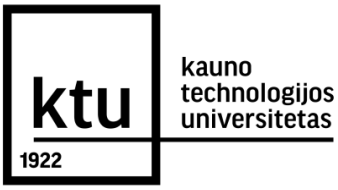


KAUNO TECHNOLOGIJŲ UNIVERSITETAS

INFORMATIKOS FAKULTETAS

PROGRAMŲ INŽINERIJOS KATEDRA

WEBDROID

Technologinis projektas

Atliko:

IF-4/14 grupės studentai

Gailius Kazlauskas

Eimantas Morkūnas

Gintaras Volkvičius

Priėmė:

dėst. Tomas Blažauskas

KAUNAS 2016

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

1. ĮVADAS

1.1. Projekto idėja

Projekto idėja kilo išbandžius *Google* sukurtą naršyklinį žaidimą *Lightsaber Escape*. Šiame statinio veiksmo žaidime su priešais yra kovojama valdant vadinamąjį „šviesos kardą“ savo mobiliojo telefono judesiais. Telefonas yra prijungiamas naudojant telefone įdiegtą naršyklę ir suvedant kodą, su kuriuo telefonas yra prijungiamas prie konkrečios naršyklės sesijos ir taip susiejamas su žaidimu.

Taigi, susivilioję telefono teikiamomis galimybės, sumanėme sukurti žaidimą, kuris būtų paprastas ir būtų žaidžiamas valdant telefoną, t. y. naudojantis išmaniojo telefono sensoriais, o visas veiksmas būtų matomas naršyklėje.

Projekto pradžioje galvojome sukurti Olimpinių žaidynių simuliacinį žaidimą su keletu rungčių, tačiau mūsų galimybės laiko atžvilgiu neatitiko lūkesčių, tad nusprendėme apsistoti ties spalvų žaidimu, kurį viduryje projekto sugalvojome sukurti kaip gerą testavimo priemonę.

Beat That Color – žaidimo pavadinimas, kuris niekur nekonfigūruoja, kadangi tiek aplikacija, tiek svetainė yra pavadinta projekto technologiniu pavadinimu *WebDroid*. Atskiru žaidimo pavadinimu norime parodyti, jog *WebDroid* labiau laikome pačią žaidimo platformą, kuri paremta telefono ir svetainės ryšiu ir jog sukurtas žaidimas yra tik vienas iš daugelio galimų žaidimų, naudojant šią platformą. Tad pristatydami savo technologinį projektą pasilikome prie technologinio pavadinimo.

1.2. Trumpas aprašymas

Žaidime žaidėjas savo mobiliojo telefono pagalba valdo rutuliuką, kurį mato naršyklės sesijoje. Judėdamas ekrane rutuliukas piešia linijas, kurios yra tos pačios spalvos, kaip ir pats rutuliukas. Žaidimo tikslas – per tam tikrą fiksuotą laiką nuspalvinti didžiausią ekrano plotą savo spalva.

Žaidimo taisyklės:

- pradinė ekrano spalva yra balta;
- rutuliukai juda pastoviu greičiu, su telefonu galima reguliuoti tik jų judėjimo kryptį;
- žaidėjas, piešdamas ant balto fono, pasisavina užpieštą plotą;
- žaidėjas, piešdamas ant kito žaidėjo užpiešto ploto, pasisavina šio žaidėjo plotą;
- laimėtoju skelbiamas žaidėjas/žaidėjai, kurių spalvos dengia daugiausiai ekrano ploto pasibaigus laikui.

Žaidime galima pasirinkti savo slapyvardį, simbolius (kurie bus rodomi ant rutuliuko; iki 3 simbolių) ir rutuliuko spalvą. Du žaidėjai negali turėti tos pačios spalvos rutuliuko, nebent jie priklauso vienai komandai.

1.3. Panašios sistemos

Kaip jau minėjome, projekto idėja kilo pamačius *Google* žaidimą *Lightsaber Escape*. Jis naudoja telefono sensorius, kurių duomenis jis gauna iš naršyklės telefone ir siunčia juos apdoroti prieš atvaizduodamas rezultatą į kitą naršyklę kompiuteryje. Telefono kalibravimas su ekranu yra atliekamas telefoną sulyginus su ekrano viduriu.

Mūsų platforma sensorius gauna iš bazinės *Android* aplikacijos, kuri pati juos ir apdoroja ir apdoroti duomenys išsiunčiami į serverį, kuris juos perduoda naršyklės sesijai kompiuteryje. Telefono kalibravimas taip pat atliekamas sulyginus telefono plokštumą su ekrano plokštuma, tačiau yra ne viena, o 6 kalibravimo padėtis, kuriomis žaidėjai gali susikalibruoti telefoną.

Dar viena panaši sistema yra *AirConsole*. Ši sistema jau yra platforma, kadangi joje yra daugelis žaidimų, kurie yra žaidžiami pasitelkus telefoną kaip valdymo pultą, o kompiuterio ekraną – žaidimo atvaizdavimui. Telefonas valdymo pultu gali būti paverstas naudojant jų sukurtą specialią programėlę, arba tiesiog tiesiai iš mobiliojo telefono naršyklės. Šia platforma gali pasinaudoti visi norintys ne tik žaisti, bet ir kurti žaidimus. Deja, daugelis kūrėjų šioje sistemoje žaidimus kūrė panaudodami tik telefono liečiamąsias galimybes (kaip valdymo pultą), ne sensorines.

2. TECHNINIS APRAŠAS IR REIKALAVIMAI

2.1. Funkciniai reikalavimai

Funkcinių reikalavimų sąrašas:

- sukurti aplikaciją skirtą *Android OS* turintiems išmaniesiems įrenginiams;
- sukurtai aplikacijai sukurti patogų ir paprastą GUI, kuri vartotojui padėtų orientuotis žaidimą žaidžiant pirmą kartą;
- sukurti aplikacijoje naršyklės sesijos ir išmaniojo įrenginio sinchronizavimo su QR kodu galimybę;
- sukurti aplikacijoje naršyklės sesijos ir išmaniojo įrenginio sinchronizavimo rankiniu būdu galimybę;
- sukurti aplikacijoje galimybę žaidėjui nurodyti savo duomenis (slapyvardį, inicialus);
- sukurti aplikacijoje galimybę žaidėjui pasirinkti norimą spalva iš laisvų spalvų sąrašo;
- sukurti aplikacijoje kalibravimo galimybę keliomis pozicijomis;
- aplikacijoje integruoti valdymą judesiu ir/ar ekranu naudojant išmanųjį įrenginį;
- sukurti serverį, kuris turėtų daugelio sesijų palaikymą;
- sukurti paprastą GUI svetainei, kuris rodytų žaidėjų judėjimą bei žymėtų jų judesius;
- internetinėje svetainėje rodyti žaidėjo būseną, jo žaidimo rezultatą;

2.2. Nefunkciniai reikalavimai

Nefunkcinių reikalavimų sąrašas:

- aplikacija su serveriu turi bendrauti žinutėmis, kurios turi būti užkoduotos JSON formatu;
- aplikacija negali leisti žaidėjams manipuliuoti į serverį siunčiamomis žinutėmis kitaip, nei per telefono judesius;
- aplikacija žaidimo metu į serverį žinutes turi siųsti tokiu dažniu, jog žaidėjai žaisdami nepajaustų trikdžių (strigimo);

2.3. Išsamus techninis aprašas

Kuriant aplikaciją ir bandant pritaikyti išmanųjį įrenginį valdymui judesiu, buvo pasitelkti įrenginio prietaisų duomenys – akcelerometro, giroskopo bei magnetometro. Pasinaudojome *Android Sensor* klasės pasisukimo vektoriaus kintamuoju, kuri pasinaudojusi anksčiau paminėtais trimis sensoriais programiškai sujungia juos ir gauną pasisukimą. Turėdami pasisukimo vektorių, iš jo galėjome gauti pasisukimo

matricą. Kadangi pasisukimas būna apskaičiuotas pasaulinės orientacijos atžvilgiu, mums reikėjo gauti santykinį pasisukimą. Tai padarėme kalibravimo metu išsaugodami pasukimo matricą, o visas likusias gautas matricas daugindami iš išsaugotos matricos inversijos. Norėdami įdiegti 6 skirtingas kalibravimo galimybes, iš pradinio pasisukimo išgauname telefono globalią poziciją ir parenkame 2 iš trijų ašių, su kuriomis toje pozicijoje būtų patogiausiai valdyti žaidimo objektą. Visi skaičiavimai ir logika atliekami aplikacijoje.

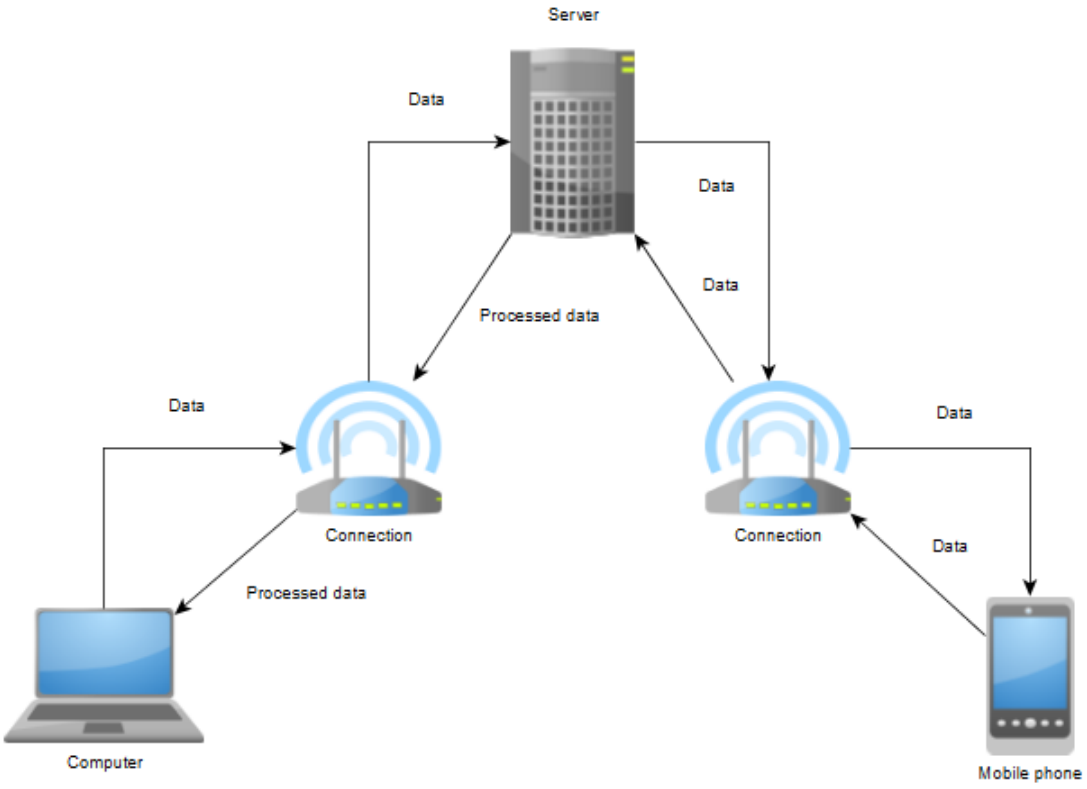
Susisiekimui su serveriu ir žinučių siuntimui naudojome *WebSocket* klasę, kuri įgalina prisijungimo bei žinučių siuntimo galimybes į nurodytą adresą. Bendravimas su serveriu vyksta žinutes siunčiant JSON tarptautiniu formatu. Pačios žinutės yra šabloninės ir turi du laukus: tipo bei duomenų lauką. Siunčiant tam tikro tipo duomenis, nurodomas tipas, o duomenų lauko masyve yra talpinami visi reikalingi duomenis. Kitame gale serveris nuskaito tipo lauką ir pagal jį ieško reikalingų kintamųjų duomenų lauko masyve.

Serveris...

3. SISTEMOS UML DIAGRAMOS

3.1. Išdėstymo diagrama

Technologinio projekto išdėstymo diagrama:

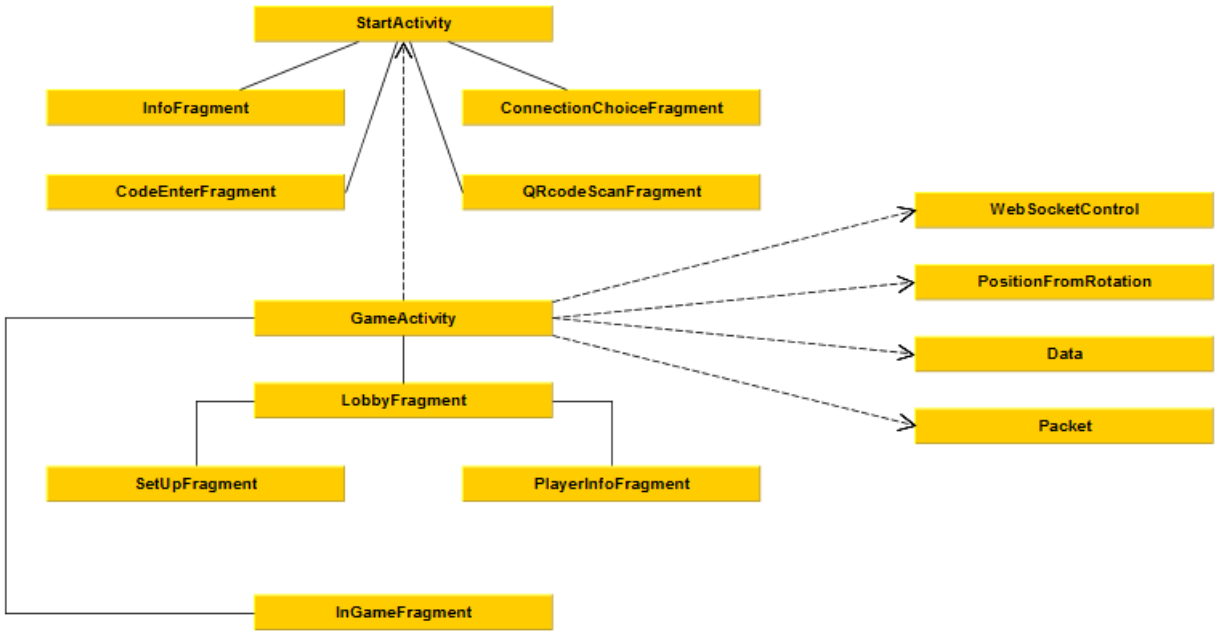


1 pav. Technologinio projekto išdėstymo diagrama

Žaidimo metu mobilusis įrenginys siunčia duomenis apie naują apskaičiuotą žaidėjo judėjimo vektorių per internetą serveriui, o šis savo atžvilgiu siunčia apdorotus duomenis apie naujas žaidėjų pozicijas bei užpieštus naujus ekrano plotus. Prieš žaidimą (sesijos puslapyje) serveris per internetą siunčia mobiliajam įrenginiui galimų ir laisvų spalvų sąrašą, o mobilusis įrenginys siunčia atnaujintą žaidėjo informaciją bei jo būseną.

3.2. Klasių diagrama

Mobiliosios aplikacijos klasių diagrama:



2 pav. Mobiliosios aplikacijos klasių diagrama

StartActivity: pradinė veikla, kuri įsijungia atidarius aplikaciją. Ji yra atsakinga už 4 pradžios fragmentų (*InfoFragment, ConnectionChoiceFragment, CodeEnterFragment, QRcodeScanFragment*) manipuliuojama ir gaunamą informaciją. Taip pat, gavusi sesijos kodą, ši veikla paleidžia žaidimo veiklą.

InfoFragment: fragmentas, atsirandantis pradžioje ir prašantis įeiti į tinklalapį, per kurį bus matomas žaidimo veiksmas.

ConnectionChoiceFragment: fragmentas, kuris suteikia pasirinkimą, kaip įvesti sesijos kodą – ar nuskanuojant QR kodą, ar suvedant jį ranka.

CodeEnterFragment: fragmentas, kuris leidžia įvesti sesijos kodą ranka.

QRcodeScanFragment: fragmentas, kuris leidžia nuskanuoti sesijos QR kodą.

GameActivity: žaidimo veikla, kuri yra sukuriama, kai pradžios veikla gauna sesijos kodą. Ši veikla naudodamasi sesijos kodu per *WebSocketControl* klasę jungiasi prie serverio, o savo *InGameFragment* fragmente naudojasi *PositionFromRotation, Data* ir *Packet* klasėmis duomenų apdorojimui bei išsiuntimui. Be *InGameFragment* fragmento, yra atsakinga už *LobbyFragment* fragmentą.

LobbyFragment: fragmentas, kuris yra tėvas dviejų kitų fragmentų (*SetUpFragment* ir *PlayerInfoFragment*) ir kuriame yra šie du fragmentai tiesiog patalpinti.

SetUpFragment: fragmentas, kuriame žaidėjas esantis sesijos puslapyje gali nusistatyti savo informaciją (slapyvardį, inicialus, spalvą).

PlayerInfoFragment: fragmentas, kuriame žaidėjas esantis sesijos puslapyje mato kitų sesijos žaidėjų informaciją.

InGameFragment: fragmentas, prie kurio pereinama kai prasideda žaidimas.

WebSocketControl: klasė, atsakinga už susijungimą ir duomenų siuntimą serveriui.

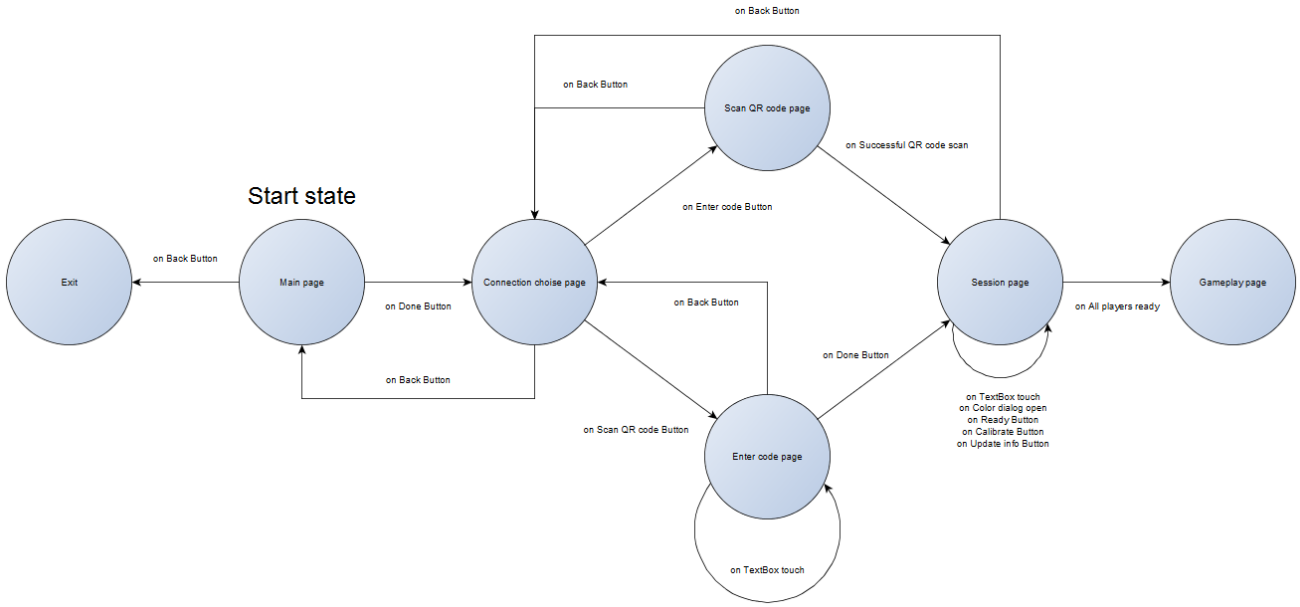
PositionFromRotation: klasė, atsakinga už sensorių duomenų apdorojimą bei žaidėjo krypties vektoriaus nustatymą ekrano atžvilgiu.

Data: klasė, kurioje yra aprašyti visi duomenys, reikalingi apie žaidėjus, ir kurie yra sugrupuoti, jog būtų patogu jų konkrečias kopijas siųsti serveriui pagal nurodytą tipą.

Packet: klasė, kuri turi du laukus – tipo ir duomenų, ir kurios objektai JSON formatu siunčiami į serverį.

3.3. Būsenų diagrama

Mobiliosios aplikacijos būsenų diagrama:



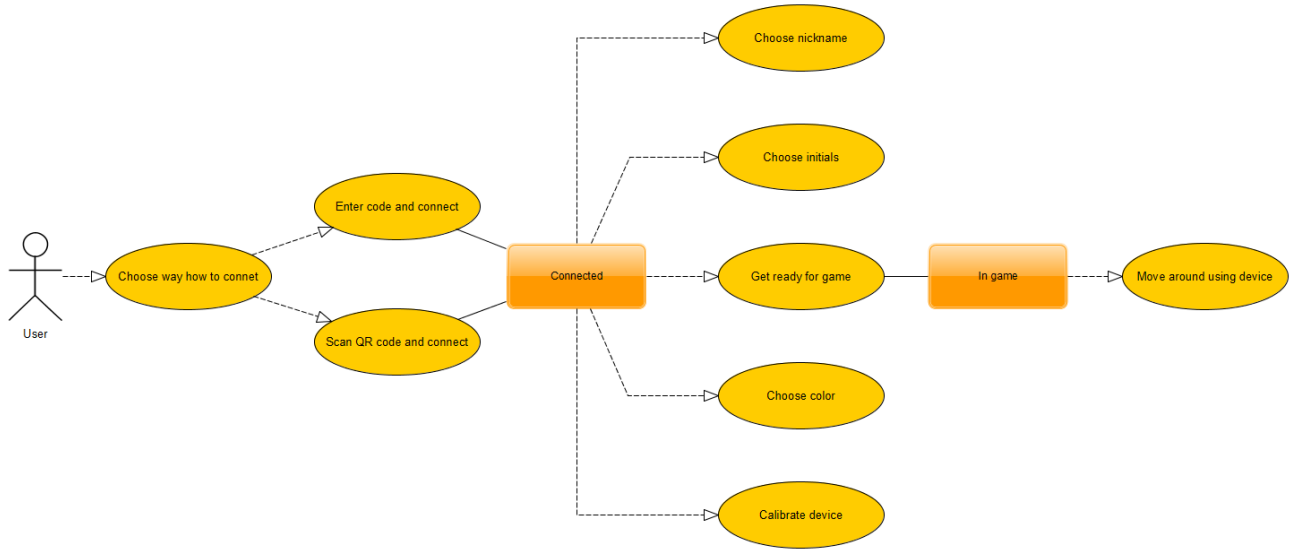
3 pav. Mobiliosios aplikacijos būsenų diagrama

Atsidarius aplikaciją, atsiduriama pagrindiniame puslapyje. Iš jo toliau paspaudus mygtuką galima atsirasti prisijungimo puslapyje, kuriame galima pasirinkti prisijungimo būdą ir taip patekti į vieną iš

dviejų prisijungimo būsenų. Po prisijungimo patenkama į sesijos būseną, kurioje būnant galima keisti žaidėjo duomenis. Iš sesijos būsenos išeinama tuomet, kai visi žaidėjai pasiruošę ir prasideda žaidimas – pereinama į žaidimo būseną. Iš kiekvienos būsenos galima pereiti į prieš tai buvusią paspaudus „atgal“ mygtuką esantį ant mobiliojo įrenginio.

3.4. Panaudos atvejų diagrama

Mobiliosios aplikacijos panaudos atvejų diagrama:



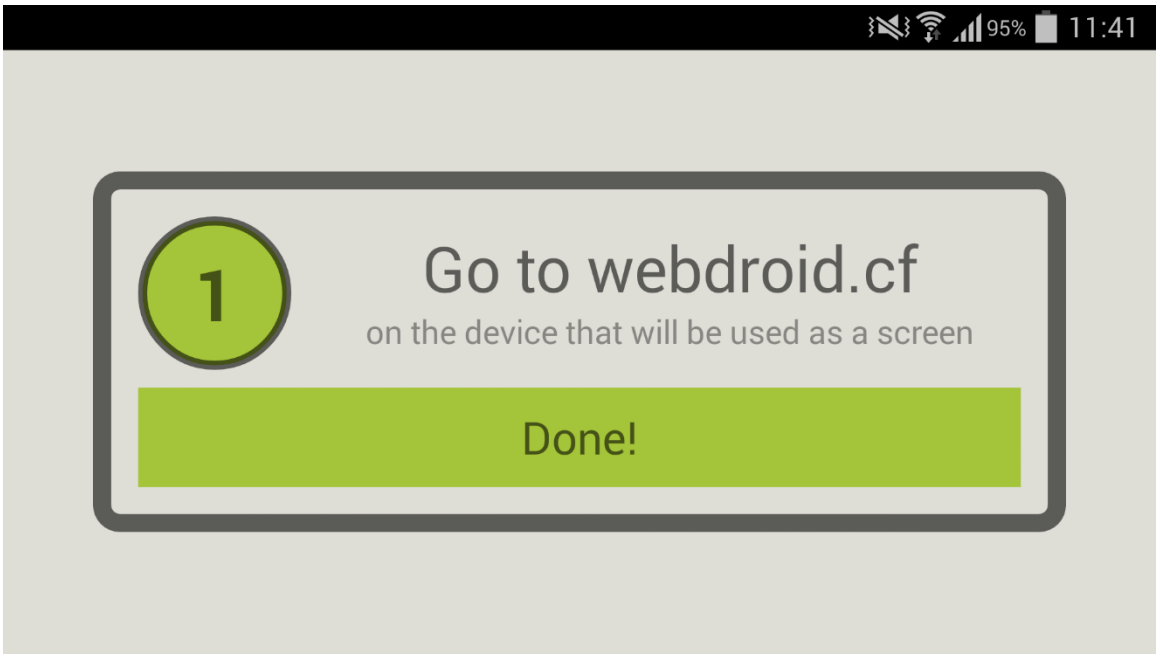
4 pav. Mobiliosios aplikacijos panaudos atvejų diagrama

Žaidėjas, įsijungęs aplikaciją, gali pasirinkti prisijungimo būdą – įvedant sesijos kodą ranka arba nuskaitant sesijos QR kodą. Žaidėjui prisijungus, jis įmetamas į sesijos puslapį, kuriame jis gali nurodyti informaciją apie save – slapyvardį, inicialus, taip pat pasirinkti spalvą bei susikalibruoti mobilųjį įrenginį jam patogiausia pozicija. Viską atlikęs, jis gali pranešti apie pasirengimą nuspausdamas pasirengimo mygtuką. Žaidimo metu, žaidėjas sukiodamas savo mobilųjį įrenginį, gali valdyti savo rutuliuką.

4. GRAFINĖ VARTOTOJO SĄSAJA

4.1. Aplikacijos grafinė vartotojo sąsaja

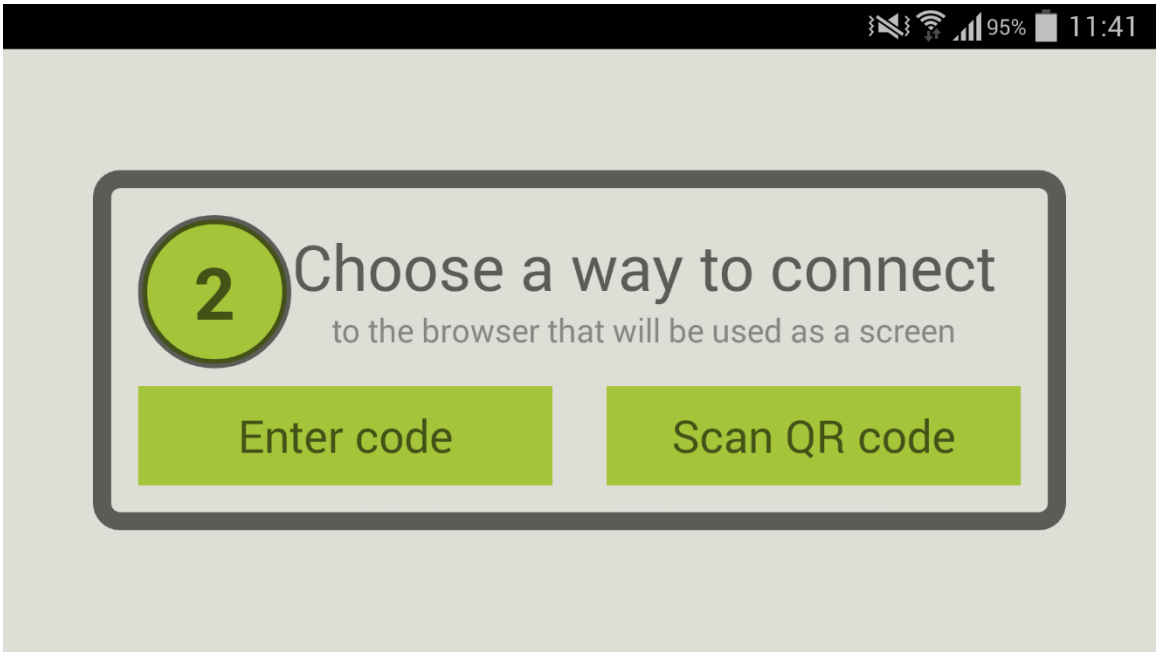
Aplikacijos grafinė sąsaja yra paremta fragmentų panaudojimu – kiekvienas skirtingas puslapis ar dalis jo yra atskiras fragmentas, kurie yra padalinti ir sudėti į dvi pagrindines veiklas: pradinę ir žaidimo. Pati sąsaja yra skaidrių-tipo, kadangi negalima peršokti iš vieno fragmento į bet kurį kitą, o tiesiog atliekant veiksmus einama į priekį, o su „atgal“ mygtuku, esančiu ant mobiliojo įrenginio, einama atgal. Taip pat, visa grafinė sąsaja yra horizontaliu užrakintu režimu, t. y. vertikalus pasukimas būtų ignoruojamas. Pradinės veiklos informacinio fragmento grafinės sąsajos vaizdas atrodo štai taip:



5 pav. Informacinio fragmento grafinės sąsajos vaizdas

Informacinis fragmentas kreipia žaidėją link webdroid.cf puslapio, kurį atsidaręs per pasirinktą įtaisą galės jame matyti žaidimą.

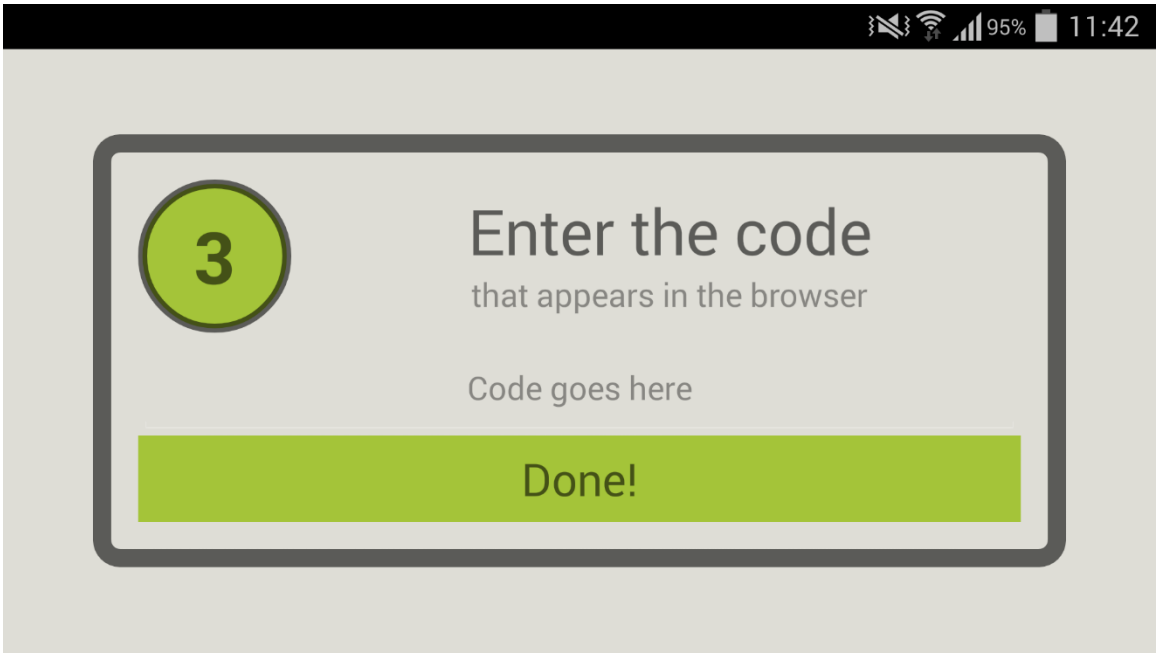
Po informacinio fragmento seka prisijungimo prie sesijos pasirinkimo fragmentas. Šio fragmento grafinės sąsajos vaizdas atrodo šitaip:



6 pav. Prisijungimo pasirinkimo fragmento grafinės sąsajos vaizdas

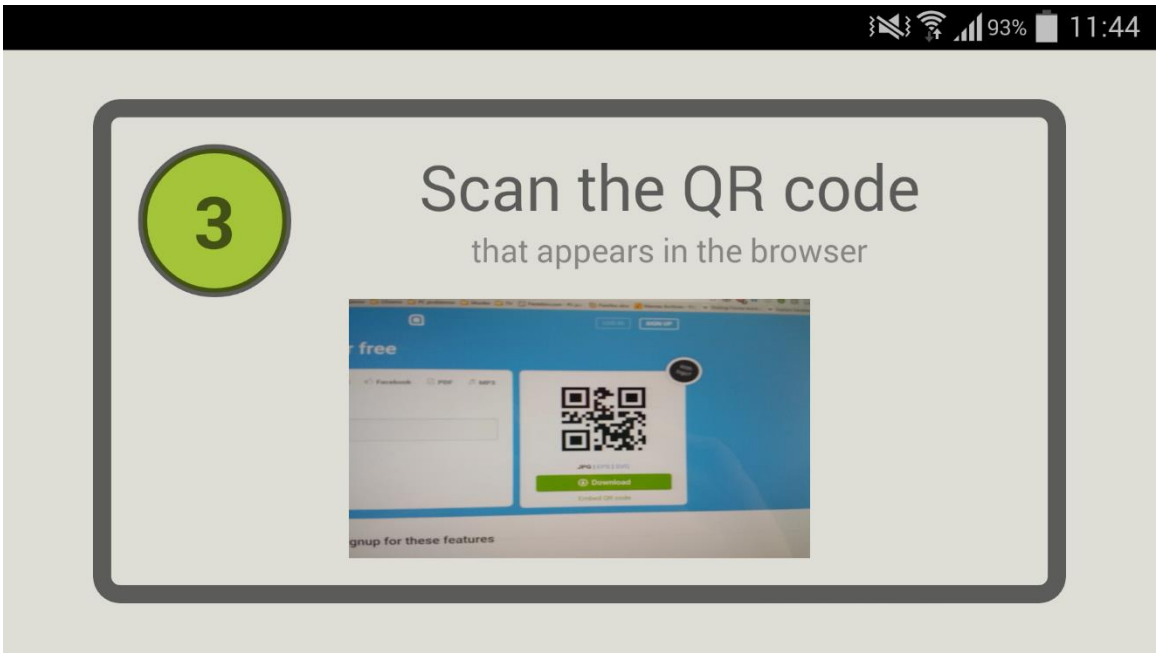
Prisijungimo pasirinkimo fragmente galima pasirinkti, kokiu būdų bus susijungiama su naršyklės sesija – įvedant sesijos kodą ar nuskaitant sesijos QR kodą.

Pasirinkus rankinį sesijos kodo įvedimą, atsidaro sesijos kodo įvedimo fragmentas, su tekstiniu lauku kodui įvesti. Šio fragmento grafinės sąsajos vaizdas atrodo štai taip:



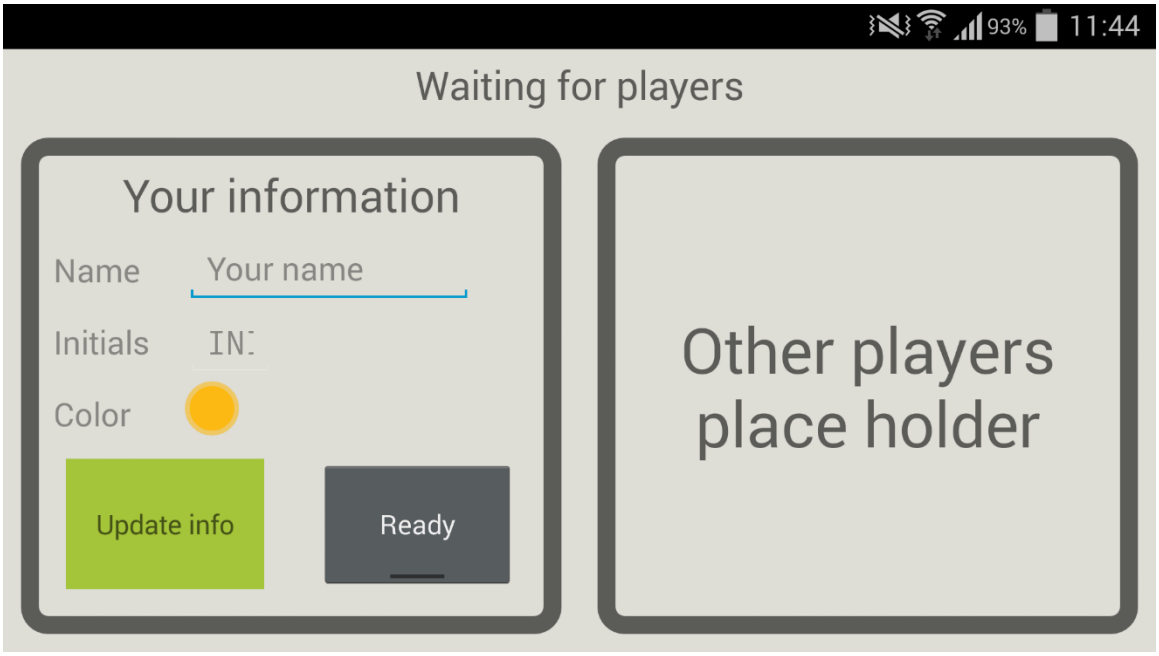
7 pav. Sesijos kodo įvedimo fragmento grafinės sąsajos vaizdas

Pasirinkus automatinį sesijos kodo įvedimą, atsidaro sesijos QR kodo nuskaitymo fragmentas, su QR kodo skaitytuvu, kuris naudojasi kamera, jog galėtų nuskaityti QR kodą. Šio fragmento grafinės sąsajos vaizdas atrodo šitaip:



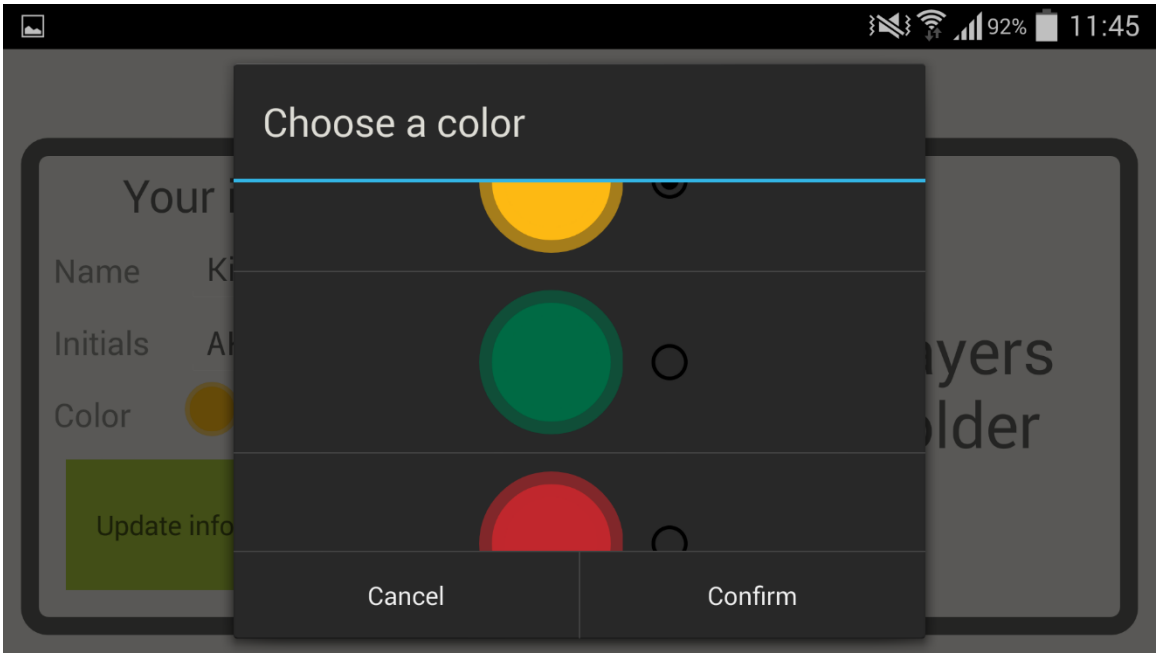
8 pav. Sesijos kodo nuskaitymo fragmento grafinės sąsajos vaizdas

Įrašius ar nuskaičius sesijos kodą, žaidėjas yra nukreipiamas į žaidimo laukimo kambario fragmentą, kuris sudarytas iš informacijos keitimo sekcijos bei kitų žaidėjų sekcijos. Bendras laukimo kambario fragmento grafinės sąsajos vaizdas:



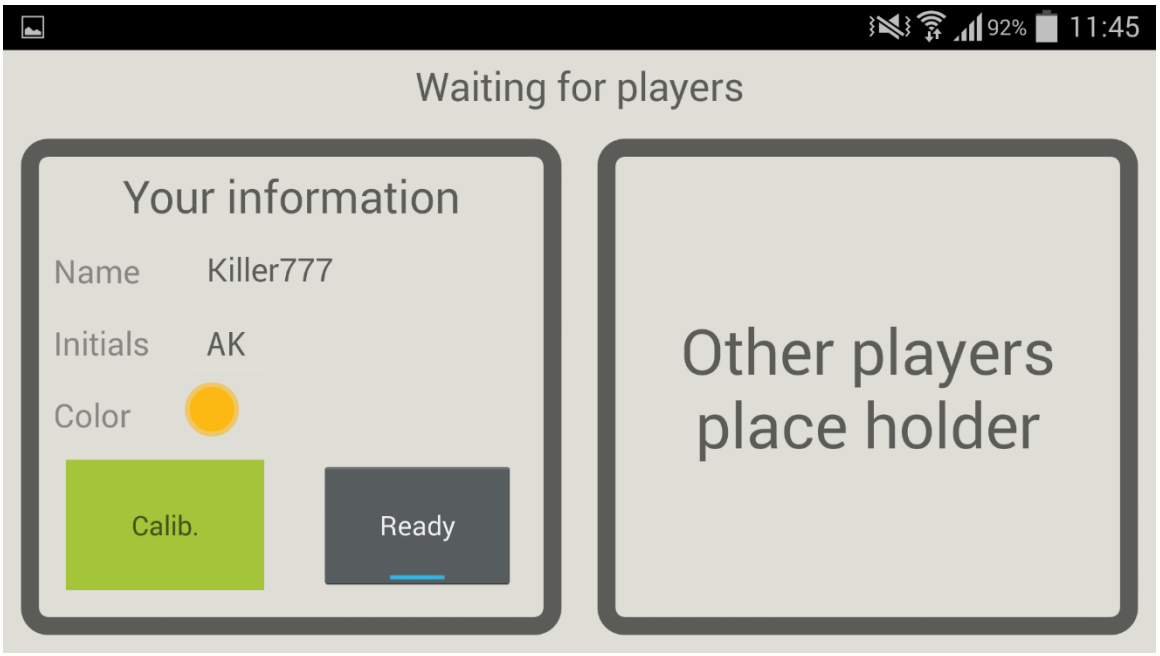
9 pav. Laukimo kambario fragmento grafinės sąsajos vaizdas

Informacijos keitimo sekcijoje šalia *Name* ir *Initials* laukų paspaudus ant tekstinių laukų galima įvesti žaidėjo slapyvardį ir inicialus. Paspaudus ant spalvos, atsidaro dialogo langas, kurio pakalba galima išsirinkti norimą spalvą. Grafinis sąsajos vaizdas atidarius spalvų dialogo langą atrodo šitaip:



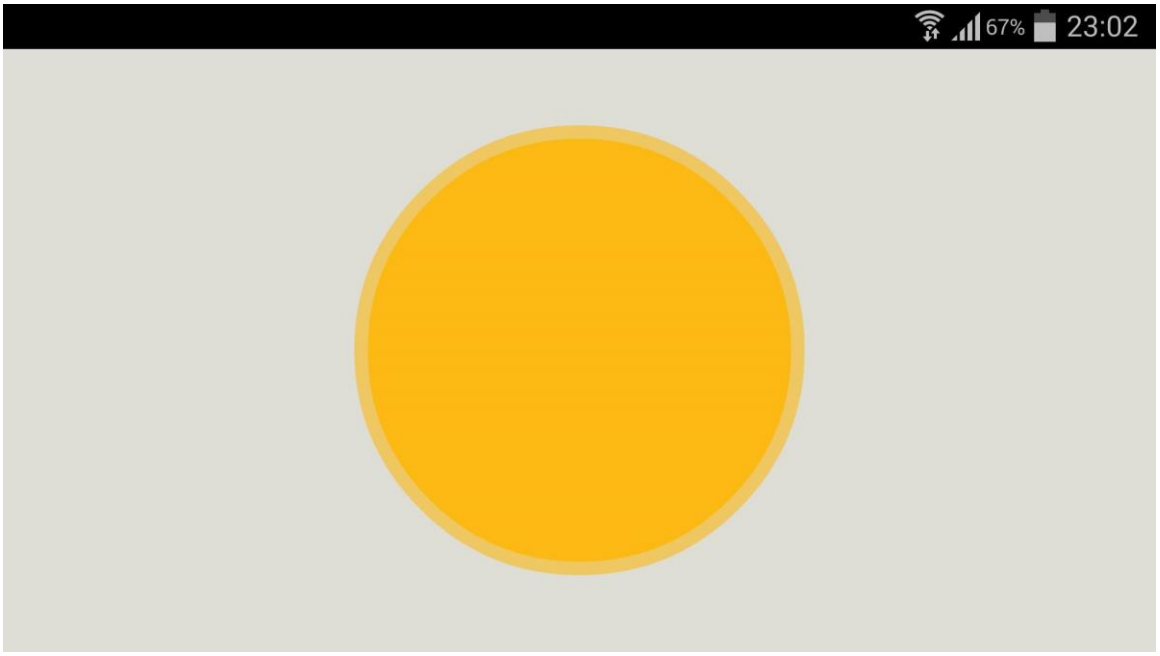
10 pav. Spalvos pasirinkimo dialogo grafinės sąsajos vaizdas

Nuspaudus mygtuką *Ready* (žaidimo laukimo kambario fragmente), pasirodo kalibravimo mygtukas, su kuriuo galima susikalibruoti telefoną ekrano atžvilgiu. Grafinės sąsajos vaizdas suvedus duomenis bei paspaudus *Ready* mygtuką atrodo šitaip:



11 pav. Laukimo kambario fragmento grafinės sąsajos vaizdas nuspaudus *Ready* mygtuką

Kai prasideda žaidimas, yra atidaromas žaidimo fragmentas, kuriame yra tiesiog pavaizduotas pasirinktos spalvos rutuliukas. Grafinės sąsajos vaizdas (mūsų atveju pasirinkus geltoną rutuliuką) atrodo štai taip:



12 pav. Žaidimo fragmento grafinės sąsajos vaizdas

4.2. Svetainės grafinė vartotojo sąsaja

5. ALGORITMAI

5.1. Mobiliojo įrenginio santykinio pasukimo apskaičiavimas

Mobiliojo įrenginio globalų pasukimą galima nustatyti iš įrenginio sensorių duomenų. Pasisukimo vektorius suteikia globalaus pasisukimo informaciją, kurią galima paversti į pasisukimo, kitaip vadinamąją posūkio matricą. Posūkio matrica – tai 3x3 matrica, kuri gali nusakyti visus įmanomus objekto pasisukimus. Kai posūkio matrica yra vienetinė – objektas yra nepasisukęs. Tad norėdami apskaičiuoti santykinį pasukimą, mes turėjome iš pradinio globalaus pasisukimo (kalibravimo pozicijos) gauti vienetinę matricą. Tai galima padaryti, padauginus globalaus pasisukimo matrica su jos pačios invertuota matrica. Kaip toliau pastebėjome, kiekvienos kitos posūkio matricos dauginimas iš sukalibruotos pradinės globalaus pasisukimo matricos inversijos generuoja matricą, kurios nulinis pasisukimas (pradinis) ir yra sukalibruota pradinė globalaus pasisukimo matrica.

Santykinio pasukimo apskaičiavimo algoritmas:

1 lentelė. Santykinio pasukimo apskaičiavimo algoritmas

Santykinio pasukimo apskaičiavimas metode *calibrate(float[] rotationMatrix)*

```
public void calibrate(float[] rotationMatrix) {
    /**
     * Unnecessary calculations
     *
     float det = rotationMatrix[0] * rotationMatrix[4] * rotationMatrix[8] +
     rotationMatrix[2] * rotationMatrix[3] * rotationMatrix[7] +
     rotationMatrix[1] * rotationMatrix[5] * rotationMatrix[6] -
     rotationMatrix[2] * rotationMatrix[4] * rotationMatrix[6] -
     rotationMatrix[1] * rotationMatrix[3] * rotationMatrix[8] -
     rotationMatrix[0] * rotationMatrix[5] * rotationMatrix[7];
     *
     */

    invertedCalibratedRotationMatrix[0] = (rotationMatrix[4] * rotationMatrix[8] -
                                           rotationMatrix[5] * rotationMatrix[7]);
    invertedCalibratedRotationMatrix[1] = (rotationMatrix[2] * rotationMatrix[7] -
                                           rotationMatrix[1] * rotationMatrix[8]);
    invertedCalibratedRotationMatrix[2] = (rotationMatrix[1] * rotationMatrix[5] -
                                           rotationMatrix[2] * rotationMatrix[4]);
    invertedCalibratedRotationMatrix[3] = (rotationMatrix[5] * rotationMatrix[6] -
                                           rotationMatrix[3] * rotationMatrix[8]);
    invertedCalibratedRotationMatrix[4] = (rotationMatrix[0] * rotationMatrix[8] -
                                           rotationMatrix[2] * rotationMatrix[6]);
    invertedCalibratedRotationMatrix[5] = (rotationMatrix[2] * rotationMatrix[3] -
                                           rotationMatrix[0] * rotationMatrix[5]);
    invertedCalibratedRotationMatrix[6] = (rotationMatrix[3] * rotationMatrix[7] -
                                           rotationMatrix[4] * rotationMatrix[6]);
    invertedCalibratedRotationMatrix[7] = (rotationMatrix[1] * rotationMatrix[6] -
                                           rotationMatrix[0] * rotationMatrix[7]);
    invertedCalibratedRotationMatrix[8] = (rotationMatrix[0] * rotationMatrix[4] -
                                           rotationMatrix[1] * rotationMatrix[3]);
}
```

Norint gauti invertuotą matricą, reiktų ne tik atlikti skaičiavimus nurodytus algoritme, bet ir kiekvieną iš matricos narių padalinti iš pradinės matricos determinanto. Tačiau, taisyklingos posūkio matricos determinantas yra lygus vienetui, tad papildomi determinanto skaičiavimai yra nereikalingi.

PS. nurodytas metodas nėra pilnas, t. y. jame palikti tik aiškinamo algoritmo fragmentai.

5.2. Mobiliojo įrenginio kalibravimo pozicijos nustatymas

Norint pasinaudoti gautais santykinės pasisukimo matricos rezultatais ar globalios pradinės matricos rezultatais, reiktų pasisukimo matricos rezultatus konvertuoti į labiau suprantamą formatą. Buvo pasirinktas seniau mobiliuosiuose įrenginiuose naudotas metodas, kuris visą pasisukimą įvardina trimis ašimis (požiūriais): horizontaliu pasisukimu, vertikaliu pasisukimu ir apsisukimu apie save patį (angl. *yawn*, *pitch*, *roll*). Tai galima padaryti dviem būdais: pasinaudojus matematinėmis formulėmis, kurios iš pasisukimo matricos apskaičiuoja minėtas tris ašis, arba pasinaudoti jau įdiegtu metodu sensoriaus klasėje. Buvo pasirinktas pirmasis būdas, kadangi kalibravimo pozicijos nustatymui užtenka apskaičiuoti *pitch* ir *roll* reikšmes. Algoritmo logika paprasta – pagal globalų telefono pasisukimą (kitaip tariant, poziciją Žemės atžvilgiu) nustatyti, kurios dvi iš trijų ašių bus naudojamos pasisukimų perskaičiavimui į dvimates koordinates.

Kalibravimo pozicijos nustatymo algoritmas:

2 lentelė. Kalibravimo pozicijos nustatymo algoritmas

Kalibravimo pozicijos nustatymas metode *initiateOrientation(float[] rotationMatrix)*

```
private void initiateOrientation(float[] rotationMatrix) {
    double pitch = Math.atan2(rotationMatrix[7], rotationMatrix[8]) *
                    180 / Math.PI;
    double roll = Math.atan2(
                   -rotationMatrix[6], Math.sqrt(Math.pow(rotationMatrix[7], 2) +
                   Math.pow(rotationMatrix[8], 2))) * 180 / Math.PI;
    if(roll > 45)
        this.pointer.pointerOrientation = ORIENTATION_PLUS_X;
    else if(roll < -45)
        this.pointer.pointerOrientation = ORIENTATION_MINUS_X;
    else if(pitch > -135) {
        if(pitch > -45) {
            if(pitch > 45) {
                if(pitch > 135)
                    this.pointer.pointerOrientation = ORIENTATION_PLUS_Z;
                else
                    this.pointer.pointerOrientation = ORIENTATION_MINUS_Y;
            } else
                this.pointer.pointerOrientation = ORIENTATION_MINUS_Z;
        } else
            this.pointer.pointerOrientation = ORIENTATION_PLUS_Y;
    } else
        this.pointer.pointerOrientation = ORIENTATION_PLUS_Z;
}
```

5.3. Mobiliojo įrenginio santykinio pasukimo perskaičiavimas į dvimates koordinates

Kaip jau minėta anksčiau, norint pasinaudoti matricos duomenimis reikia ją konvertuoti į trijų ašių formatą. Perskaičiuojant pasukimo matricą į dvimates koordinates geriausia apsiskaičiuoti visų trijų ašių reikšmes. Tačiau priešingai nei su pradine globalia pasisukimo matrica, kurią mes gavome iš sensoriaus, santykinę pasisukimo matricą reikia apsiskaičiuoti patiems. Norint pasinaudoti antru būdu, reiktų apskaičiuoti visą santykinę matricą, o norint pasinaudoti pirmu metodu – tik 5 iš 9 narių, kas sutaupo nemažai skaičiavimų laiko. Tad vėl buvo pasirinktas pirmas metodas. Algoritmas, priklausomai nuo globalaus pradinio pasisukimo, perskaičiuoja dvi iš trijų ašių į dvimates koordinates. Algoritmo bazinis maksimalus reikalingas pasukimas, norint išgauti tolimiausia koordinatę – 45 laipsniai, tačiau šis skaičius gali būti sumažintas didinant jautrumo kintamojo (*this.pointer.sensitivity*) dydį.

Santykinio pasukimo perskaičiavimo į dvimates koordinates algoritmas:

3 lentelė. Santykinio pasukimo perskaičiavimo į dvimates koordinates algoritmas

Santykinio pasukimo perskaičiavimas į dvimates koordinates metode *processRotation(float[] rotationMatrix)*

```
public void processRotation (float[] rotationMatrix) {
    relativeRotationMatrix[0] = invertedCalibratedRotationMatrix[0] *
    rotationMatrix[0] + invertedCalibratedRotationMatrix[1] * rotationMatrix[3] +
    invertedCalibratedRotationMatrix[2] * rotationMatrix[6];

    relativeRotationMatrix[3] = invertedCalibratedRotationMatrix[3] *
    rotationMatrix[0] + invertedCalibratedRotationMatrix[4] * rotationMatrix[3] +
    invertedCalibratedRotationMatrix[5] * rotationMatrix[6];

    relativeRotationMatrix[6] = invertedCalibratedRotationMatrix[6] *
    rotationMatrix[0] + invertedCalibratedRotationMatrix[7] * rotationMatrix[3] +
    invertedCalibratedRotationMatrix[8] * rotationMatrix[6];

    relativeRotationMatrix[7] = invertedCalibratedRotationMatrix[6] *
    rotationMatrix[1] + invertedCalibratedRotationMatrix[7] * rotationMatrix[4] +
    invertedCalibratedRotationMatrix[8] * rotationMatrix[7];

    relativeRotationMatrix[8] = invertedCalibratedRotationMatrix[6] *
    rotationMatrix[2] + invertedCalibratedRotationMatrix[7] * rotationMatrix[5] +
    invertedCalibratedRotationMatrix[8] * rotationMatrix[8];

    double yawn = Math.atan2(
                  relativeRotationMatrix[3], relativeRotationMatrix[0]) *
                  180 / Math.PI;
    double pitch = Math.atan2(
                   relativeRotationMatrix[7], relativeRotationMatrix[8]) *
                   180 / Math.PI;
    double roll = Math.atan2(
                  -relativeRotationMatrix[6],
                  Math.sqrt(Math.pow(relativeRotationMatrix[7], 2) +
                  Math.pow(relativeRotationMatrix[8], 2))) * 180 / Math.PI;

    double x;
    double y;
```

```
    int orientation;

    if(this.pointer.pointerType == LINEAR_POINTER) {
        if((orientation =
            this.pointer.pointerOrientation - ORIENTATION_PLUS_X) < 1) {
            x = (pitch / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
            y = (roll / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
        }
        else if ((orientation =
                  this.pointer.pointerOrientation - ORIENTATION_MINUS_Y) < 2) {
            x = -(roll / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
            y = (pitch / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
        }
        else {
            orientation = this.pointer.pointerOrientation - ORIENTATION_MINUS_Z;
            x = -(yawn / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
            y = (pitch / 45) *
                this.pointer.sensitivity * Math.pow(-1, orientation);
        }
    }


    if(x > 1)
        x = 1;
    else if(x < -1)
        x = -1;

    if(y > 1)
        y = 1;
    else if(y < -1)
        y = -1;
}
```

PS. nurodytas metodas nėra pilnas, t. y. jame palikti tik aiškinamo algoritmo fragmentai.

5.4. Žaidėjo pozicijos apskaičiavimas priimant krypties vektorių

Žaidime mobiliojo įrenginio siunčiamos dvimatės koordinatės yra skirtos nurodyti žaidėjo judėjimo kryptį, o ne pačia žaidėjo poziciją – t. y. žaidėjai juda kryptimi, kuria yra pasuktas mobilus įrenginys ir greičiu, kuris yra tiesiogiai proporcingas kampui, kuriuo mobilus įrenginys yra pasuktas. Tad apskaičiuojant sekančia žaidėjo poziciją, yra saugoma buvusi žaidėjo pozicija ir pridedama vektoriaus bei maksimalaus greičio, esant maksimaliam kampui, sandauga.

Žaidėjo pozicijos apskaičiavimo algoritmas:

4 lentelė. Žaidėjo pozicijos apskaičiavimo algoritmas

Žaidėjo pozicijos apskaičiavimas

```
players.forEach(function(pl, pID) {
    pl.lastPos.x = pl.vec.x * speed + pl.lastPos.x;
    pl.lastPos.y = pl.vec.y * speed + pl.lastPos.y;

    if(pl.lastPos.x > gameWidth)
                pl.lastPos.x = gameWidth;
    if(pl.lastPos.x < 0)
                pl.lastPos.x = 0;
    if(pl.lastPos.y > gameHeight)
                pl.lastPos.y = gameHeight;
    if(pl.lastPos.y < 0)
                pl.lastPos.y = 0;


    drawPlayer(pl.lastPos, pl.color);
});
```

6. TESTAVIMAS

7. IŠVADOS

Kurdami šį technologinį projektą, išmokome *Android* programėlių kūrimo pagrindus, įsisavinome grafinės vartotojo sąsajos programėlėse ypatumus. Taip pat išmokome sėkmingai apdoroti mobiliojo telefono sensorius ir juos pritaikyti kurtame žaidime. Be to, išbandėme ir sėkmingai pritaikėme programėlių galimybės prisijungti prie serverių ir keistis su jais informacija. Susipažinome ir išmokome panaudoti atviro standarto formatą *JSON*, skirtą perduoti duomenų objektus. Išbandėme ir pritaikėme QR kodo teikiamas galimybes, realizuodami nuskaitymą ir generavimą. Susipažinome ir įsisavinome serverių kūrimo pagrindus panaudojant *Node.js* technologiją. Kurdami interneto svetainės interaktyvumą, įsisavinome *HTML, Javascript* programavimo kalbų pagrindus.

Šis technologinis projektas nors ir turėjo aiškią ir paprastą struktūra, tačiau apimtis buvo didesnė, nei iš pradžių buvo manyta. Lanksčių metodų naudojimas projekto pradžioje leido gan greitai įsibėgėti tiek komandai, kuri projekto darbus aiškiai pasiskirstydavo tarp jos narių, tiek pačiam projektui. Projektas išryškino kiekvieno nario stipriąsias puses, kurias reiktų toliau puoselėti, ir silpnąsias puses, kurias reiktų patobulinti.