Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Nico Hämäläinen

Riichi-mahjong-tekoäly

Metropolia Ammattikorkeakoulu

Tietotekniikka

Ohjelmistotuotanto

Insinöörityö

28.4.2019

Tekijä Otsikko Sivumäärä Aika	Nico Hämäläinen Riichi-mahjong-tekoäly 33 sivua 2.5.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotuotanto
Ohjaaja	Lehtori Miikka Mäki-Uuro

Insinöörityö käsittelee mahjongpelin suunnittelua ja toteutusta. Toteutettu peli pyrkii noudattamaan Euroopan mahjongliiton ylläpitämiä kilpailusääntöjä mahjongin riichi-variaatiosta.

Peli toteutetaan JavaScriptillä, ja graafinen käyttöliittymä käyttää HTML5-grafiikkaa. Socket.io-kirjastoa hyödynnetään kommunikoimaan verkkopelissä palvelin- ja asiakasohjelman välillä.

Verkkopelaamisen lisäksi peliin toteutetaan mahdollisuus käyttää tekoälypelaajia sekä arvioidaan, kuinka hyviä tai ihmisen kaltaisia siirtoja ne tekevät.

Vaikka tekoälypelaajat onnistuvat joskus tekemään voittavaan tilanteeseen johtavia siirtoja, voidaan lopputuloksena todeta, että mahjong on hyvin monimutkainen eivätkä ainakaan yksinkertaiset tekoälypelaajat pelaa läheskään yhtä tehokkaasti kuin kokeneet ihmispelaajat.

Avainsanat	Videopeli, moninpeli, javascript, node.js, HTML5, mahjong, riichi, tekoäly

Author Title Number of Pages Date	Nico Hämäläinen Artificial Intelligence for Riichi Mahjong 33 pages 2 May 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Mäki-Uuro Miikka, Senior Lecturer

This thesis aims to cover the steps of designing and implementing a multiplayer mahjong videogame based on the European Mahjong Association's Riichi Competition Rules.

The game is implemented using JavaScript, and its graphical user interface is done with HTML5. Socket.io library is used for the online multiplayer when communicating between server and host applications.

In addition to the online multiplayer, two players models controlled by artificial intelligence are implemented. The thesis evaluates how good they play and how their decisions seem to compare against human players.

Even though the computer-controlled players can sometimes make moves that end up winning, it can be concluded that mahjong is very complicated, and simple artificial intelligence will not play as well as a good human player does.

Keywords	Videogame, multiplayer, javascript, node.js, HTML5, mahjong, riichi, artificial intelligence

Sisällys

1 Johdanto

Mahjong on Aasian ulkopuolella vähemmän tunnettu neljän pelaajan uhkapeli. Pelissä pelaajat toimivat vuorotellen nostamalla ja poistamalla yhden tiilen kädestään, tavoitteena saada sopiva yhdistelmä ennen muita. Peli voi aluksi vaikuttaa monimutkaiselta. Siihen liittyy paljon satunnaisuutta mutta myös taitoa. Kyseessä ei siis ole samaa nimeä kantava, yhden pelaajan pasianssimainen kuvioparien napsauttelupeli.

Nykyään mahjongin pelaamiseen keskittyviä seuroja on muodostettu ympäri maailmaa, ja pelivälineitä myydään lautapelikaupoissa myös Suomessa. Mahjong vaikutti mielenkiintoiselta peliltä, ja viime vuosien uutiset tekoälyistä saivat pohtimaan mahjongia pelaavan tekoälyn toteutusta.

Insinöörityössä rakennettiin kokeilumielessä neljän pelaajan mahjongpeli sekä siinä toimivia tekoälykonsepteja. Tavoitteena oli oppia lisää sekä tekoälyistä että pelien kehittämisestä. Neljän pelaajan vaatimus antoi myös hyvän syyn selvittää, miten pelin voisi toteuttaa pelattavaksi verkon yli.

Pelin logiikka pyrittiin mallintamaan mahdollisimman tarkasti pelin sääntöjä noudattaen. Toteutuksessa hyödynnettiin MVC-mallia, joka mahdollisti useiden erilaisten pelaajaohjainten toteutuksen. Sekä palvelin- että asiakasohjelma toteutettiin JavaScriptillä, jotta käyttäminen onnistuisi helposti eri laitteilla. Peliin toteutettiin ihmispelaajalle tarkoitettu ohjain ja käyttöliittymä sekä kaksi erilaista tekoälyn ohjaamaa ohjainta. Tekoälyohjaimia toteutettiin kaksi erilaista, jotta niiden suorituskykyä voisi vertailla.

Ensimmäisessä luvussa esitellään mahjongsääntöjä ja siihen liittyviä yleisimpiä termejä. Toisessa luvussa suunnitellaan ja toteutetaan videopelin logiikka aiemmin esiteltyjä sääntöjä noudattaen sekä suunnitellaan graafinen käyttöliittymä. Kolmas luku tarkastelee lähempää pelin ja pelaajan vuorovaikutusta, ja tuo mukaan pelaamisen internetin yli. Neljäs luku käy kevyesti läpi yleisiä asioita tekoälyistä sekä tutkii, kuinka Monte Carlo ja kattavampi puuhaku algoritmeina sopivat mahjongiin. Tekoälyjen vahvuuksia arvioidaan voittoprosentin ja vastaavassa tilanteessa ihmispelaajan tekemiin siirtoihin vertaamalla. Lopuksi pohditaan projektin jatkokehitystä: mitä vielä puuttuu tai tehtäisiin toisin.

2 Mahjong yleisesti ja sen pelaaminen verkossa

Mahjong on Kiinasta maailmalle levinnyt neljän pelaajan lautapeli. Pelistä on olemassa useita maakohtaisia sääntövariaatioita ja pelimuotoja. Suomalaisen Mahjong Finland -järjestön mukaan Suomessa ja muualla Euroopassa turnaussääntöinä käytetään Euroopan Mahjong-liitto EMA:n ylläpitämiä mahjongsääntöjä, joista eräs suosittu variantti on riichi. Säännöt tunnetaan nimellä Riichi Competition Rules (RCR). Mahjongpeli ja tekoälyt, joita tässä insinöörityössä käsitellään, noudattavat RCR:n sääntöjä.

Mahjongliittojen järjestämät turnaukset pelataan yleensä perinteisenä lautapelinä eikä videopelinä. On olemassa muutamia verkkosivuja ja videopelejä, jotka mahdollistavat usean pelaajan mahjongin digitaalisena. Näistä suosituin on tenhou.net, mutta sen ja käytännössä kaikkien muidenkin sivustojen palvelut ovat tarjolla vain japaniksi.

2.1 Termit

Suuri osa mahjongia käsittelevästä materiaalista on tarjolla englanniksi, kiinaksi tai japaniksi. Englanniksi kirjoitetussa materiaalissa käytetään paljon termejä, jotka on lainattu pääosin kiinalaisista tai japanilaisista termeistä.

Tiili (tile) ja tiilet (tiles) ovat pelivälineenä käytetyt 136 tiiltä. Tiilet voivat olla pelilaattoja tai -kortteja, joissa on erikseen selkä- ja kuvapuoli. Tiiliä on 34 erilaista, joista jokainen esiintyy neljä kertaa. 34 erilaista tiiltä ovat perinteisesti bambutiilet numerosta 1 numeroon 9, kolikkotiilet numerosta 1 numeroon 9, merkkitiilet numerosta 1 numeroon 9. Lisäksi löytyy neljän eri ilmansuunnan tiilet (itä, etelä, länsi, pohjoinen) sekä punaisen, valkoisen ja vihreän lohikäärmeen tiilet.

Setti (set) pitää sisällään jonon, kolmoset tai neloset. Valmiissa kädessä on tavallisesti neljä settiä ja yksi pari.

Pari (pair) tarkoittaa kädessä samaan aikaan olevaa kahta samanlaista tiiltä. RCR määrittelee erikseen kaksi muusta kuin neljästä setistä ja parista koostuvaa voittokättä. Toinen poikkeuksellisista voittokäsistä on nimeltään seitsemän paria, ja se koostuu nimensä mukaisesti seitsemästä erilaisesta parista. Käsi antaa pelaajalle kaksi voittokerrointa.

Varastaminen (steal) tarkoittaa pelaajalle hyödyllisen tiilen viemistä toiselta pelaajalta. Varastaminen on mahdollista voittamiseen. Varastaminen minkä tahansa setin täydentämiseen ilman, että kädestä tulee valmis, muuttaa käden avoimeksi kädeksi.

Kolmoset (pung, pon) ovat setti, jossa on kolme samaa tiiltä. RCR:n mukaan pelaajan on sanottava ääneen ”pung” tai ”pon”, jos hän haluaa varastaa toiselta pelaajalta kolmannen tiilen. Kolmosen varastaminen on mahdollista vain, jos kädessä on valmiiksi kaksi tai kolme samanlaista tiiltä.

Neloset (kong, kan) ovat neljän saman tiilen setti. Pelaajan on RCR:n mukaan sanottava ääneen ”kong” tai ”kan” silloin, kun neljä samanlaista tiiltä halutaan muodostaa yhdeksi setiksi. Neloset lasketaan valmiissa kädessä yhdeksi setiksi, joten sen muodostaminen aiheuttaa erikoistoimenpiteitä. Neljäs tiili voi olla joko itse nostettu tai varastettu toiselta pelaajalta. Pelaajan on poikkeuksellisesti nostettava ylimääräinen tiili käteensä, jotta neljän setin ja yhden parin muodostaminen olisi edelleen mahdollista. Neloset on niiden muodostamisen jälkeen pidettävä näkyvillä pöydässä eikä settiä ole mahdollista purkaa tai muuttaa.

Mahjongissa on kolmen eri maan (suite) tiiliä. Maat ovat bambu, kolikko ja merkki.

Jonolla (chow, chi) tarkoitetaan settiä, jossa on kolme numerojärjestyksessä peräkkäistä saman maan tiiltä. Toiselta pelaajalta varastaessa sanotaan sääntöjen mukaan ”chow” tai ”chi”. Ainoastaan omaa vuoroa edeltävän pelaajan tiilen voi varastaa jonoon. Tuulet ja lohikäärmeet eivät ole mitään maata, joten niitä ei voi käyttää jonoissa.

Muuri (wall) on muodostelma tiiliä, josta pelaajat nostavat vuorollaan tiilen käteensä. Muuri on verrattavissa korttipakkaan; se on sekoitettu ja siitä nostetaan tiiliä aina samassa järjestyksessä. Pelin alkaessa ennen jakoa muurissa on kaikki 136 tiiltä.

Voittokerroin (yaku) vaikuttaa voitettujen pisteiden määrään. RCR-sääntöjen mukaan kädessä on oltava vähintään yksi voittokerroin ennen kuin sillä voi julistaa voiton. Tarkempi lista voittokertoimista käydään läpi myöhemmin.

Käsi (hand) on kunkin pelaajan tiilet, joista mahdollinen voittokäsi muodostuu. Pelaajan varastaessa tiilen toiselta pelaajalta tulee kädestä avoin käsi (open hand). Sitä ennen kyseessä on suljettu käsi (closed hand).

Riichi on voittokerroin, jonka pelaaja voi julistaa itselleen käden ollessa suljettu ja yhden tiilen päässä voittamisesta. Pelaajan on RCR-sääntöjen mukaan sanottava ääneen ”riichi” ja asetettava tuhannen pisteen tikku pelipöydälle. Tämän jälkeen pelaajan kädessä olevat tiilet on lukittu eikä niitä saa heittää pois. Pelaajan seuraavat nostot, jotka eivät johda voittoon, on asetettava pöydälle välittömästi. Kierroksen voittaja saa itselleen kaikki pöydälle sen hetkisen ja edellisten kierrosten aikana asetetut tuhannen pisteen tikut.

2.2 Säännöt ja pelin kulku

Pelin alussa tiilet sekoitetaan ja kasataan pöydälle muuriksi. Pelaajat aloittavat nostamalla muurista 13 tiiltä. Jakajana oleva pelaaja nostaa yhden ylimääräisen tiilen, jolloin hänen kädessään on 14 tiiltä. Vuoro koostuu tiilen nostosta ja poisheitosta. Ensimmäisenä jakaja valitsee kädestään yhden tiilen ja asettaa sen eteensä pöydälle. Pelivuoron saa jakajasta vastapäivään seuraavana oleva, joka nostaa muurista seuraavan tiilen ja heittää yhden tiilen kädestään pois. Peli kiertää vastapäivään, ja pelaajat toimivat kukin vuorollaan. Tavoitteena on muodostaa valmis käsi, joka koostuu neljästä kolmen tiilen setistä ja yhdestä parista. Ensimmäisenä valmiin käden saavuttanut pelaaja voittaa pisteitä muilta pelaajilta.

Voitettu pistemäärä määräytyy lopullisten tiilien ja voittokertoimien mukaan. Pelaajan nostaessa muurista käteen viimeisenä sopivan tiilen, saa pisteitä kaikilta pelaajilta. Mikäli pelaaja voittaa toisen pelaajan poisheittäneellä tiilellä, maksaa tiilen pois heittänyt pelaaja koko summan voittajalle. Pisteet liikkuvat ainoastaan pelaajien välillä, eli yhden voitto on toisen häviö. Esitetyn mallin mukainen pisteiden siirto tekee pelistä nollasummapelin [Millington 2006: 648].

Pelaajan ei aina ole pakko nostaa uutta tiiltä muurista, vaan viimeksi pois heitetty tiili on joissakin tapauksissa mahdollista varastaa. Kuka tahansa pelaaja voi varastaa kolmannen samanlaisen tiilen omaan käteensä, jos kädestä löytyy valmiiksi kaksi samaa tiiltä. Pelaaja, jolla on samasta maasta kaksi lähekkäistä tiiltä, voi varastaa itseään edeltävältä pelaajalta tiilen osaksi jonoa. Jonoon varastettu tiili voi olla kahden peräkkäisen tiilen edeltävä tai seuraava tiili (esimerkiksi 2 tai 5, mikäli pelaajalla on kädessään 3 ja 4), tai se voi olla jonon päätyjen väliin sopiva tiili (esimerkiksi 7, mikäli pelaajalla on kädessään 6 ja 8).

Varastettaessa suljettu käsi muuttuu aina avoimeksi kädeksi. Varastetut setit jätetään näkyviin auki käännettynä pelaajan käden oikealle puolelle. Varastettua settiä ei voi enää purkaa, eikä avoimesta kädestä voi enää tehdä suljettua. Osa voittokertoimista lasketaan mukaan vain suljetussa kädessä.

Säännöt kattavat myös muita erikoistilanteita, mutta niiden yksityiskohtainen läpikäynti ei ole tässä vaiheessa tarpeen.

3 Mahjong tietokoneohjelmana

Suurehkoja ohjelmistoja on mahdollista suunnitella ja toteuttaa oliomallin mukaisesti, koska se on ihmiselle luontaisin tapa tarkastella ongelmia. Järjestelmien voidaan katsoa koostuvan keskenään rajapintojen kautta keskustelevista osista. Jokainen olio mallintaa yksittäisen kokonaisuuden, jolle on määritelty tarkka vastuualue. Olio, joka toteuttaa rajapinnan, voidaan korvata toisella saman rajapinnan toteuttavalla oliolla, jolloin ohjelmiston osista saadaan uudelleenkäytettäviä. [Rintala 2003: 36-40.]

3.1 Olioiden tunnistaminen

Pelin kulun selittäminen ja samalla pelivälivälineiden tunnistaminen auttaa hahmottamaan tarvittavia olioita ja toimintoja. Pelivälineitä ovat ainakin nopat, tiilet ja pistelaskussa käytettävät tikut. Tietokoneohjelmassa noppa voidaan toteuttaa satunnaislukugeneraattorilla. Pisteet on mahdollista laskea kokonaislukuina ilman erillisiä esineitä. Pisteet kuuluvat pelin aikana pelaajille. Pelaajaan voidaan mieltää myös muita ominaisuuksia, kuten tehtävissä olevat toimenpiteet.

Pelin kulussa on mainittu myös pelaajien kädet sekä avatut setit. Avatut setit ovat pelissä näkyvissä pöydällä, joten voi olla sopivaa määritellä olioksi myös pelipöytä. Pelipöydän kautta avoin informaatio voi olla kaikille pelaajien samalla tavalla nähtävissä. Pelaajille avointa informaatiota ovat myös muurissa jäljellä olevien tiilien määrä, kädestä poistetut tiilet, Riichi-panokset, kierroksen jakaja ja pelaajien pisteet.

Mahjongissa pisteiden laskuun liittyy paljon logiikkaa. RCR:n mukaan kaikki pelaajat ovat yhdessä vastuussa siitä, että voittavan käden pisteiden maksu tapahtuu oikein. Tietokoneohjelmassa pelaajien ei tarvitse olla pisteiden laskusta vastuussa, vaan

pisteiden laskeminen ja pisteiden siirto pelaajien välillä voidaan jättää kokonaan tietokoneohjelman tehtäväksi.

On olemassa käsiä, jotka koostuvat neljästä setistä ja yhdestä parista, mutta eivät ole minkään arvoisia. Ohjelman on siis osattava laskea ainakin sen verran, että kädestä voidaan löytää voittokertoimet. Mikäli tietokoneohjelma osaa tunnistaa pöydän ja pelaajan tilojen perusteella mahdolliset voittokertoimet, on pisteiden lasku suurimmaksi osaksi jo toteutettu.

Pelitilanteen arvio on oleellista myös tekoälyalgoritmeille. Tekoälyn on usein pystyttävä arvioimaan mistä tahansa todellisesta ja simuloidusta tilasta senhetkinen voittaja. Tai jos voittajaa ei ole, niin jonkinlainen arvio siitä, onko tilanne pelaajalle hyvä vai huono. Pisteiden lasku voi siis toimia sellaisenaan myös osana tekoälyä.

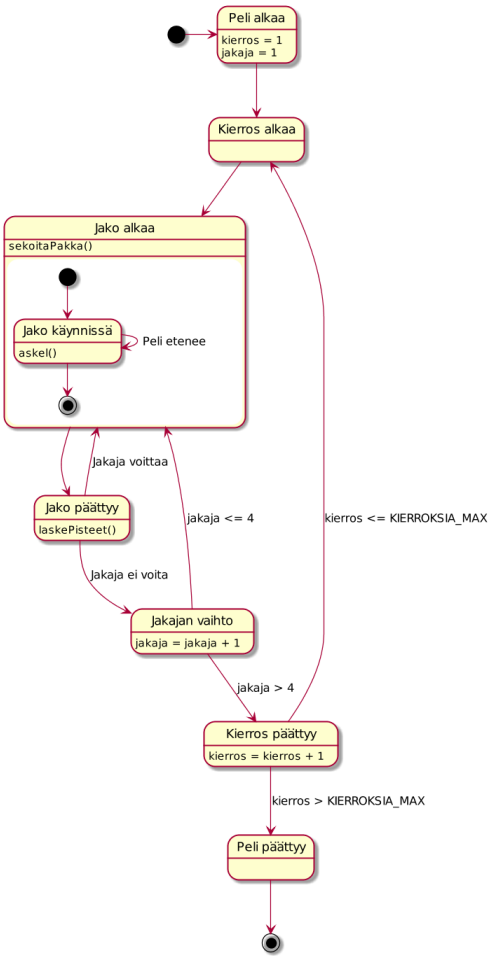
RCR määrittelee muutaman virhetilanteen, josta koituisi pelaajalle maksettavia sakkopisteitä. Määriteltyjä virheitä ovat esimerkiksi voiton julistaminen ilman yhtään voittokerrointa tai tiilien paljastaminen pelaajien käsistä tai muurista. Virhetilanteiden syntymisen voi estää ohjelmallisesti estämällä pelaajilta virheelliset siirrot.

Havaintojen perusteella seuraavat asiat voitaisiin mallintaa olioiksi:

- peli
- tiili
- pöytä
- pelaaja
- pisteet.

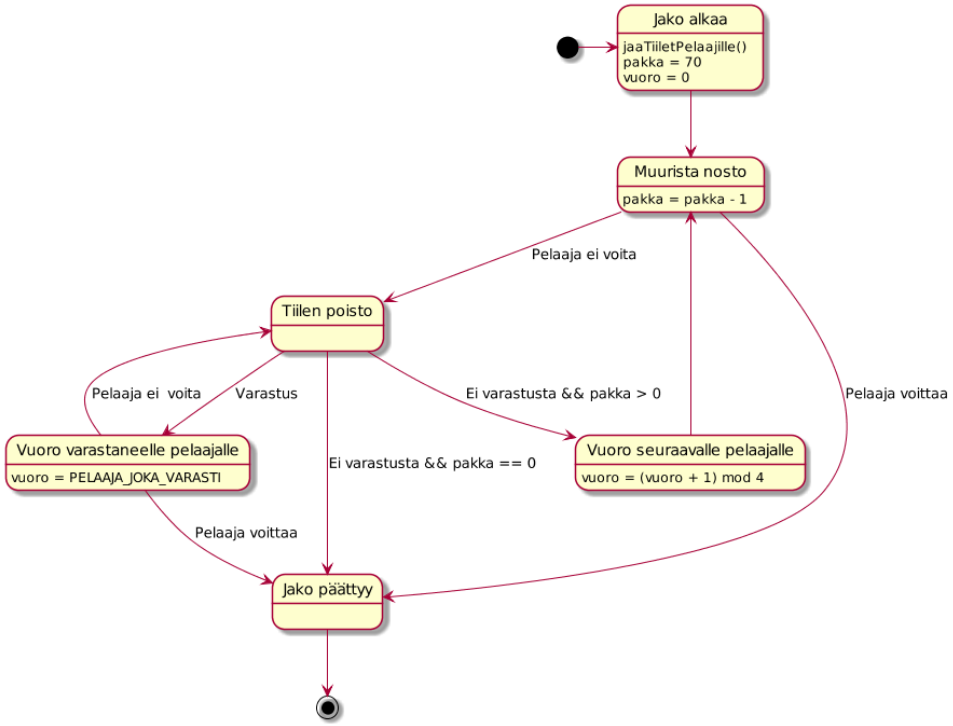
3.2 Pelin tapahtumien mallinnus

Pelin kulun pääkohdat voidaan havainnollistaa vuokaavioilla (kuvio 1), jotta saadaan parempi käsitys pelin tapahtumista ja tapahtumien järjestyksestä.



Kuvio 1. Mahjongpelin kulku vuokaaviona esitettynä.

Kokonainen pelisessio koostuu kierroksista (round). Kullakin kierroksella suoritetaan vähintään neljä jakoa (deal) siten, että jokainen pelaaja on vuorollaan jakaja. Mikäli jakaja voittaa sen hetkisen jaon, jakaja ei vaihdu. Kierros päättyy jakajan vaihtuessa neljännen kerran, eli jakajana aloittaneen pelaajan päätyessä uudelleen jakajaksi. Kuviossa 1 on esitetty kierrosten ja jakojen kulku vuokaaviona. RCR:n mukaisessa turnauspelissä pelataan kaksi kierrosta: itä- ja eteläkierros. Tarkempi kuvaus jaosta ja pelaajien toimenpiteistä näkyy kuviosta 2.



Kuvio 2. Mahjongpelin jako esitettynä vuokaaviona.

3.3 Pelimoottori

3.3.1 Tilakone

Vuokaavioiden kautta havaitaan, että peli on aina jossakin ennalta määritellyssä tilassa ja tilat vaihtuvat lineaarisesti. Varastamistilannetta lukuun ottamatta pelaajat tekevät toimintoja vain omalla vuorollaan. Vuoron aikana sallitut toiminnot ovat myös ennalta määrätyt. Pelin tila sekä siirtymiset tilojen välillä ovat siis tiedossa, joten pelilogiikan toteuttamisessa voi hyödyntää tilakone-suunnittelumallia (finite state machine).

Tilakone on suunnittelumallina geneerinen, eikä ole olemassa yhtä oikeaa tapaa sen toteutukseen. [Millington 2006: 319-320.]

Tässä mahjongpelin toteutuksessa tilakoneen tehtävänä on selkeyttää suoritettavien toimintojen etenemisjärjestys ja riippuvuus. Pelioliolle on määritelty tilat, alkutila, useita pelitilanteisiin liittyviä funktioita ja päivitysfunktio. Päivitysfunktio määrää sen, mitä pelitilanteen funktioita suoritetaan. Tilan vaihtaminen on kokonaan suoritetun funktion vastuulla.

Peli päätettiin toteuttaa JavaScriptillä. Päätökseen vaikuttivat kielen ominaisuudet ja aiempi JavaScript-kokemus. JavaScriptiä ajetaan tulkilla suoraan lähdekoodista ilman erillistä käännösvaihetta konekielelle kuten esimerkiksi Java- tai C-kielissä [Peltomäki 2001]. Koodia voi ajaa yksinkertaisimmillaan käyttöjärjestelmästä riippumatta pelkällä verkkoselaimella. Lisäksi on olemassa JavaScriptin ajamiseen tarkoitettuja ohjelmistoja, kuten node.js, jotka mahdollistavat laajemman resurssienhallinnan, kuten massamuistin käytön. Valmis koodi on siksi helposti suoritettavissa sekä paikallisesti että pilviympäristössä.

Lopullisessa toteutuksessa tarvittiin seuraavat tilat:

Taulukko 1. Tilakone-mallisen pelitoteutuksen edeltävät ja seuraavat tilat.

Edeltävä tila	**Seuraava tila**
NEW_GAME	NEW_ROUND
END_GAME	
NEW_ROUND	NEW_DEAL END_GAME
END_ROUND	NEW_ROUND END_GAME
NEW_DEAL	PICK_NEW_TILE
END_DEAL	NEW_DEAL CHANGE_DEALER
CHANGE_DEALER	NEW_ROUND END_ROUND
PICK_NEW_TILE	CHECK_TSUMO DISCARD
CHECK_TSUMO	CHECK_WIN CHECK_KAN_AFTER_DRAW
CHECK_WIN	END_DEAL CHECK_KAN_AFTER_DRAW
CHECK_KAN_AFTER_DRAW	ACTION_KAN_AFTER_DRAW CHECK_RIICHI
ACTION_KAN_AFTER_DRAW	PICK_NEW_TILE CHECK_RIICHI

CHECK_RIICHI	ACTION_RIICHI DISCARD
ACTION_RIICHI	DISCARD
DISCARD	CHECK_STEAL
CHECK_STEAL	CHECK_WIN ACTION_KAN_AFTER_DRAW PICK_NEW_TILE END_DEAL

”CHECK”-alkuisissa tiloissa tarvitaan pelaajan kyllä- tai ei-päätös ennen jatkamista, muuten tila ei vaihdu. ”CHECK”-alkuisten tilojen funktiot lukevat joka kerta pelaajan ohjaimelta, onko pelaaja tehnyt päätöksen.

3.3.2 Pisteiden lasku

Suuri osa mahjongpelin logiikasta on pisteiden laskemisessa. RCR listaa yhteensä 37 eri voittokerrointa tai “yakua”. Voittokertoimia, joissa käden on oltava suljettu, on kaksitoista. Lisäksi voittokertoimista kuusi ovat sellaisia, jotka ovat yhtä kerrointa arvokkaampia silloin, kun käsi on suljettu. Voittokertoimet voivat olla myös toisensa poissulkevat, niin ettei pienempiarvoista kerrointa enää lasketa mukaan arvokkaamman voittokertoimen löytyessä.

Useimmassa tapauksessa voittokertoimen tunnistamiseen riittää tieto pelaajan kädessä olevista tiilistä. On olemassa myös pelin tilan tai käden sekä pelin tilan yhdistelmän mukaan määräytyviä voittokertoimia.

Pisteiden lasku toteutettiin mahjongpeliin useamman funktion yhdistelmänä. Osat ovat kokonaisuudessaan

1. settien tunnistus
2. voittokerrointen listaaja
3. yksittäisen voittokertoimen tunnistus ja
4. pisteiden laskeminen.

Setit tunnistava algoritmi käy läpi listan tiiliä ja poimii yksitellen kaikki mahdolliset parit, kolmoset, neloset ja suorat. Algoritmi tekee tunnistetuista seteistä settioliot, jotka muodostuvat setin tiilistä, setin tyypistä, listasta ylijääneitä tiiliä sekä listasta lapsisettiolioita. Ylijääneet tiilet annetaan rekursiivisesti funktiolle ja tunnistetut setit lisätään edellisen settiolion lapsiksi. Lopputuloksena saadaan lista settejä, joilla voi olla lapsiolioita, joilla voi olla lapsiolioita ja niin edelleen. Tätä puurakennetta havainnollistetaan kuviossa 3, jossa esimerkkisyötteestä

- käsi ()

tunnistetaan ensimmäisellä kierroksella

- pari ()
- kolmoset ()
- suora ().

Näistä sellaiset haarat, joiden lopussa ei enää ole ylijääneitä tiiliä, voidaan lähettää edelleen voittokerrointen listaajafunktiolle. Muussa tapauksessa käsi ei ole valmis, eikä sitä tarvitse käsitellä enempää.

Kuvio 3. Settien tunnistusalgoritmin palauttama puurakenne esimerkkisyötteestä. Puun haaroista juuren ensimmäinen haara sekä juuren kolmannen haaran ensimmäinen lapsihaara johtavat valmiiksi tunnistettaviin käsiin. Oikeassa pelissä settejä löytyisi 14 tiilellä tavallisesti yksi pari ja neljä kolmen tiilen settiä.

Tunnistetut setit annetaan listana voittokertoimet tunnistavalle funktiolle, joka käy yksitellen läpi jokaisen voittokertoimen ehdot ja palauttaa listan löydetyistä kertoimista. Läpikäynnissä otetaan huomioon toisensa poissulkevat voittokertoimet niin, että arvokkaampi tarkistetaan ensin ja sen löytyessä vaihtoehtoista ei tarkisteta.

Mikäli voittokertoimia löytyy, annetaan voittokertoimet ja niitä vastaavat tunnistetut setit pisteidenlaskufunktiolle. Yli neljän kertoimen pisteet palautetaan suoraan voittokerrointen lukumäärän perusteella taulukon 2 mukaan.

Taulukko 2. Pisteiden määräytyminen voittokerrointen perusteella.

Voittokerrointen lukumäärä	**Pohjapisteet**	**Erikoisnimitys**
5	2000	Mangan
6 – 7	3000	Haneman
8 – 10	4000	Baiman
Yli 11	6000	Sanbaiman
Yli 13*	8000	Yakuman

*) Vuoden 2016 RCR:n sääntömuutoksien myötä yli 13 voittokerrointa ei enää pisteytetä Yakumaniksi. Yakuman-pisteet on mahdollista saavuttaa ainoastaan erikoiskäsillä, jotka mainitaan sääntökirjan voittokerrointaulukossa.

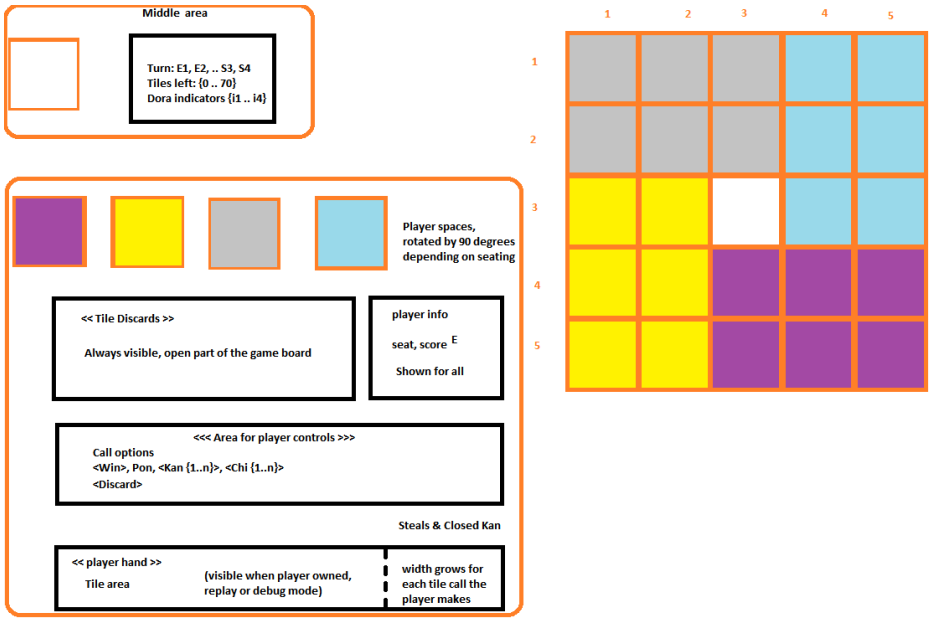
Mikäli kertoimia on neljä tai vähemmän, lasketaan settien antamat pisteet yhteen, kerrotaan voittokertoimista lasketulla kertoimella ja pyöristetään ylöspäin lähimpään sataan pisteeseen kaavalla

$$pohjapisteet = 100 \times \text{pyöristä\_ylös}\left(\frac{settipisteet \times 2^{2+voittokertoimet}}{100}\right).$$

3.4 Graafinen käyttöliittymä

Konsolitulosteista oli haastava lukea informaatiota, joten kehittämisen ja testaamisen tueksi sopiva käyttöliittymä tuli hahmoteltua pian pelimoottorin ensimmäisen version

toteuttamisen jälkeen. Tarvetta oli pelitilan hahmottamiselle sekä pelaajalle avoimen informaation näkemiselle. Konsolitulosteesta oli hidasta ja sekavaa lukea edes esimerkiksi pelaajan kädestä löytyviä tiiliä. Alkuvaiheessa otettiin huomioon myös pelaamisen mahdollistavan syötteen lähetys peliohjaimelle. Käyttöliittymää on hahmoteltu ennen toteutusta kuviossa 4.



Kuvio 4. Ensimmäinen luonnos graafisen ulkoasun asettelusta, ohjaustoiminnoista ja näytettävästä datasta.

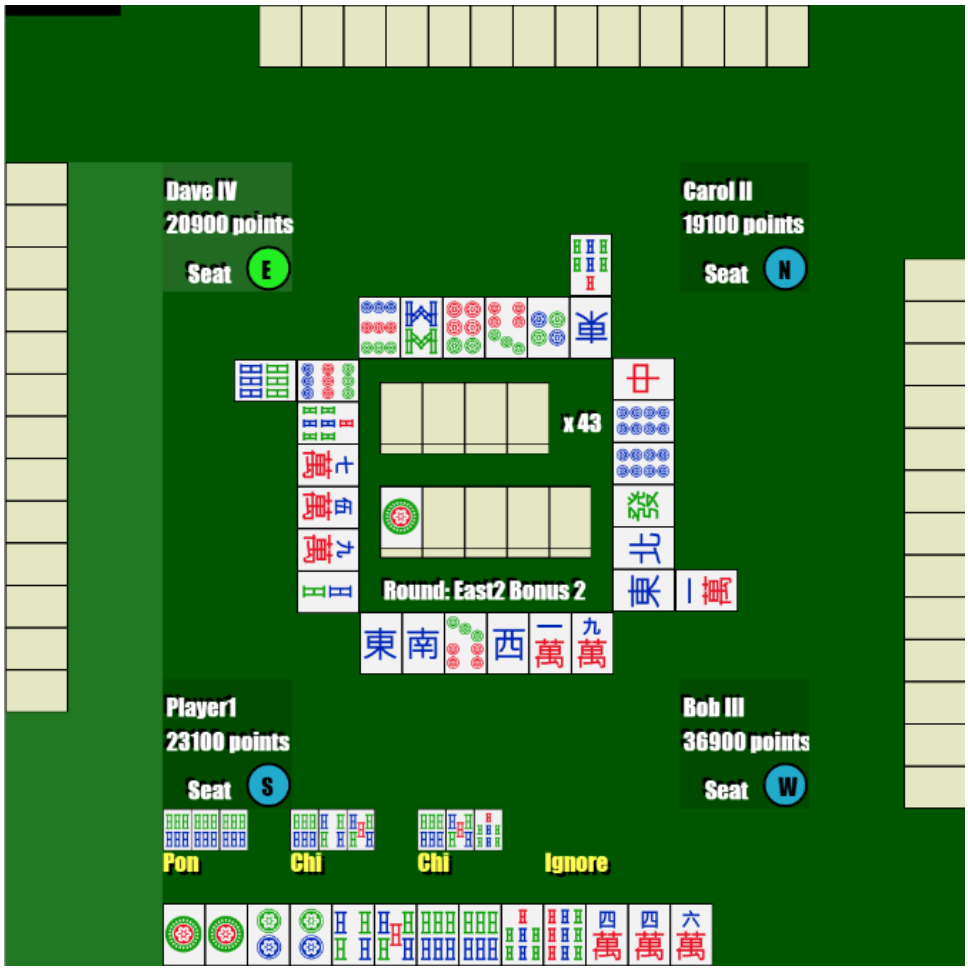
Graafinen käyttöliittymä koostuu neljästä keskenään samanlaisesta pelaajan alueesta, jotka on kierretty 90 asteen kulmassa. Asettelu vastaa neliön muotoista pöytää, jonka ympärillä istuu neljä pelaajaa. Kullakin pelaajalla on edessään omat tiilet ilman näkyvyyttä toisen pelaajan tiilten kuvapuolelle. Testausta helpottamaan haluttiin kuitenkin mahdollistaa myös piiloon jäävän informaation näkyminen. Erillisellä testausasetuksella vastustajien tiilten kuvapuolet on mahdollista saada näkyviin.

Käyttöliittymä on haluttu pitää yksinkertaisena, mutta sallivan silti pelaajalle kaikki toimenpiteet, mitä kullakin hetkellä voi tehdä. Ideoita valintojen esitystavoille ja pelaajan huomion kiinnittämiselle on otettu teoksesta ”Käytettävyyden Psykologia” [Sinkkonen ym. 2006].

Pelaajan on omalla vuorollaan yleensä ainoastaan mahdollista valita poisheitettävä tiili, mutta joskus on myös mahdollista julistaa riichi tiilen poisheiton yhteydessä, yhdistää neljä kädessä olevaa samaa tiiltä nelosiksi tai julistaa voitto nostamallaan tiilellä. Toisen

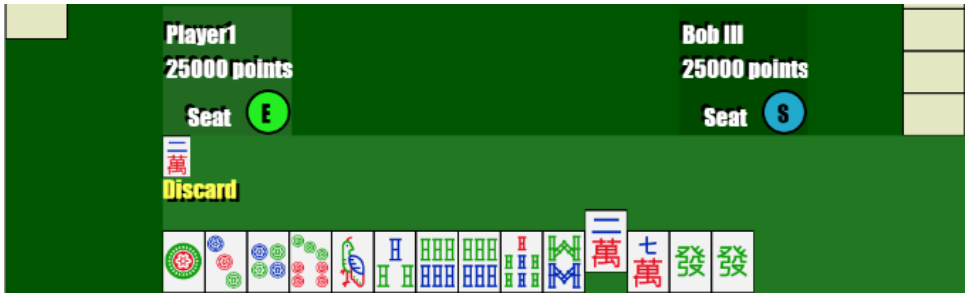
pelaajan vuoron lopuksi on myös silloin tällöin mahdollista varastaa poisheitetty tiili itselleen tai julistaa voitto.

Toimenpiteitä ei ole kerrallaan suurta määrää. Siksi kaikista mahdollisista tapahtumista ilmestyy pelaajan alueelle valinta, jota napsauttamalla sen voi suorittaa tai peruuttaa. Valintanappulan ilmestymisen ja värien on tarkoitus kiinnittää pelaajan huomio tekemään päätös. Nappulat pyrkivät selittämään itsensä tekstin ja kuvan avulla. Esimerkki useammasta samanaikaisesta valinnasta käyttöliittymässä näkyy kuviossa 5. Valintanappulan ilmestymisen lisäksi varastus- ja voittamistilanteista ilmoitetaan oletuksena myös äänimerkillä. Lisäksi aktiivisena olleen pelaajan vuoro näytetään käyttöliittymässä vaaleammalla taustavärillä.



Kuvio 5. Tilanne, jossa edellinen pelaaja ”Dave IV” on heittänyt pois bambu 6 -tiilen ja pelaajalle ”Player 1” tarjotaan käyttöliittymässä useampi vaihtoehtoinen toimenpide. Pelaajan on mahdollista varastaa bambu 6 -tiili joko kolmosiksi tai suoraksi jossa olisi bambu 4, 5, 6 tai 5, 6, 7. Pelaajan on mahdollista olla tekemättä mitään valitsemalla ”Ignore”-toiminnon, jolloin peli etenee normaalisti.

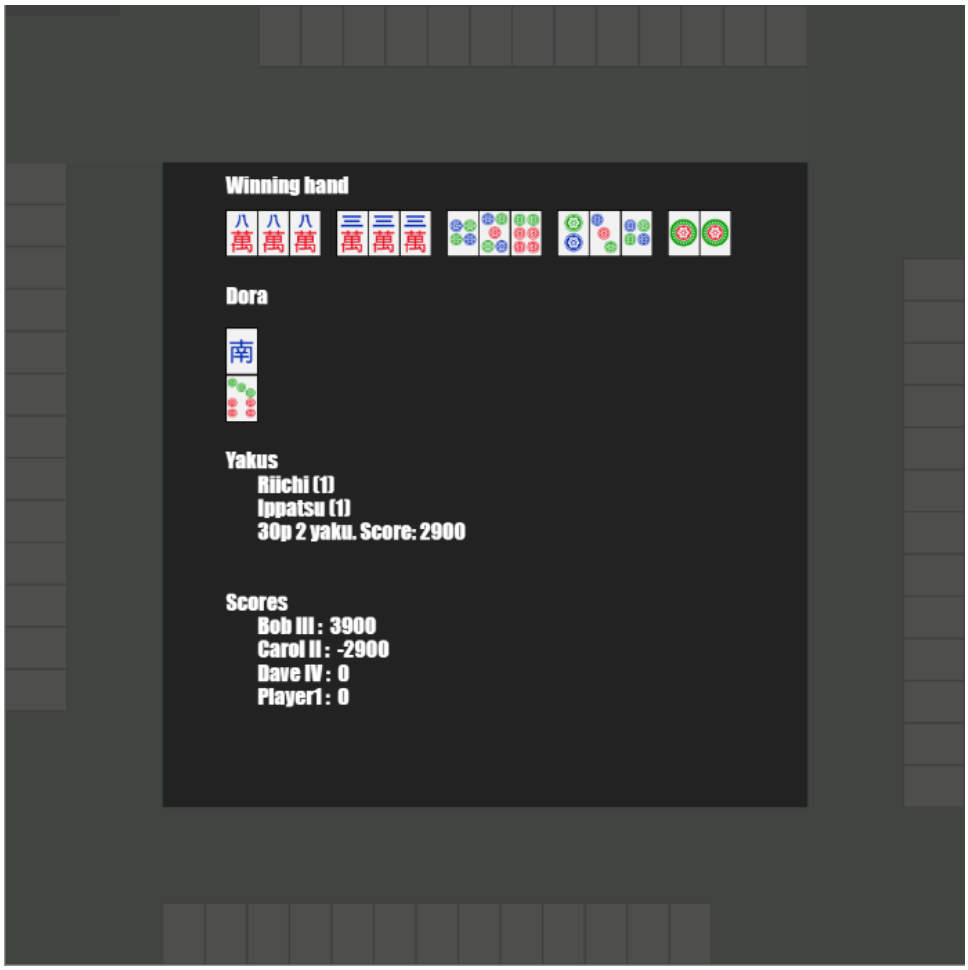
Kuviossa 6 näkyy pelaajan vuoron lopulla oleva toimenpide, jossa valitaan tiili ja heitetään se pois. Käyttöliittymässä on tarkoitus valita napsauttamalla kädestä tiili, jonka jälkeen toimintanappula päivittyy vastaamaan valittua tiiltä. Tällä estetään varsinkin kosketusnäytössä epätakka painallus ja vaaditaan pelaajaa vahvistamaan tekemänsä valinta.



Kuvio 6. Pelaajan vuoro, jossa on valittava pois heitettävä tiili. Toimintatapa ei välttämättä aukea käyttäjälle heti, mutta sillä on pyritty varmistamaan pelaajan tekemä toiminto.

Valinnan tekeminen käyttöliittymässä saa valintanapit katoamaan, joten pelaaja saa heti palautteen tekemästään toiminnosta.

Kierroksen lopussa käyttöliittymä näyttää pelaajien pisteiden muutokset, josta on esimerkki kuviossa 7.



Kuvio 7. Pisteiden muutokset kierroksen lopussa. Kuvasta käy ilmi, kuka sai pisteitä ja kuka menetti niitä. Lisäksi voittokertoimet ja pohjapisteet on mainittu erikseen. Pelaaja ”Bob III” on voittanut 2900 pistettä pelaajalta ”Carol II”. Voittaja saa myös kaikki pöytään pelatut tuhannen pisteen riichi-panokset, joten kokonaissumma on 3900 pistettä.

3.4.1 HTML5 ja JavaScript

Graafisen käyttöliittymän toteutukseen valittiin HTML5 ja JavaScript. HTML5 toimii kaikilla nykyaikaisilla selaimilla sekä mobiililaitteilla. HTML5:n avulla on mahdollista piirtää reaaliaikaista grafiikkaa Canvas-elementin avulla, josta löytyvät piirtofunktiot yksinkertaisille muodoille, tekstille ja valmiille bittikartoille. JavaScript taas mahdollistaa mm. käyttäjäsyötteen lukemisen ja piirtofunktioiden kutsumisen.

JavaScriptin tapa käsitellä käyttäjäsyötteitä verkkosivuilla on tapahtumakeskeinen (event driven). Käytännössä se tarkoittaa sitä, että esimerkiksi käyttäjän toiminnoista aiheutuvat tapahtumat on mahdollista napata kiinni ja reagoida niiden perusteella. Käyttäjän tekemiä toimintoja ovat mm. hiiren napsautukset ja liikutus, selainikkunan koon

muuttaminen tai näppäimistön painallukset. Tapahtumia syntyy käyttäjän toimien lisäksi myös verkkoselaimesta itsestään. Esimerkiksi kuvan tai sivun valmiiksi latautuminen aiheuttaa tapahtuman. Tapahtumat käyttäytyvät ”kuplimalla” lapsielementistä aina elementin vanhemmalle, kunnes kaikkein ylin vanhempi on saavutettu tai kuplinta pysäytetään koodissa. [Peltomäki: 2001.]

Mahjong-pelin graafisessa käyttöliittymässä hyödynnetään enimmäkseen hiiren napsautuksista syntyviä tapahtumia. Mikäli tapahtumasta käy ilmi, että käyttäjä on napsauttanut hiirtä valintanappulan kohdalla, lähtee käyttöliittymästä ohjaimen kautta tieto pelille, että pelaaja suorittaa valitun toiminnon.

Hiiren sijainnin tunnistaminen ei ole aina yksiselitteistä. Ikkunan (window) ja dokumenttipohjan (document) vasen yläreuna on aina koordinaatti 0, 0, mutta peliä ei aina haluta piirtää alkaen sieltä. Myös selaimen vieritys pysty- ja sivuttaissuunnassa sekä skaalaus vaikuttaa koordinaatteihin. Hiiren napsautuksen voi ottaa kiinni ja käsitellä minkä tahansa verkkosivun elementin kautta, jolloin napsautetun elementin sijainti sivulla on mahdollista ottaa huomioon. Selaimet kuten Firefox ja Chrome antavat funktiot elementin todellisen sijainnin lukemiseen, mutta esimerkiksi Internet Explorerilla täytyy itse ottaa huomioon sivun pysty- ja vaakasuuntainen vieritys. Kuvion 8 koodiesimerkissä on pitkälti kokeilun ja korjailun tulosta, mutta vaikuttaa toimivan kaikilla testatuilla selaimilla: Chromella, Edgellä, Firefoxilla, Internet Explorerilla, Operalla, sekä matkapuhelinten Android 6.0- ja iOS 7 -oletusselaimilla. Funktio pyrkii löytämään napsautetulta elementiltä e.target sen BoundingClientRect-olion, joka pitää sisällään elementin todellisen sijainnin ja koon verkkosivulla. Mikäli oliota ei löydy, pyritään lukemaan erikseen sivun pysty- ja vaakavieritys. Menetelmä mahdollistaa napsautuksen koordinaattien lukemisen luotettavasti myös vieritetyltä sivulta.

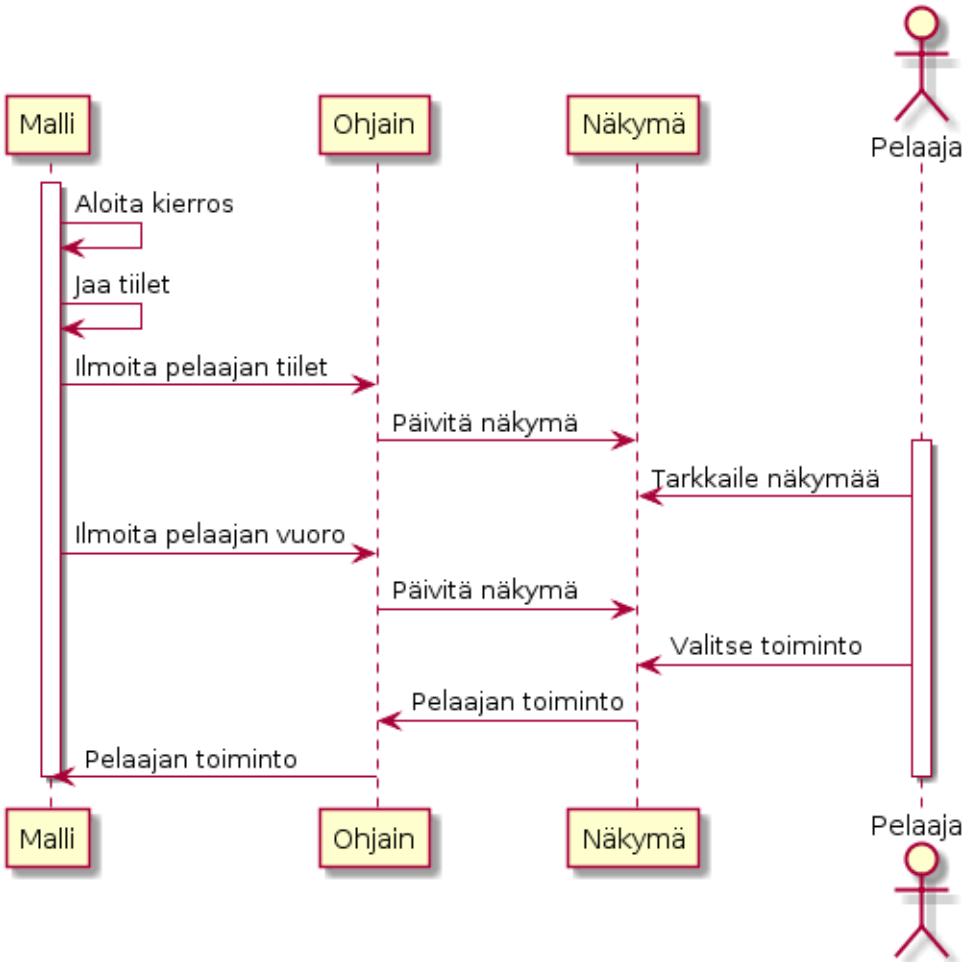
```
1  var __mousePositionAction = function(e, targetAction) {
2      if (e != undefined) {
3          var tx = e.clientX;
4          var ty = e.clientY;
5          var etb = e.target != undefined ?
6                    e.target.getBoundingClientRect()
7                    : null;
8          if (etb != undefined) {
9              tx -= etb.left;
10             ty -= etb.top;
11         } else if (document.body != undefined &&
12                    document.body.scrollLeft != undefined &&
13                    document.body.scrollTop != undefined) {
14             tx += document.body.scrollLeft;
15             ty += document.body.scrollTop;
16         } else if (tx == undefined || ty == undefined) {
17             tx = e.x;
18             ty = e.y;
19         }
20
21         targetAction(scaleUnit*tx, scaleUnit*ty);
22         if (MOUSEDEBUG) {
23             ctx.fillRect(scaleUnit*tx, scaleUnit*ty, 10, 10);
24         }
25     }
26 }
27
```

Kuvio 8. Funktio, jolle annetaan tapahtumaolio e ja kohdefunktio targetAction, jota kutsutaan tapahtumaoliosta tunnistetuilla x- ja y-koordinaateilla. Funktio piirtää myös hiiren sijaintiin neliön rivillä 23 mikäli, MOUSEDEBUG-testaustila on käytössä.

4 Peliohjaimen laajennus verkko-ohjaimeksi

4.1 MVC-arkkitehtuuri

Pelilogiikan, ihmis- ja tietokonepelaajien sekä näkymän toteutukseen sovellettiin Model-View-Controller-arkkitehtuuria (MVC). Arkkitehtuurin tarkoitus on pitää malli (model) ja näkymä (view) toisistaan riippumattomina siten, että kumman tahansa voisi vaihtaa koskematta toiseen. Mallin ja näkymän yhdistää ohjain (controller). Ohjaimella voidaan viedä tietoa käyttäjältä malliin sekä tässä kuvaillussa mahjongpelin tapauksessa myös lähettää käyttäjälle tietoa mallista. Osien vuorovaikutusta on hahmotettu kuviossa 9.



Kuvio 9. Mahjongpelin ja pelaajan vuorovaikutus MVC-arkkitehtuurilla.

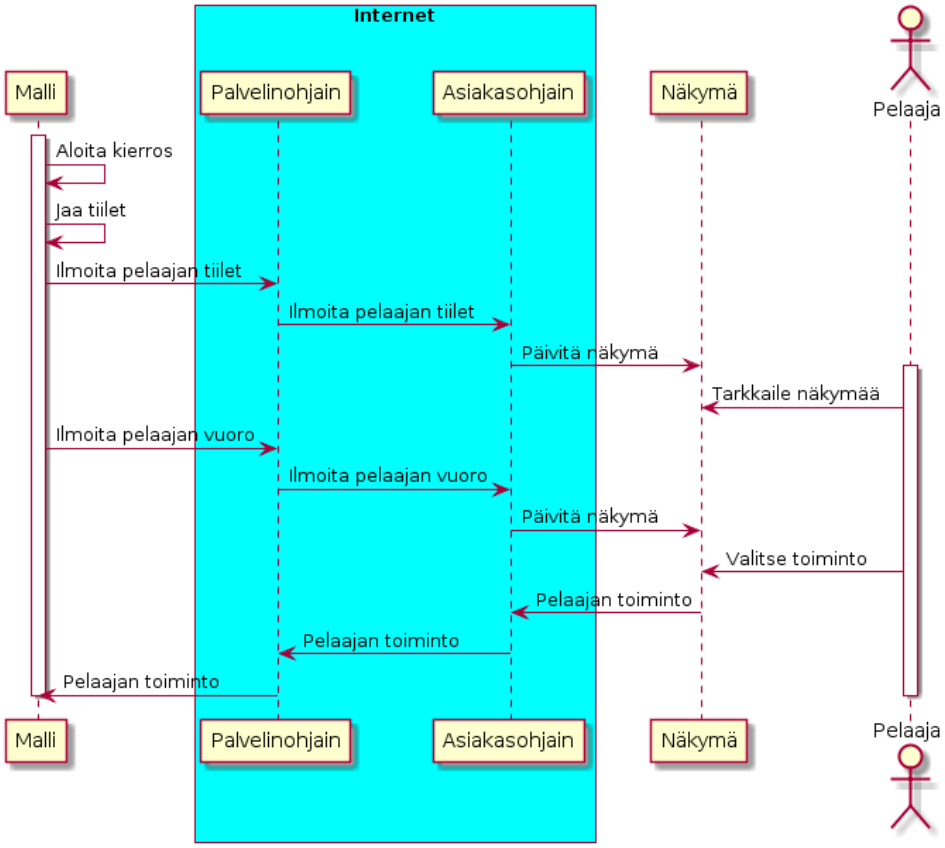
Pelaaminen onnistuu yksinkertaisimmillaan ajamalla yhtä yksittäistä ohjelmaa verkkoselaimessa. Tällöin kaikki toiminnallisuus on ajossa käyttäjällä itsellään. Kaikki informaatio on saatavilla, ja JavaScriptin tapauksessa se on myös manipuloitavissa.

4.2 Verkkosovellus

Verkkopelaamisen mahdollistamisessa käytetään socket.io-ohjelmakirjastoa.

Socket.io on saatavilla JavaScriptillä toteutettuna. Siitä löytyy Node.js-ajoympäristössä toimiva palvelinohjelma, ja sen asiakasohjelma toimii kaikissa yleisissä verkkoselaimissa. Palvelinohjelma ja asiakasohjelma voivat kommunikoida keskenään internetin yli. [Socket.io, 2019.]

Näkymän tai pelin mallin ei tarvitse erikseen tietää internettoiminnallisuuksista, mikäli ne on toteutettu ohjaimessa. Ohjain voidaan rakentaa uudelleen kahtena erillisenä komponenttina; palvelin- ja asiakasohjaimena. Molemmissa ohjaimissa käytetään socket.io-ohjelmakirjastoa. Pelin ja pelaajan vuorovaikutus internetin välityksellä on esitetty kuviossa 10.



Kuvio 10. Pelin ja pelaajan vuorovaikutus internetin yli.

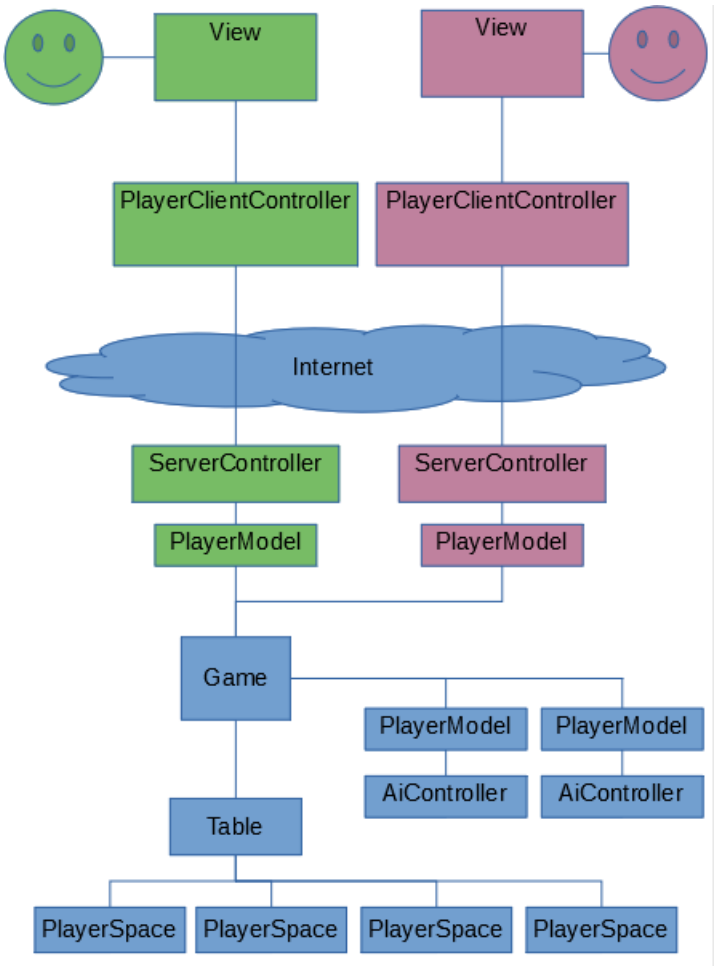
Suuri osa uusien ohjaimen alkuperäisistä funktioista oli mahdollista käyttää sellaisenaan. JavaScriptissä ei suoraan voi välittää funktio-osoitinta, mutta tunnetun olion funktiota on mahdollista kutsua pelkän funktion nimen perusteella. Tätä toiminnallisuutta on hyödynnetty niin, että palvelinohjelmalta saapuvat funktionimet suoritetaan nimen perusteella asiakasohjelmassa. Asiakasohjelma lähettää vastaavasti suoritettavien funktioiden nimiä palvelimelle. Niiden kohdalla varmistetaan kuitenkin listalta, sallitaanko funktion ajo asiakasohjelman kutsumana. Jos näin ei tehtäisi, pelaajan olisi mahdollista ohjata itse palvelinohjainta samalla tavalla kuin mallikin ohjaa. Kuviossa 11 esitellään vielä koodiesimerkki, jolla havainnollistetaan palvelimen ohjaimeen lisättyä koodia, joka mahdollistaa asiakasohjaimen lähettämien komentojen suorittamisen. Alkuperäisen

ohjaimen ja verkossa toimivan ohjaimen eroavaisuus jää tällä toteutustavalla hyvin pieneksi, koska siinä kutsutaan suoraan nimellä jo olemassa olevia funktioita.

```
/*
 * Initializations in Node.js for http and socket.io
 */
var app = express();
var server = require("http").createServer(app);
var io = require("socket.io")(server);

/*
 * Functions which the client is allowed to call
 */
var ALLOWED_REMOTE_COMMANDS =
["join",     "doWin",     "doNotWin", "doRiichi", "doNotRiichi",
 "doKan",    "doNotKan", "doPon",    "doNotPon", "doChi",
"doNotChi", "doDiscard","ignoreAction",         "playerReady"];

/*
 * Initializing the socket.io to handle data from remote clients
 */
io.on('connection', function(client){
    // Creates a controller that can be used with game
    client.controller = new MahjongServerController();
    /*
    * This gets called when client emits data. The string 'rcmd'
    * is defined as the event name and is used by the clients
    */
    client.on('rcmd', function(inData){
        if (inData != undefined) { // Null check for incoming data
            // Check if the function in "call" is allowed
            if (ALLOWED_REMOTE_COMMANDS.indexOf(inData.call) >= 0) {
                // Calls a function by name from client's controller
                client.controller[inData.call](); // JavaScript <3
            } else {
                // Logs non-allowed, empty or null calls to trace
                logError('Non-allowed function: ' + inData.call);
            }
        } else {
            // Log if rcmd event is sent from client without data
            logDebug('No Payload');
        }
    });
});

```

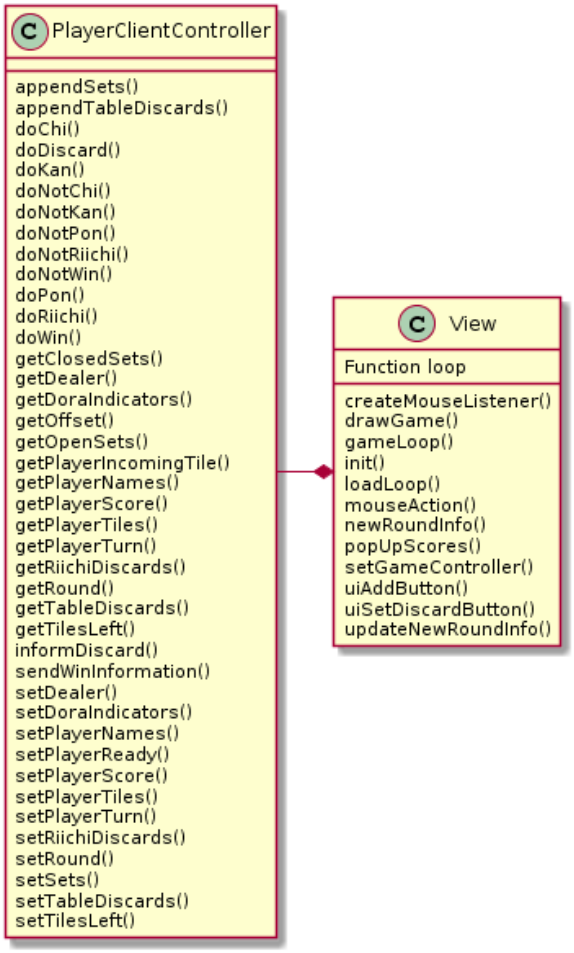
Kuvio 11. Koodiesimerkki, jossa palvelinohjain kuuntelee asiakasohjaimelta saapuvia viestejä socket.io-kirjaston avulla. Viestissä olevan ”call”-parametrin nimeämä funktio suoritetaan palvelinohjaimella.

Verkkopelitoteutuksessa esiintyvät luokat ovat pelin malli (Game), siihen liittyvät informaatiota järjestelevät apuoliot pöytä (Table) ja pelaajiin liittyvä paljastettu informaatio (PlayerSpaces), malli pelaajasta (PlayerModel) sekä palvelinpuolinen pelaajan ohjain (PlayerServerController). Tekoälytoteutuksia varten käytetään tekoälyohjainta (AiController). Luokkakaaviot palvelimen ja asiakasohjaimen komponenteista esitellään kuvioissa 12.



Kuvio 12. Luokkakaavio, joka esittelee palvelinohjelman ja asiakasohjelman luokat.

Asiakasohjelman puoleiset luokat funktioineen esitellään kuviossa 13. Asiakasohjelmassa ohjaimen luokan rakenne on lähes identtinen palvelinpuolen versioon. Ero on funktioiden toteutuksissa. Asiakasohjelman puolella lähetetään socket.io-ohjelmakirjaston avulla tietoa palvelimelle toimenpiteistä, esimerkiksi mikä tiili poistetaan tai halutaanko varastaa tarjolla olevaa tiiltä.



Kuvio 13. Asiakasohjelman luokat. Näkymä (View) -luokka sisältää lähinnä grafiikan piirtoa ja hiiren kautta ohjattujen toimintojen tunnistamista. Pelaajaohjain sisältää funktiot tiedon vastaanottoon, näkymän käyttämien tietojen päivitykseen sekä tiedon lähettämiseen palvelinohjelmalle.

Asiakasohjelman puolella ei tässä toteutuksessa varmisteta syötettä tai lähetettä sen enempää, vaan oikeellisuuden varmistus on jätetty palvelimen vastuulle. Oikeellisuudella tarkoitetaan tässä tapauksessa esimerkiksi sitä, että poistetuksi ilmoitetun tiilen on oikeasti löydyttävä pelaajan kädestä. Ratkaisulla saadaan aikaan se, ettei pelaaja voi tahallisesti tai virhetilanteen sattuessa aiheuttamaan haittaa muille pelaajille eikä pelin kulkuun yleensäkään.

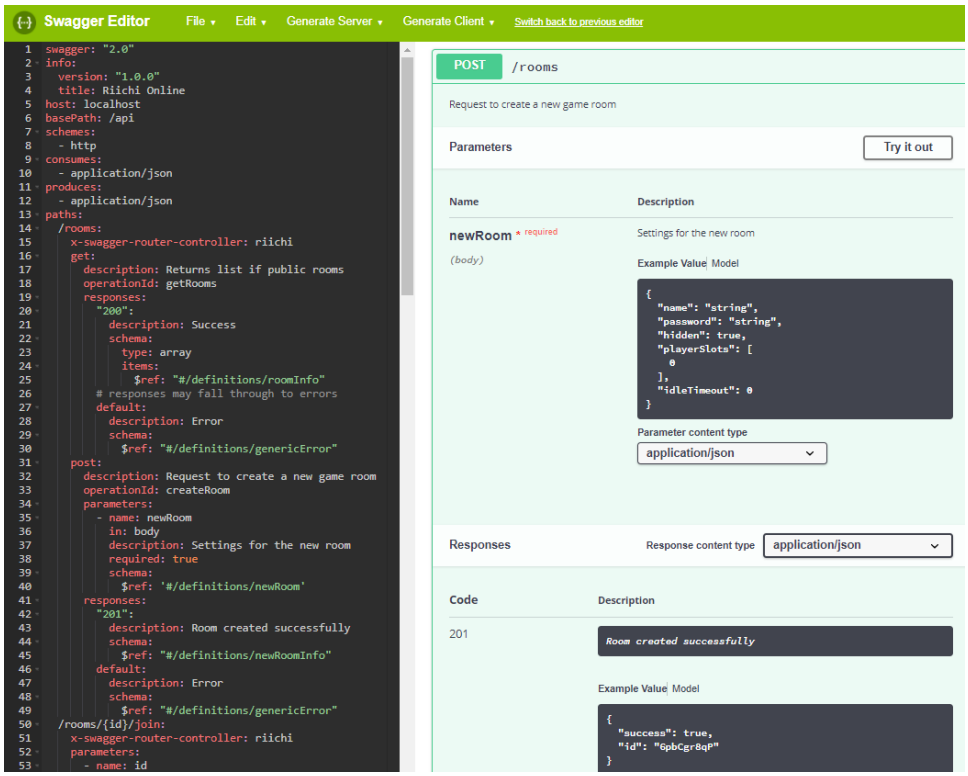
4.3 Pelaajan liittyminen mukaan verkkopeliin

Palvelimelle toteutettiin myös toiminnallisuudet uuden pelin luomiseen, luotujen pelien listaamiseen sekä peliin liittymiseen. Toimintoihin pääsee käsiksi HTTP:llä, johon

tarjotaan node.js alustalla express-ohjelmakirjaston kautta GET- ja POST-operaatioita. Tavallinen www-selain soveltuu siis sellaisenaan toimintojen käyttöön.

Toiminnoissa käytetään tietosisällön kuvaamiseen JSON-formaattia. JSON sopii erinomaisesti rakenteellisen datan kuvaamiseen.

Rajapinnasta on mahdollista julkaista dokumentaatio esimerkiksi Swagger-kuvauksena (kuvio 14), jolloin mahdollisten kolmansien osapuolten on helpompi toteuttaa oma käyttöliittymänsä palvelua vasten.



Kuvio 14. Kuvakaappaus Swagger 2.0 -muotoisesta rajapintakuvauksesta ja sen visuaalisesti esityksestä mahjongpelin huoneiden hallintaan.

Tarkoituksena oli ensisijaisesti tarjota mahdollisimman helppo ja nopea tapa päästä pelaamaan. Minkäänlaista rekisteröintimahdollisuutta tai ylimääräistä tietojen keräämistä ei toteutettu. Palvelin generoi aikaleimaan ja juoksevaan numeroon perustuvan tunnuksen pelaajan liittyessä peliin. Tunnusta osoiteparametrina käyttäessään pelaaja saa käyttöönsä ohjainpaikan palvelimelta. Tämä ohjainpaikka säilyy varattuna pelaajalle niin kauan, kuin yhteys on olemassa, joten haittaa ei aiheudu edes muun pelaajan yrittäessä käyttää samaa linkkiä. Yhteyden tilaa on mahdollista valvoa automaattisesti socket.io-kirjastolla.

Mikäli yhteys katkeaa, pelaaja voi liittyä takaisin peliin vaivatta. Uudelleenliittyminen onnistuu joko vanhaa linkkiä käyttämällä, mikäli se vielä löytyy selaimen osoiteriviltä. Vaihtoehtoisesti liittyminen onnistuu myös samalla nimimerkillä, jota on viimeksi käyttänyt. Palvelimen on mahdollista antaa ohjain, jonka yhteyden huomattiin katkenneen, takaisin alkuperäistä nimimerkkiä esittävälle. Tällöin palvelimella ei ole tarvetta pitää erikseen kirjaa edes ohjainta käyttäneistä IP-osoitteista, vaan käyttäjän on mahdollista jopa vaihtaa pelaamiseen käyttämäänsä selainta tai mobiililaitetta kesken pelin.

Palvelinohjelma oli mahdollista asentaa saataville julkisen IP-osoitteen ja domainin taakse ilmaiseen Heroku-palveluun. Heroku tarjoaa palvelualustan kontitetuillle (containerized) sovelluksille [heroku.com/what, verkkosivusto, 2018]. Lyhyesti sanottuna kontittaminen on tapa paketoida ja virtualisoida sovellus.

5 Tekoälyt ja mahjong

5.1 Yleistä tekoälyistä

Tekoälyistä puhuttaessa tarkoitetaan yleensä tietyn ongelman ratkaisuun soveltuvaa tietokoneohjelmaa. Tällaista sovellusta kutsutaan suppeaksi tai ”heikoksi tekoälyksi”, jonka vastakohtana olisi ajatteleva ja tietoisuuden omaava ”vahva tekoäly”. [Haikonen 2017.]

Mahjongpeliä pelaavan tekoälyn tapauksessa kyse on myös heikosta tekoälystä. Tavoitteena on luoda ohjelma, joka osaa tehdä pelin sääntöjen mukaisia siirtoja. Lisäksi pyritään siihen, että ohjelma myös tavoittelisi valitsemillaan siirroilla voittoa ja siten kykenisi antamaan ihmispelaajalle haastetta.

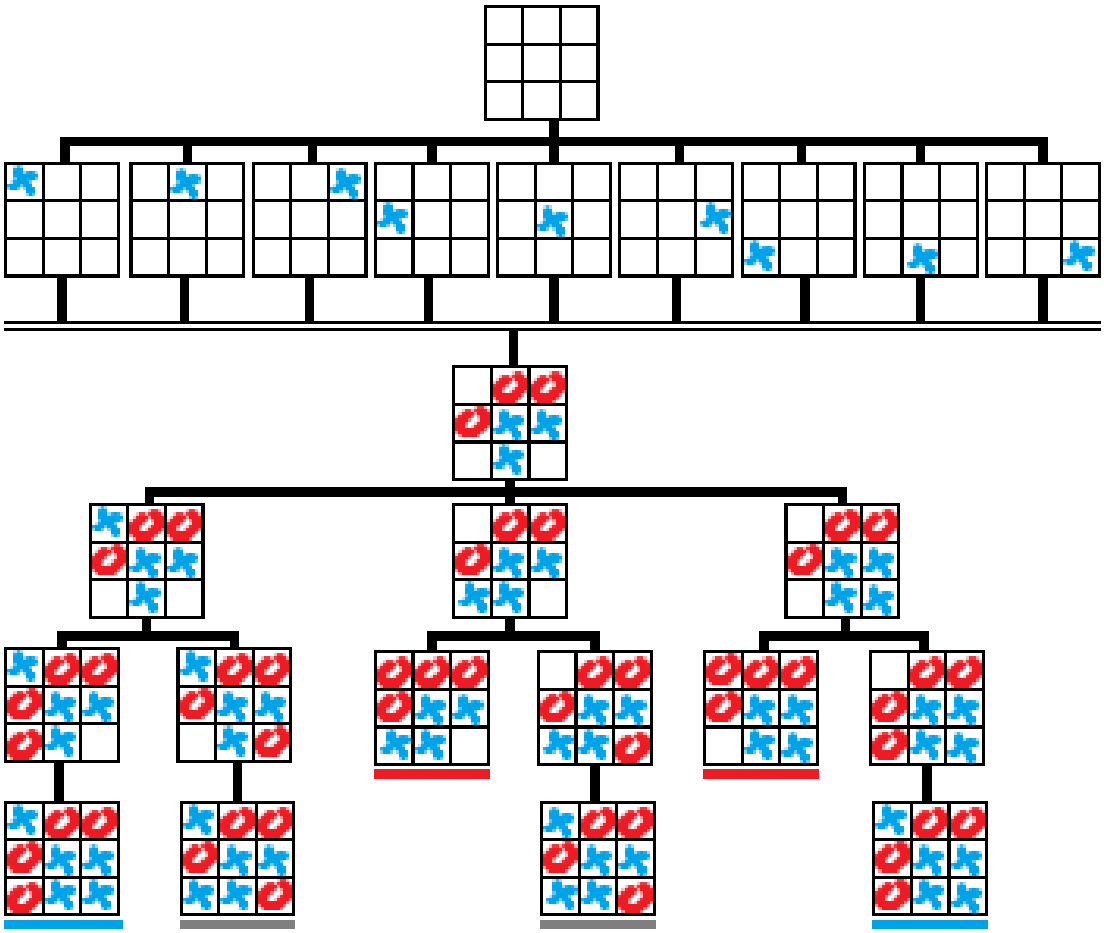
Heikkoja tekoälyjä on kehitetty jo usean vuosikymmenen ajan. Vuonna 1995 amerikkalaisen yliopiston kehittämä CHINOOK oli ensimmäinen tekoäly, jolle myönnettiin maailmanmestarin titteli [Schaffer ym. 1996]. Tekoäly pelasi tammea (checkers) tasavertaisesti sen aikaisten huippupelaajien kanssa. Googlen kehittämä AlphaGo taas voitti monimutkaisemmassa go-pelissä maailman parhaana pidetyn pelaajan turnauksessa neljällä pelillä viidestä vuonna 2016 [Silver ym. 2017].

Tammea pelaava tekoäly CHINOOK koostui pelitiloja läpikäyvistä hakualgoritmeista, pelitilanteen arvioivasta evaluointifunktiosta sekä ennakkoon lasketuista pelin alku- ja lopputilanteet arvioivista tietokannoista. Go-peliä pelaava AlphaGo taas sisälsi pelitilanteita läpikäyvän Monte Carlo -algoritmin sekä useimmiten pelattuja siirtoja palauttavan ihmispelaajien pelihistorialla esikoulutetun neuroverkon.

5.2 Tekoälyjen soveltuvuus mahjongiin

Peliä pelaavan tekoälyn voidaan kuvitella olevan näkymä, jota käyttää ihmisen sijaan tietokoneohjelma. Tekoälyllä ei tarvitse olla suoraa pääsyä varsinaiseen pelin malliin, vaan sen on mahdollista tehdä päätökset piiloinformaatiota sisältävässä pelissä saman tiedon varassa, joka ihmispelaajallakin olisi.

Tekoälyalgoritmit, jotka on suunniteltu vuoropohjaisille peleille, hyödyntävät keskeisessä roolissa pelipuuta. Täydellisessä pelipuussa juurisolmuna olisi pelin alkutila, sen lapsisolmuina seuraavat mahdolliset tilat, joiden lapsisolmuina taas niistä seuraavat tilat. Tällaisen täydellisen pelipuun solmut päätyisivät lopulta pelin päättävään tilanteeseen. Se voi olla pelaajan voitto, tai pelin päättyminen muulla sääntöjen mukaisella tavalla. Esimerkkinä kuviossa 15 on kuvattu pelipuu ristinolla-pelistä.



Kuvio 15. Pelipuun visualisointi, jossa juurisolmuna on tyhjä ristinolla-pelin pelialue. Lisäksi kuva kattaa yhden satunnaisesti valitun haaran solmut pelin päättymiseen asti. Päättyneiden pelitilanteiden solmut on alleviivattu voittaneen pelaajan värillä; sininen on risti, punainen on nolla ja harmaa tarkoittaa tasapeliä.

Esimerkkinä ristinolla on helppo hahmottaa, koska kaikki pelin tilat on mahdollista laskea auki. Mahjongissa pelaajalla kuitenkin voi olla omalla vuorolla jopa neljätoista eri siirtoa, eli kädessä olevien erilaisten tiilien määrä. Kierroksen alkaessa nostettavana on 69 tiiltä, eli jokainen neljästä pelaajasta voi saada 17 vuoroa. Yhden pelaajan osalta erilaisia päätöksiä voi siis olla noin $14^{17} \approx 3.05 * 10^{19}$ kappaletta. Todellisuudessa määrä on kuitenkin pienempi. Ihmispelaajien välisten pelitallenteiden [Tenhou.net. 2016] analysoinnin perusteella voittaja löytyy keskimäärin 11,6 vuoron aikana. Otteen 33 720 jaosta 4 698 eli noin 14 prosenttia päättyi nostojen loppumiseen. Pelaajalla ei myöskään yleensä ole kädessä neljäätoista erilaista tiiltä, koska parien muodostuminen on todennäköistä.

Päätökseen vaikuttaa kädessä olevien tiilien lisäksi myös muiden pelaajien poisheitetyt tiilet. Niiden perusteella pelaajan on mahdollista nähdä, onko tiettyä tiiltä vielä jäljellä.

Tiilet, joita ei näy, voivat olla joko nostettavissa muurissa tai muiden pelaajien kädessä. Jos jokainen pelaaja heittää kierroksen aikaan pois yhden tiilen 34 mahdollisen eri tiilen joukosta, peli voi saada minkä tahansa 34^4 = 1 336 336 uudesta tilasta. Mikäli jako etenee pisteeseen, jossa jokainen pelaaja on heittänyt pois 17 tiiltä, voi loppuun pelattu jako, ottaen huomioon poisheitettyjen tiilten järjestyksen, teoriassa olla mikä tahansa $(34^4)^{17} \approx 1{,}38 * 10^{104}$ tilasta.

Mahjongin kompleksisuus on erittäin korkea, eikä kaikkia tiloja kattavaa pelipuuta voi nykytietokoneella laskea. Tekoälyjen toteutuksessa päätettiin kokeilla satunnaisiin siirtoihin perustuva Monte Carlo -algoritmia sekä hyvin matalaa puuhakua. Tekoälyn ei myöskään määritelty ottavan huomioon muiden pelaajien poisheittämien tiilten järjestystä, koska sille ei uskottu olevan merkitystä toimivuuden kannalta.

5.3 Monte Carlo -algoritmi

Monta Carlo -algoritmin tavoitteena on generoida täysin satunnaisia pelitilanteita, jotta nähdään, mikä lopputulos olisi todennäköisin. Hyvä puoli on, ettei pelin sääntöjä tai strategioita tarvitse tietää, vaan mikä tahansa sääntöjen mukainen siirto voidaan valita. Mitä enemmän tilanteita lasketaan, sitä luotettavampia saadut todennäköisyydet ovat.

Mahjongiin Monte Carlo -algoritmi istuu luontevasti. Pelissä on sekoitettu pakka, josta nostetaan ennalta tuntematon tiili. Myöskään tiiltä pois heitettäessä ei voida olla varmoja mikä johtaisi parhaaseen lopputulokseen. Satunnaisuuden vuoksi vaikuttaa siltä, että Monte Caro -algoritmillä voisi hyvinkin saada aikaan ihmismäisesti pelaavan tekoälyn.

5.4 Puuhaku-algoritmi

Toinen lähestymistapa oli tehdä täydellisempi puuhaku, jossa käden jokainen tiili vaihdetaan vuorollaan jokaiseen pakassa mahdollisesti vielä olevaan tiileen. Jokaista tiiltä on pakassa aluksi neljä kappaletta. Todennäköisyys nostaa tiili T_x, kun näkyvillä on samasta tiilestä T_n kappaletta, ja kaiken kaikkiaan pelissä on käännetty näkyviin N tiiltä, on kaavan

$$T_x = \frac{4 - T_n}{136 - N}$$

mukainen. Todennäköisyys otetaan huomioon yhdessä sen kanssa, kuinka hyväksi saatu käsi on evaluoitu. Pelipuun haaroista sellaiset, joissa päästään edellistä tilannetta paremman arvion saaneeseen käteen tallennetaan listaan. Lista järjestetään lopuksi sen mukaan, kuinka todennäköisesti tilanne saavutetaan, jonka jälkeen listalta valitaan todennäköisin. Jälkimmäisessä toteutuksessa on mukana paljon enemmän vain mahjongille ominaisia toimintoja kuin aiemmin esitellyssä Monte Carlo toteutuksessa.

5.5 Tekoälyjen vertailu

Toteutuksia on mahdollista vertailla peluuttamalla niitä keskenään. Suoriutumista voi arvioida myös esimerkiksi sillä, kuinka usein tekoäly saavuttaa valmiin käden. On myös mahdollista vertailla tekoälyn tekemiä päätöksiä ihmispelaajien päätöksiin valmiiden pelitallenteiden avulla.

Monte Carlo -toteutuksessa on mahdollista antaa tekoälylle useampia parametreja. Parametrit määrittävät, kuinka monella erilaisella satunnaisesti sekoitetulla pakalla tilanteita simuloidaan, kuinka monta satunnaista tiiltä heitetään pois ja kuinka monta vuoroa eteenpäin tilanteita simuloidaan.

Parametrien arvojen kasvatus lisää tekoälyn päätöksentekoon kuluvaa aikaa. Sopivan suoritusajan ja suorituskyvyn löytämiseksi tekoälytoteutusta ajettiin kevennetyssä pelissä, jossa ei ollut muita pelaajia, ja tarkoituksena on saada valmis käsi 13 tiilestä siten, että oma nosto tai toisen pelaajan pois heittämä tiili johtaisi voittoon. Mittausta varten simuloinnissa sallittiin yhden pelaajan nostaa enimmillään 35 tiiltä per jako, joka on noin kaksi kertaa enemmän kuin suoraan muurista tehtäviä nostoja olisi oikeassa pelissä. Simulointia ajettiin useammalla eri parametriyhdistelmillä 50 jakoa, ja lopputulokset ovat nähtävissä kuvion 16 taulukosta.

AI	Tenpai rate (%)	Average Draws	Average Draws (win)	Time spent per state (ms)
Tree Search d1	72.00%	11.8	11.7	22.29
Tree Search d2	72.00%	21	20.3	203.02
Tree Search d3	70.00%	30	24.3	1491.82
MC 20/20/1	64.00%	26.5	22.4	45.16
MC 20/40/1	56.00%	33.1	18	41.62
MC 40/20/1	72.00%	30.5	27	88.82
MC 40/40/1	82.00%	24.5	13.7	84.62
MC 20/20/3	86.00%	23.1	23.1	113.52
MC 20/40/3	84.00%	22.6	22.5	128.55
MC 40/20/3	78.00%	32	22.5	241.5
MC 40/40/3	86.00%	22.7	22.4	247.56

Kuvio 16. Mitatut vuoromäärät ja ajat, kuinka hyvin tekoälytoteutukset pääsevät voittavaan tilanteeseen eri parametreilla.

Tenhou-sivustolta ladattuja ihmispelaajien välisiä XML-muotoisia pelitallenteita parsimalla oli mahdollista tallentaa peleistä tilanne, jossa informaationa ovat pöydässä näkyvät tiilet, pelaajan käsi ja poisheitetty tiili. Informaatio syötettiin ohjelmalla simuloimaan käynnissä olevaa peliä, ja kutsuttiin tekoälyltä minkä siirron se tekisi. Tätä palautetta verrattiin ihmispelaajan päätökseen, jolloin saatiin lista eriävistä ja yhtenevistä siirroista.

Otokseen poimittiin 100 peliä, joista saatiin 26 800 tilannetta poisheitetyn tiilen valintaan.

Kuviossa 17 on näkyvissä eri asetusten tuottamat tulokset ja siirron kesto taulukkona. Asetukset Monte Carlo (MC) -tekoälyllä ovat järjestyksessä simuloitujen sekoitettujen pakkojen määrä, satunnaisten poistojen määrä sekä pelattava syvyys.

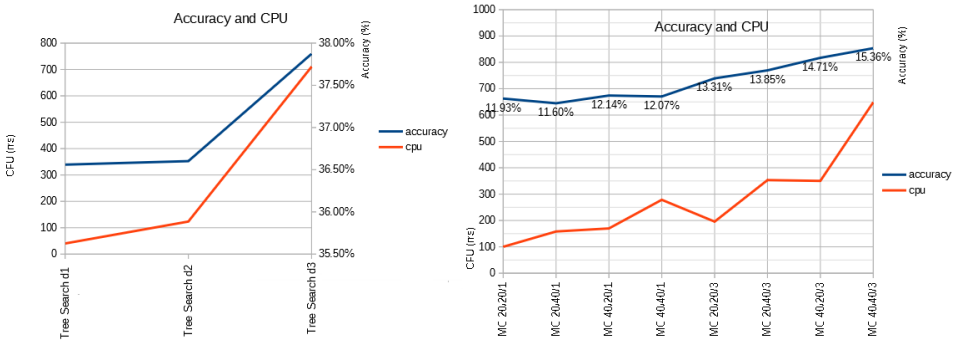
AI	Total States	Agreed States	Agreed States (%)	Time spent per state (ms)
Tree Search d1	26,805	9,800	36.56%	40.338
Tree Search d2	26,805	9,811	36.60%	123.498
Tree Search d3	26,805	10,152	37.87%	710.604
MC 20/20/1	26,799	3,196	11.93%	100.086
MC 20/40/1	26,802	3,110	11.60%	158.281
MC 40/20/1	26,806	3,253	12.14%	169.774
MC 40/40/1	26,805	3,235	12.07%	278.484
MC 20/20/3	26,805	3,567	13.31%	195.719
MC 20/40/3	26,805	3,713	13.85%	353.253
MC 40/20/3	26,806	3,943	14.71%	350.118
MC 40/40/3	26,803	4,118	15.36%	648.314

Kuvio 17. Tekoälysiirtojen vertautuminen ihmispelaajien tekemiin eri asetuksilla.

Tekoälyjen tekemiä päätöksiä ihmispelaajien siirtoihin verrattaessa näyttää siltä, että Monte Carlo -toteutus on hyvin harvoin samaa mieltä. Asetuksista riippuen samaan siirtoon päädytään vain noin 11 - 14 prosenttia ajasta. Käytännössä yhtäläisyys on niin pieni, että siirto vaikuttaa lähes satunnaiselta.

Puuhakuun perustuva tekoäly tekee ihmispelaajaa vastaavan päätöksen yli kolmasosan ajasta. Asetuksista riippuen tarkkuus on 36 ja 38 prosentin välillä. Asetuksena puuhaussa on perättäisten poistojen määrä eli pelin syvyys.

On huomattavaa, että kaikkien arvojen kasvatus näyttäisi johtavan pieneen todennäköisyyden paranemiseen, mutta myös siirtojen kesto pitenee. Suorituskyvyn ja tarkkuuden suhdetta on havainnollistettu graafisesti kuviossa 18.



Kuvio 18. Parametrien ja suoritinkäytön riippuvuus graafisena käyränä. Vasemmalla puolella on puuhaku ja oikealla Monte Carlo.

Yksittäisiä tapauksia tarkastelemalla on mahdollista huomata myös, että tekoäly on joidenkin pelaajien kanssa enemmän samaa mieltä kuin muiden. 100 pelitallenteessa on mahdollisesti jopa 400 eri pelaajaa, joilla voi olla omat tottumuksensa ja strategiansa. Joissakin tapauksissa huomataan pelaajan heittävän pois tiili, jota tekoäly on ehdottanut vuoroa tai kahta aiemmin poisheitettäväksi.

6 Parannuskohteita ja huomioita

Mahjongpelistä tuli loppujen lopuksi toimiva kokonaisuus, jota voi pelata internetin välityksellä muita ihmispelaajia sekä tekoälyjä vastaan.

Pisteiden lasku on kattava, mutta ei käsittele kaikkia erikoistapauksia. Pelissä on sääntöjen mukaan tilanne, jossa pelaajan on mahdollista voittaa, jos toinen pelaaja lisää itse nostamansa tiilen aiemmin varastamiinsa kolmosiin tehden niistä neloset. Mikäli lisätty tiili on voittava tiili jonkun pelaajan käteen, hän voi julistaa voiton aivan kuten tiiltä varastaessa. Ominaisuutta ei alkuun toteutettu sen poikkeavan mekaniikan vuoksi, mutta se olisi mahdollista lisätä peliin omaksi tilakseen.

Pelin toimintojen, kuten pisteiden laskun, optimointi ja jatkokehitys saattaa aiheuttaa ennalta-arvaamattomia ongelmia pelin toimintaan. Jotta muutoksia olisi turvallisempi toteuttaa, tulisi regressiotestausta varten toteuttaa testitapauksia kaikille oleellisille osille

pelimoottorissa. Tällöin olisi mahdollista varmistaa, ettei peli lakkaa toimimasta muutoksien toteuttamisen jälkeen.

Toteutetut tekoälyt eivät yleensä kykene voittamaan ihmispelaajaa kuin sattumalta. Niiltä puuttuu kyky päätellä, mikä tiili poisheitettynä voisi mahdollisesti johtaa toisen pelaajan voittoon, joten ne eivät osaa vältellä häviämistä. Tekoälyt eivät myöskään tunnu saavuttavan valmista kättä yhtä usein kuin ihmispelaaja, mutta se luultavasti johtuu evaluointialgoritmien toteutuksesta. Voi olla, että liian haastavaa tekoälyä vastaan pelaaminen olisi turhauttavaa, mutta turhauttavaa on myös haasteen puuttuminen.

Lähteet

European Mahjong Association. 2016. Riichi Competition Rules (2016 Edition). Verkkodokumentti. <http://mahjong-europe.org/portal/images/docs/Riichi-rules-2016-EN.pdf>. Luettu 3.3.2018.

Haikonen, Pentti. 2017. Tietoisuus, tekoäly ja robotit.

Mahjong Finland ry. Verkkosivusto. <http://www.mahjongfinland.fi/saannot/>. Luettu 4.3.2018.

Millington, Ian. 2006. Artificial Intelligence for Games.

Peltomäki Juha. 2001. JavaScript.

Rintala, Matti – Jokinen Jyke. 2003. Olioiden ohjelmointi C++:lla.

Schaffer, Jonathan – Lake, Rober – Lu, Paul – Bryant, Martin. 1996. AI Magazine Vol. 17.

Silver, David ym. 2017. Nature Vol 550: Mastering the game of Go without human knowledge.

Sinkkonen, Irmeli – Kuoppala, Hannu – Parkkinen, Jarmo – Vastamäki, Raino. 2006. Käytettävyyden psykologia.

Socket.io. Verkkosivusto. <https://socket.io/docs/>. Luettu 2.5.2019.