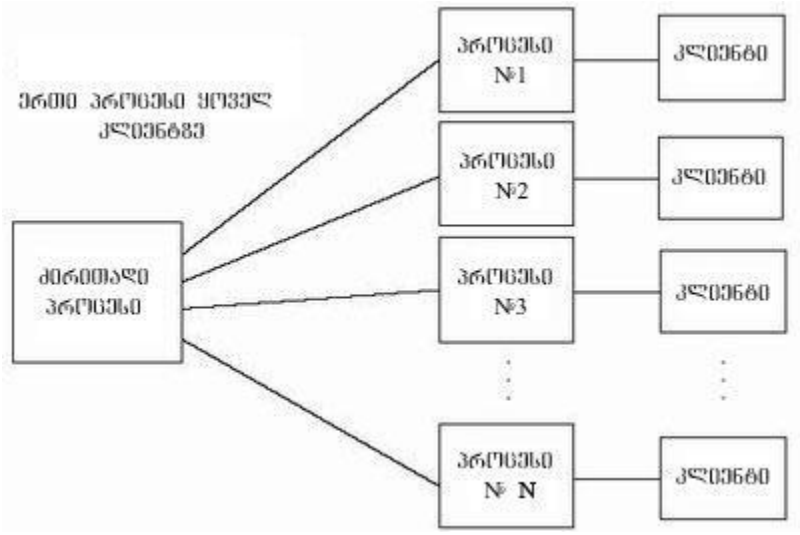
   DatagramSocket ობიექტის close მეთოდით

   datagramSocket.close();



2.6.5 მრავალმომხმარებლიან თამაშებში სოკეტების გამოყენება

ზემოთ განხილული მაგალითები მრავალი კლიენტის ერთდროულად მომსახურებას არ ითვალისწინებდა. იმისათვის რომ მოხდეს მრავალი კლიენტის მომსახურება სერვერიდან, ამისათვის სერვერზე არსებობს ერთი საბაზისო პროცესი, რომელიც მუდმივად პასიურად მომლოდინე რეჟიმშია. როდესაც კლიენტის მხრიდან კავშირი ხორციელდება ამ საბაზისო პროცესთან, ეს პროცესი უზრუნველყოფს ახალი პროცესის შექმნას და კლიენტის ახალ პროცესზე გადართვას, ხოლო თვითონ უბრუნდება ისევ საწყის მომლოდინე მდგომარეობას. ამრიგად თითოეული კლიენტისათვის სერვერზე იქმნება ცალკე პროცესი, რომელთა მუშაობაც მიმდინარეობს პარალელურ რეჟიმში. ეს ყველაფერი ნათლადაა ასახული დიაგრამაზე. ამ პროცესებს შორის ინფორმაციის გაცვლა სერვერზე ხორციელდება რეალურ დროში და რადგან ეს პროცესები ემსახურებიან კლიენტებს კლიენტებს შორის ინფორმაციის გაცვლაც ხორციელდება შესაბამისად. დიაგრამაზე ნათლად არის ნაჩვენები თუ როგორ ხდება მრავალმომხმარებლიანი სისტემის სოკეტებით ორგანიზება.



2.7 სოკეტ სერვერები

როგორც აღვნიშნეთ სოკეტ სერვერის გამოყენება აქტიურად ხდება თამაშებში, შესაბამისად საჭიროების გამო მსოფლიო ბაზარზე არსებობს რამდენიმე სოკეტ სერვერი, მათ შორის ყველაზე მოთხოვნადი სერვერებია თამაშების მიმართულებით: SmartFoxServer, ElectroServer, Flash Media Interactive Server, Red5

SmartFoxServer

SmartFoxServer (http://smartfoxserver.com) წარმოადგენს კომპანია gotoAndPlay()-ს პროდუქტს. ის ძალიან მოქნილი და ძლიერი ინსტურმენტია რესურსების ოპტიმალური მართვისთვის, რაც საშუალებას იძლევა მაღალი პროდუქტიულობის მქონე ვირტუალური სოციალური თამაშების შესაქმნელად.

მას ძალიან მრავალფეროვანი მხარდაჭერა აქვს კლიენტის ტექნოლოგიების: Flash, Unity, iOS, Android, HTML5, .Net, Java და ა.შ. აქვს ძალიან მოქნილი API რომელიც უშუალოდ თამაშების შექმნისკენაა ორიენტირებული და რაც შეიძლება მოკლე დროში ხდის შესაძლებელს სხვადასხვა თამაშების შექმნას. ასევე ჩაშენებული აქვს სტრიმი გადასაცემი სერვერი, რომელიც ღია კოდის მქონე Red5-ზეა დამყარებული.

თუ სხვადასხვა ტიპის მომხმარებლებისაგან შეუძლებელია სოკეტ კავშირის გახსნა, სხვადასხვა მიზეზების გამო, SmartFoxServer საშუალებას იძლევა კლიენტმა კავშირი დაამყაროს სერვერთან HTTP-ს საშუალებით, რომელიც რეალიზებლია BlueBox მოდულის სახით.

SmartFoxServer არის დახურული კოდის მქონე, ფასიანი პროდუქტი. მისი ლიცენზია მომხმარებლების რაოდენობის მიხედვით იცვლება, რამდენიმე ეტაპად.

ElectroServer

ElectroServer წარმოადგენს ერთ-ერთ ყველაზე გავრცელებულ პლატფორმას მრავალმომხმარებლიან სოციალური თამაშების შესაქმენლად. იგი შექმნილია კომპანიის Electrotank-ის მიერ 2001 წელს. SmartFoxServer-ის მსგავსად მასაც ბევრი ინსტრუმენტი და

წინასწარ დამუშავებული საჭირო საშუალებები აქვს თამაშების სწრაფად და ხარისხიანად თანამედროვე სტანდარტების შესაბამისად დასამუშავებლად.

ElectroServer-ს აქვს ასევე მასშტაბირების მხარდაჭერა. ლოგიკა ინახება ერთ სერვერს, ხოლო დანარჩენი გამოიყენება კლიენტებისგან ინფორმაციის მიღება გაცემისთვის.

ElectroServer-იც წარმოადგენს დახურული კოდის მქონე, ფასიან პროდუქტს და მისი ლიცენზიაც მომხმარებელეთა რაოდენობის მიხედვით იცვლება.

Flash Media Interactive Server

Flash Media Interactive Server - შექმნილია კომპანია Adobe-ს მიერ და ძირითადად წარმოადგენს ვიდეო სტრიმინგისა და რეალურ რეჟიმში კლიენტების კომუნიკაციის საშუალებას. იგი საშუალებას იძლევა ვიდეო ნაკადების ძალიან პოპლარული კოდეკების სასუალებით გადაცემას და ასევე პირდაპირ რეჟიმში ვიდეო ნაკადის გადაცემისა. სერვერული ნაწილის მხარეს პროგრამული რეალიზაცია ხდება ActionScript-ის საშუალებით.

მას ასევე აქვს მასშტაბირების მხარდაჭერაც და მსგავსად ElectroServer-ისა ისიც ძირითად ლოგიკას მთავარი სერვერიდან იღებს.

Flash Media Interactive Server წარმოადგენს დახურული კოდის მქონე ფასიან პროდუქტს.

Red5

Red5 წარმოადგენს ღია კოდის მქონე სოკეტ სერვერს. მას აქვს real-time protocol (rtm) მხარდაჭერა. იგი წარმოადგენს Flash Media Interactive Server-ის უფასო ალტერნატივას ვიდეო ნაკადების გადასაცემად.

საკუთარი სოკეტ სერვერის შექმნა

თუ ზემოთ აღწერილი პროდუქტები ვერ აკმაყოფილებს კლიენტის მოთხოვნებს, ტექნიკური რეალიზაციის ან თუნდაც მისი ღირებულის გამო, შესაძლებელია შეიქმნას სოკეტ სერვერი საკუთარი მოთხოვნების შესაბამისი. ძირითადად სოკეტ სერვერების შექმნა ხდება C++ ან Java პროგრამირების ენებზე. საკუთარი სოკეტ სერვერის შექმნას თავისი უპირატესობების მხრივა ასევე შეიძლება მოყვეს გარკვეული რისკები. მისი შექმნა მოითხოვს

საკმაოდ დიდ რესურსს, რადგან უნდა დამუშავდეს რაც შეიძლება სრულყოფილი მოდელი, რომელიც მორგებული იქნება ამოცანაზე, ასევე მისი ტესტირებაც საკმაოდ რთულია. ზემოთ ჩამოთვლილი მზა პროდუქტები უკვე ტესტირებულია რამდენიმე ათეულ ათას მომხმარებელზე და მათი არჩევის შემთხვევაში, ათეულ-ათასობით მომხმარებელზე სერვერის სტაბილურობა გარანტირებულია. მათ ასევე ახლავს საკმაოდ მრავალფეროვანი API როგორც სერვერის მხარეს თამაშების ლოგიკის შესაქმნელად, ასევე კლიენტის მხარესაც. ამ საკითხების სიღრმისეული ანალიზის შემდეგ, უკვე თვითონ მომხმარებელი წყვეტს რომელი მიმართულება აირჩიოს, ამოცანიდან გამომდინარე, უკვე მზა პროდუქტი გამოიყენოს, თუ დაიწყოს მუშაობა საკუთარი სოკეტ სერვერის შექმნაზე.

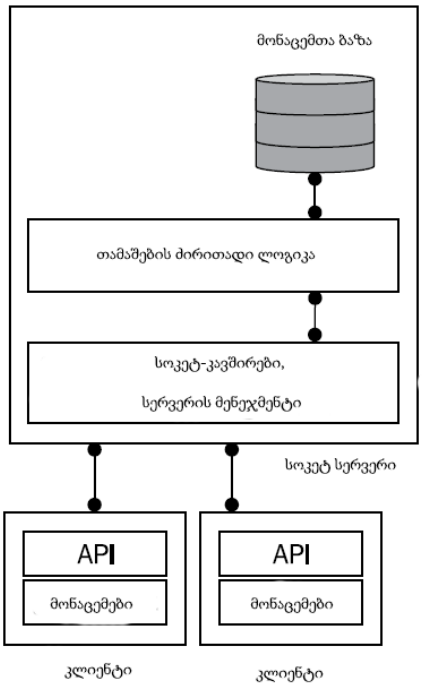
მრავალმომხმარებლიანი სოციალური თამაშებისთვის ახალი პლატფორმის პროგრამული რეალიზაცია

როგორც ზემოთ აღვნიშნეთ, მსოფლიო ბაზარზე არსებობს საკმაოდ პოპულარული რამდენიმე პროდუქტი, რომელიც გამოიყენება მრავალმომხარებლიანი თამაშების შესაქმნელად. თითოეულს თავისი უპირატესობები გააჩნია, მაგრამ ყველა არის დახურული კოდის მქონე და ფასიანი ლიცენზიის პირობებით. იმისთვის, რომ თამაშების რეალიზაცია ყველასთვის ხელმისაწვდომი ყოფილიყო და ბევრად მარტივი, მიზნად დავისახე შემექმნა ახალი პლატფორმა, რომელიც იქნებოდა ღია კოდის პრინციპით და უფასო ლიცენზიის მქონე.

ახალი პლატფორმის შესაქმნელად, გამოყენებულია სოკეტზე დამყარებული კავშირი კლიენტსა და სერვერს შორის.

პროგრამული კოდი დაწერილია ობიექტზე ორიენტირებული პროგრამირების ენაზე Java. კლიენტისთვის კი გამოყენებულია ActionScript 3.0.

კლიენტისა და სერვერის კავშირი უზრუნველყოფილია ძირითად შემთხვევაში TCP პროტოკოლის საშუალებით, ასევე აქვს UDP პროტოკოლის მხარდაჭერაც.



პროგრამის ზოგადი სქემა ასეთია. რამდენიმე კლიენტი უკავშირდება სოკეტ სერვერს. სერვერი ჯერ ახდენს ამ კავშირების მენეჯმენტს და შემდეგ უკვე შესაბამისი ლოგიკის საშუალებით ახდენს მონაცემების ბაზისთვის მიწოდებას და ასევე კლიენტთან დაბრუნებას.

საწყის ეტაპზე სერვერი იწყებს მუშაობას განსაზღვრულ პორტზე. ეს პორტი არის ძირითად პროცესისთვის განკუთვნილი და არის მომლოდინე რეჟიმში, ელოდება ახალ კლიენტს.

პირველ ეტაპზე სერვერზე ხდება აღნიშნული პლატფორმის გაშვება. გაშვებისას საწყის პარამეტრებში ეთითება ძირითადი პორტი, რომელზეც ხდება თითოეული კლიენტის დაკავშირება.

მოკლედ აღვწეროთ ის ძირითადი კლასები და მეთოდები, რომელზეც დამყარებულია პლატფორმა:

პლატფორმაზე შეგვიძლია გავუშვათ სხვადასხვა რამდენიმე აპლიკაცია (თამაში), თითოეულისთვის აუცილებელია საკუთარი ლოგიკის არსებობა. თითოეულისთვის ხდება

სხვადასხვა გაფართოებების შექმნა, ჩვენს შემთხვევაში TrisExtension, რომელიც წარმოადგენს საბაზისო TCPExtension კლასის გაფართოებას.

TCPExtension ძირითადი აღწერა

1. საწყის ეტაპზე ხდება კლასის ინიციალიზაცია, სადაც განისაზღვრება რომელ პორტზე მუშაობს იგი, მისი სახელი და სხვა პარამეტრები

```
this.port = port;
this.extensionInstance = this;
this.extension = this.getClass();
this.extensionName= this.extensionInstance.getClass().getSimpleName();
```

2. იმ პროცესის გაშვებისას, რომელიც კონკრეტულად ამ გაფართოებას ემსახურება, ხდება საბაზისო სოკეტის ინიციალიზაცია და ამ სოკეტის მიხედვით დაკავშირებული მომხმარებლისთვის ცალკე პროცესის შექმნა, რომელიც მოემსახურება უშუალოდ ამ კლიენტს.

```
setSocket( new ServerSocket( getPort(), 500 ) );
while ( isRunning() ) {
    try {
          client = getSocket().accept();
    } catch (Exception e) {}

    new TCPClient( client, this ).start();
}
```

3. TCPExtension კლასს შეუძლია გაუგზავნოს მონაცემები ყველა იმ კლიენტს რომელიც ჩვენს პლატფორმასთანაა დაკავშირებული, ან კონკრეტულად მხოლოდ იმ კლინეტებს, რომლებიც მოცემულ გაფართოებას უკავშირდებიან

```
          synchronized (TCPClient.getClients()) {
                for ( Map.Entry<Object, TCPClient> item :
TCPClient.getClients().entrySet() ) {
                      item.getValue().send( call );
                }
          }

synchronized (clients) {
                for    (    Map.Entry<Object,    TCPClient>    item    :
clients.entrySet() ) {
```

```
                        item.getValue().send( call );
                    }
                }
```

4. ასევე ხდება იმ ლოგიკის აღწერა, რომელიც უნდა შესრულდეს კლინეტის დაკავშირებისას, კავშირის გაწყვეტისას, მონაცემების გაგზავნისას ან მიღებისას

```
onExtensionReady()
onConnect( TCPClient client );
onDisconnect( TCPClient client );
onDataSpecial( TCPClient client, String methodName, JsonObject params );
```

მეორე საბაზისო კლასი ჩვენს პლატფორმაზე არის TCPClient, რომელიც აღწერს კლიენტის ძირითად ფუნქციებს.

1. ინიციალიზაციისას განისაზღვრება მისი ძირითადი პარამეტრები

```
public TCPClient( Socket client, TCPExtension extension ) {
        this._client = client;
        this._extension = extension;

        String                      cleanIp                      =
client.getRemoteSocketAddress().toString().split(  ":",  2)[0].replace(
"/", "");

        this._ipAddress = cleanIp;
        Logger.log( "Client connected from: " + _ipAddress );
}
```

2. კლიენტის დაკავშირებისას კლიენტს ენიჭება უნიკალური იდენტიფიკატორი this.setUniqueId(this.toString());
3. იმისთვის რომ სანქცირებული იყოს კლიენტისგან სერვერზე წვდომა, დაცვის კუთხით უგზავნის შესაბამის PolicyFile-ს
4. read() მეთოდით ხდება კლიენტისგან გამოგზავნილი მონაცემების წაკითხვა

```
while ( !(input = _in.readLine()).equals( null ) ) {
        process( input );
}
```

5. `process` მეთოდი ახორციელებს კლიენტისგან მოსული მონაცემების დამუშავებას, და მის მიხედვით სხვადასხვა შესაბამისი ლოგიკის შესრულებას.
   კლიენტისგან იგზავნება JSON ობიექტი ამიტომ ხდება მისი დაშლა
   params = (JsonObject) new JsonParser().parse(line);
   methodName = params.get("method").getAsString();
   methodName არის იმ მეთოდის სახელი, რომლის გამოძახებასაც ცდილობს კლიენტი და ხდება შესაბამისი პარამეტრებით ამ მეთოდის გამოძახება

```
if ( _extension.onDataSpecial( this, methodName, params ) == false ) {
    Object[] o = { this, params };
    m = _extension.getExtension().getMethod( methodName, classes );
    m.invoke( _extension.getExtensionInstance(), o );
}
```

6. ერთ კლიენტს შეუძლია მონაცემები გაუგზავნოს მეორე კლიენტს ან კლიენტების ჯგუფს, რასაც შემდეგი მეთოდებით ახორიცელებს

```
public void send( RemoteCall message ) {
    setOutboundBytes( getOutboundBytes() +
message.properties.toString().getBytes().length );
    _out.print(            message.properties.toString()            +
_extension.getNewlineCharacter() );
    _out.flush();
}
public  static  void  send(  Map<String,  User>  userList,  RemoteCall
message ) {
          for  ( Map.Entry<String, User> user : userList.entrySet() )
{
                user.getValue().getTcpClient().send( message );
          }
    }
```

მესამე ძირითად კომპონენტს პლატფორმის წარმოადგენს მონაცემების გაცვლის წესი - სქემა. რა თქმა უნდა მონაცემები გადაიცემა TCP/UDP პროტოკოლებით, მაგრამ უნდა იყოს ამ გადაცემის ერთი სტანდარტი. ჩვენს შემთხვევაში მონაცემების გაცვლა ხდება JSON სტრიქონის საშუალებით. რომელიც კლიენტიდან გადაეცემა სერვერს, ან სერვერიდან კლიენტს და მისი მიხდევით ხდება შესაბამისი ოპერაციების შესრულებას.

მონაცემების დამუშავებას და მომზადებას გასაგზავნად ახორციელებს RemoteCall კლასი.

1. მისი ინიაცილიზაციის აუცილებლად ეთითება ის მეთოდი, რომლის გამოძახებაც უნდა მოხდეს ან კლიენტის ან სერვერი მხარეს

```
public RemoteCall( String method ) {
          properties.addProperty( "method", method );
    }
```

2. გაგზავნისას ასევე აუცილებელია სხვადასხვა პარამეტრების გადაცემა, რომელსაც ახორციელებს put მეთოდი

```
put( String key, String value )
```

ზემოთ აღწერილი ძირითადი კლასების გარდა პლატფორმისთვის უკვე დამუშავებულია დამატებითი ფუნქციონალი, რომელიც თამაშების სწრაფი რეალიზაციის საშუალებას იძლევა. თამაშების ძირითად ელემენტს წარმოადგენს Lobby, რომელიც არის თამაშის წინა ინტერფეისი. ლობიში ხდება სხვა მოთამაშეების ნახვა, არსებული ოთახების (Room) ჩვენება. ჩვენთვის სასურველი ახალი Room-ის შექმნა და ა.შ.

Lobby კლასის მოკლე აღწერა:

1. getRoomList - ამჟამად არსებული ოთახების სიის ამოღება და კლიენტისთვის მიწოდება
2. getFullUserList - მომხმარებელთა სრულად ჩვენება
3. createRoom - ახალი ოთახის შექმნა, რომელსაც გამოვიყენებთ უკვე უშუალოდ ჩვენი თამაშისთვის.
4. joinRoom - კონკრეტულ ოთახში შესვლა.
5. sendRoomMessage - ოთახში მყოფ მომხმარებლებთან მონაცემების გაგზავნა
6. sendPrivateMessage - კონკრეტულად ერთ კლიენტთან მონაცემების გაგზავნა
7. leaveRoom - ოთახის დატოვება
8. getRoomByName - კონკრეტულად რომელიმე ოთახის ნახვა
9. sendToList - მომხმარებელთა გარკვეული ჯგუფისთვის მონაცემების გაგზავნა.

Room - კლასი. ეს არის კონკრეტული ოთახი, რომელშიც არსებული მოთამაშეები ეთამაშებიან ერთმანეთს. ოთახები იზოლირებული ერთმანეთისგან.

1. Room-ის ინიციალიზაციის ხდება ოთახის პარამეტრების მითითება: სახელის, პაროლის, მაქსიმალური მომხმარებლების რაოდენობა.

```
name = room.get( "name" ).getAsString();
password = room.get( "password" ).getAsString();
maxUsers = room.get( "maxUsers" ).getAsInt();
customData = room.get( "customData" ).getAsJsonObject();
if ( (password != null) && !password.equals( "" ) ) {
      this.isPrivate = true;
      this.password = room.get( "password" ).getAsString();
}
setOwner( user );
setRoomId();
```

2. კლიენტის შესვლისას Room-ში ხდება სხვადასხვა ტიპის მონაცემების გაგზავნა, რომელსაც onUserJoin(User user) ახორიცელებს.
3. კლიენტის დატოვებისას შესაბამისი ინფორმაციის გაცვლას ახორციელებს onUserLeave(User user)
4. თუ რაიმე მიზეზის გამო, კლიენტს უნდა დავატოვებინოთ ოთახი, ეს შესრულდება მეთოდით kickUser(User user, JsonObject json)

User კლასში აღწერილია ის ძირითადი ლოგიკა, რომელიც ყველა კლიენტს სჭირდება

1. ინიციალიზაციის ხდება საბაზისო პარამეტრების მითითება

```
tcpClient = client;
username = client.getUniqueId().toString();
setUserID();
```

2. მაგიდაზე შესვლისას შესაბამისი ოთახის მინიჭება

   setRoom(Room room)

3. უნიკალური იდენტიფიკატორის მინიჭება

   void setUserID()

უფრო დეტალურად შეგიძლიათ იხილოთ თანდართული პლატფორმის პროგრამული კოდი.

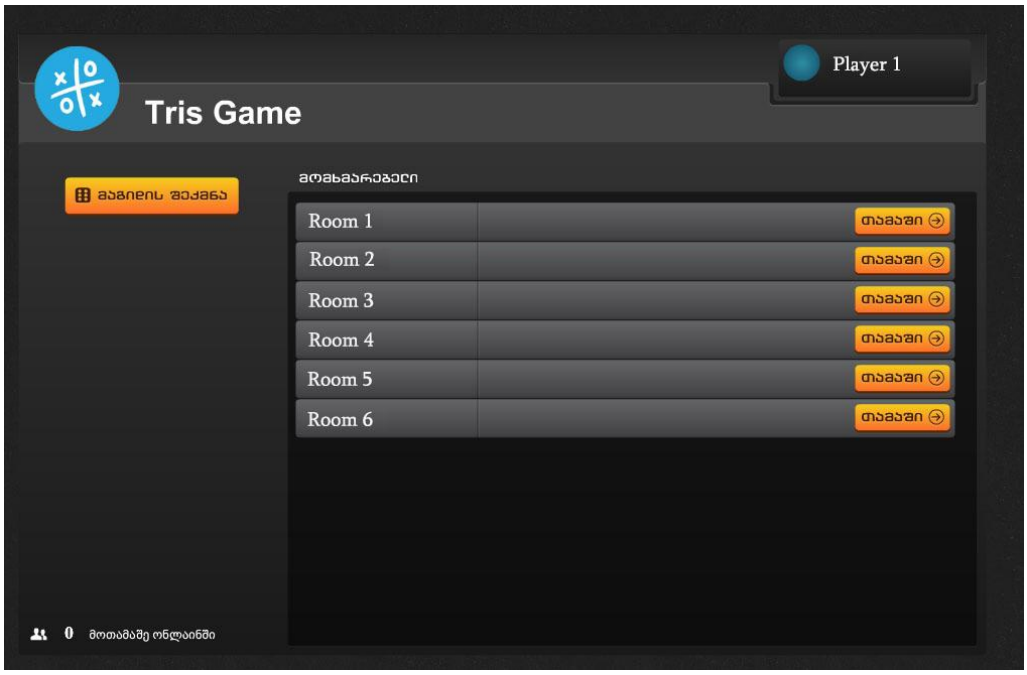
ახალი პლატფორმის გამოყენების სადემონსტრაციო მაგალითი

ამოცანის აღწერა

კლიენტი უნდა დაუკავშირდეს სერვერს და ავტორიზაცია მოახდინოს სასურველი მომხარებლის სახელის მიხედვით. ავტორიზაციის წარმატებით გავლისას, გადამისამართდეს ლობიში, სადაც ნაჩვენები იქნება ამჟამად მიმდინარე თამაშები. კლიენტს უნდა შეეძლოს ან უკვე არსებულ თავისუფალ თამაშში შესვლა, ან ახალი თამაშის შექმნა სურვილისამებრ. თამაშში ორი მოთამაშის შესვლისას იწყება თამაში. თამაშის ლოგიკა შემდეგია. ორივე მომხმარებელი მონაცვლეობით ასრულებს სვლებს. 3x3 უჯრებიან დაფაზე, ხდება მონაცვლეობით კლიენტის სურვილისამებრ X ან O სიმბოლოების განთავსება. ის კლიენტი რომელმაც შექმნა მაგიდას სვამს დაფაზე X სიმბოლოს, მეორე კი - O სიმბოლოს. გაიმარჯვებს ის კლიენტი, რომელიც პირველი შეავსებს ვერტიკალს, ჰორიზონტალს ან დიაგონალს. თუ ვერცერთმა ვერ შეავსო, თამაში სრულდება ფრედ.

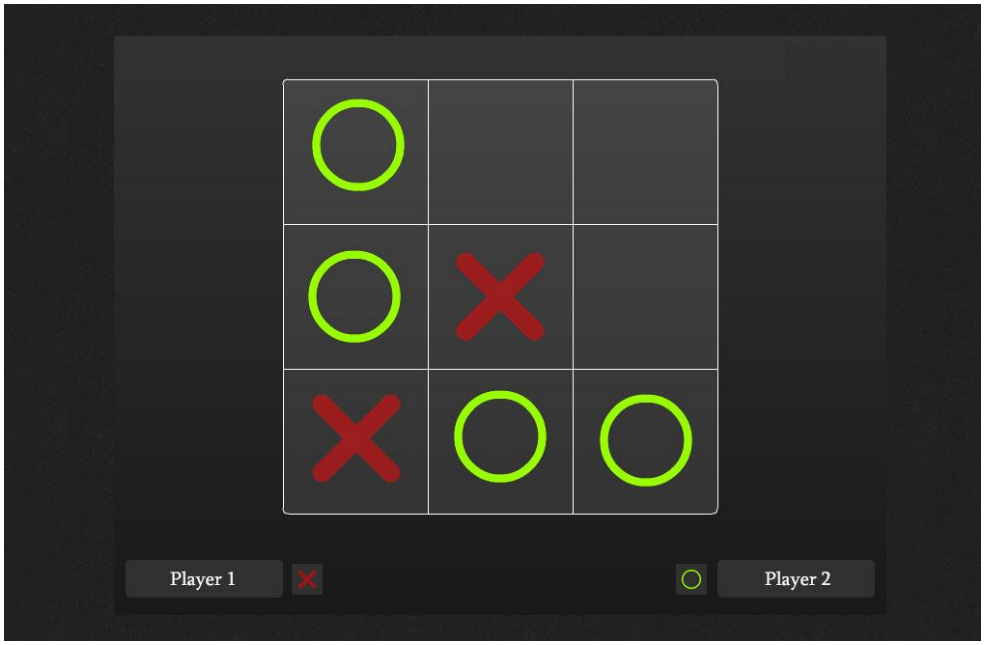
ამოცანის პრაქტიკული რეალიზაცია

საწყის ეტაპზე ხდება თამაშის ინიციალიზაცია. კლიენტი უკავშირდება სერვერ 8889 პორტზე და ხსნის სოკეტზე დამყარებულ კავშირს. გამოდის ავტორიზაციის ფორმა, სადაც ეთითება მომხმარებლის სახელი და პაროლი. მომხმარებლის სახელი არის ის, რომელიც შემდეგ თამაშში გამოჩნდება სხვა მოთამაშეებთან. შესვლის ღილაკზე დაჭერით, მონაცემები იგზავნება სერვერზე, მოწმდება შესაბამისად და ვალიდაციის წარმატებით გავლის შემთხვევაში იტვირთება თამაშის ლობი.

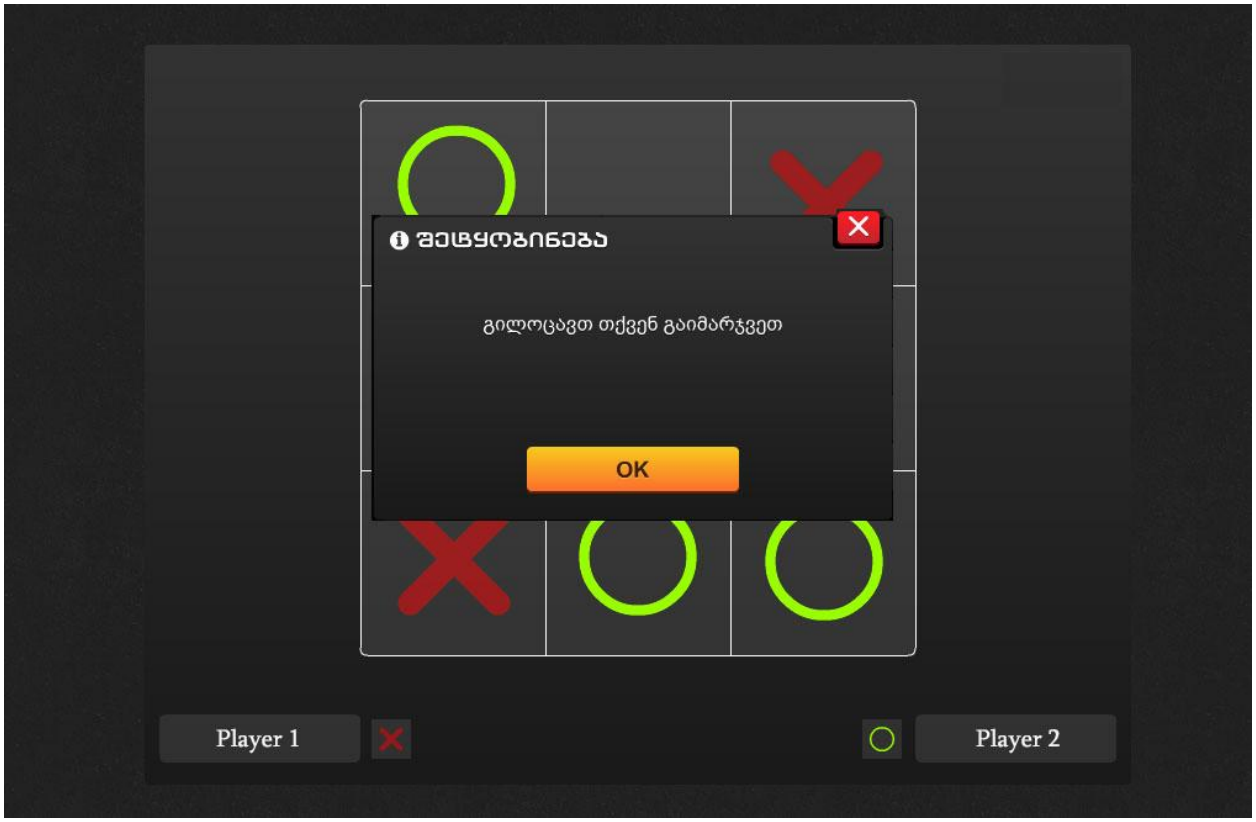


ლობიში ნაჩვენებია არსებული Room-ების ჩამონათვალი, რომელსაც მისი სახელი აქვს მითითებული. თამაშის ღილაკზე დაჭერით ხდება ოთახში შესვლა და შემდეგ თამაშის დაწყება. თუ არ გვსურს აქ ჩამოთვლილ Room-ებში თამაში, მაშინ შეგვიძლაი მაგიდის შექმნის ღილაკით შევქმნათ ჩვენთვის სასურველი Room. შექმნილი Room-ის მონაცემები ეგზავნებ სხვა კლიენტებს და მათ ჩამონათვალში ჩვენი შექმნილი ოთახიც ემატება.

მაგიდის შექმნისას ან თამაშში შესვლისას ხდება თამაშის ინტერფეისის ჩატვირთვა.



თამაშის მიმდინარეობს, მოთამაშეები მონაცვლეობით აკეთებენ სვლებს. სურათზე ნაჩვენებია თამაშის კონკრეტული ეტაპი. სერვერზე ყოველი სვლის გაკეთებისას მოწმდება თამასის ლოგიკა, ხომ არ გაიმარჯვა რომელიმე კლიენტმა და ოპონენთან ხდება განხორციელებული სვლის გაგზავნა. ტუ რომელიმე მოთამაშემ გაიმარჯვა თამაში სრულდება.



ეს არის მარტივი დემონსტრაცია თამაშების ახალი პლატფორმისთვის. თამაშის სიმარტივე, პლატფორმის სიმარტივეს არ ნიშნავს. პლატფორმაზე შესაძლებელია მარტივი და ამავე დროს ძალიან რთული ლოგიკის მქონე თამაშების აწყობა. პლატფორმა წარმოადგენს იმ ბაზის რომელიც აუცილებელია სხვადასხვა ტიპის თამაშების შესაქმნელად. ძირითადი სამუშაო სწორედ პლატფორმის დახვეწა-განვითარება და თამაშებისთვის მაქსიმალურად მოქნილ მექანიზმის შექმნაა. არსებული პლატფორმა ორიენტირებულია თამაშებზე, თუმცა ეს არ გამორიცხავს მისი საშუალებით სხვა ტიპი აპლიკაციების შექმნას. მაგალითად ონლაინ

ჩატი, ან ონლაინ კონსულტაციის სისტემა და ა.შ. თუ გავითვალისწინებთ თანამედროვე ტენტენციებს, ამ ტიპის პლატფორმებს საკმაოდ დიდი მომავალი აქვს.

დასკვნა

ჩემი შექმნილი მრავალმომხმარებლიანი სოციალური თამაშების პლატფორმა წარმოადგენს ღია კოდის და უფასო ლიცენზიის მქონე პროგრამულ უზრუნველყოფას. თუ გავითვალისწინებთ იმას, რომ ამჟამად მსოფლიო ბაზარზე არსებული ძირითადი პროდუქტებიდან ყველა არის დახურული კოდის და საკმაოდ ძვირი ლიცენზიის მქონე, ამიტომ დადგა საჭიროება უფასო ვერსიის შექმნისა. მსოფლიო მასშტაბით სოციალური თამაშები აღმავლობის ეტაპზე, დღითი-დღე უფრო პოპულარული ხდება და შესაბამისად არსებული პლატფორმის საჭიროებაც დადგა. რადგან მცირებიუჯეტიანი პროექტის დასაწყებად, ძალიან რთული იქნება იმ ტიპის ლიცენზიის ყიდვა, რასაც სხვა ზემოთჩამოთვლილი პროდუქტები სთავაზობენ, ამიტომ ვფიქრობ საკმაოდ პოპულაული გახდება ახალი პლატფორმა.

მრავალმომხმარებლიანი სოციალური თამაშების ახალ პლატფორმაში დამუშავებულია ის ელემენტები, რომლებიც თამაშების უმეტესობაში აქტიურად გამოიყენება, ამიტომ ეს პროდუქტი დიდ დაინტერესებას გამოიწვევს. იგი ისევე მარტივად გასაგებია, როგორც გამოცდილი პროგრამისტებისთვის, ასევე დამწყებთათვისაც. მოითხოვს მხოლოდ მრავალმომხმარებლიანი სისტემების საბაზისო ელემენტების ცოდნას.

აღნიშნული პლატფორმის სერვერული ნაწილი თავსებადია ბევრ ოპერაციულ სისტემასთან, რადგან იყენებს ობიექტზე ორიენტირებულ პროგრამირების ენას Java-ს. რადგან ახალი პლატფორმის კოდი თავისუფლად ხელმისაწვდომია, ეს უფრო მოქნილს ხდის მას და მისი გამოყენების შემთხვევაში, სურვილსამებრ სხვადასხვა გაფართოების საშუალებას იძლევა.

გამოყენებული ლიტერატურა

1. Flash Multiplayer Virtual World – Makzan (Author), Packt Publishing 2010
2. http://docs.oracle.com/javase/tutorial/networking/sockets/
3. http://smartfoxserver.com/
4. http://www.electrotank.com/
5. http://www.adobe.com/products/adobe-media-server-professional.html
6. http://www.red5.org/
7. http://gotoandplay.it/_articles/2003/12/xmlSocket.php