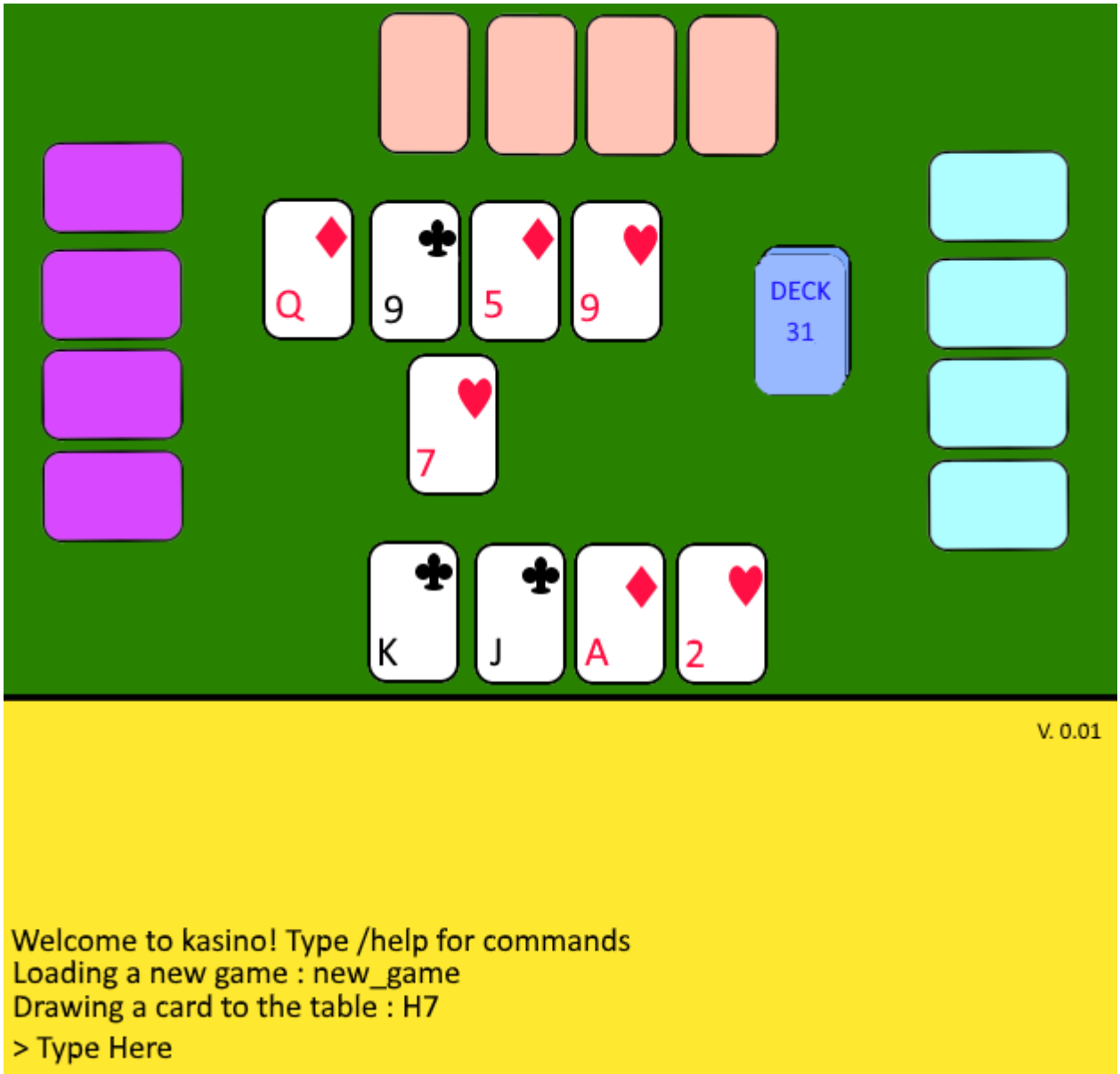
Tekninen Suunnitelma

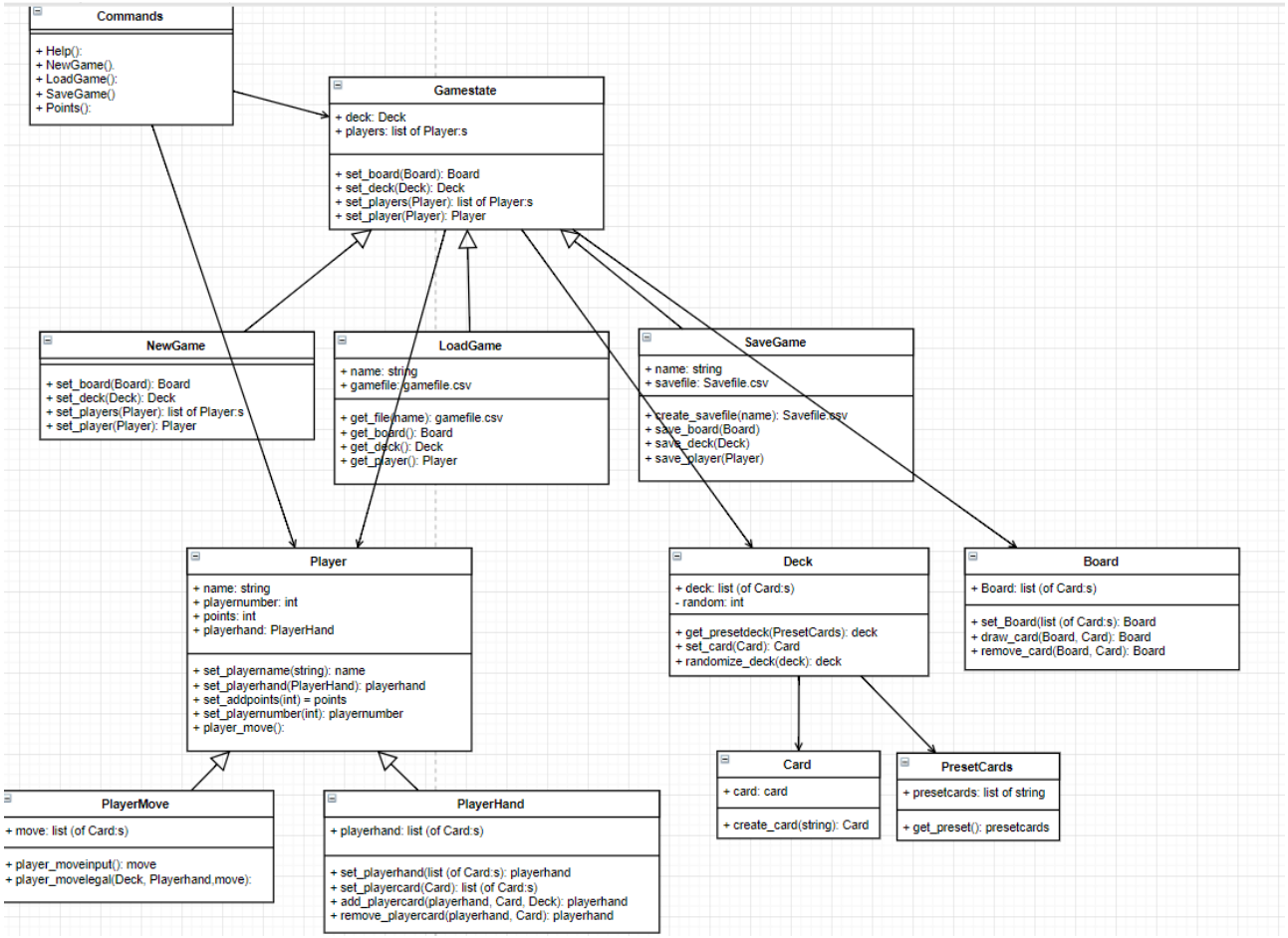
Lari Haapaniemi

651624

Kasino



Ohjelman rakennesuunnitelma



Ohjelman UML mallin hahmotelma. Kuvaan ei ole piirretty läheskään kaikkia yhteyksiä, koska kuvan tarkoitus on lähinnä mahdollistaa ohjelman rakenteen hahmottaminen. Tämän lisäksi mallinnuksesta riippuu lukuisia käskyjä (klassiset return käskyt) ja muutama muuttuja (esimerkiksi gamestatesta kuka pelaajista on jakaja)

Tässä tiivis lista siitä, mitä kukin tiedosto tekee.

-PresertCards: Noutaa arvottavat kortit valmiista listasta

-Deck: Arpoo deckin

-Board: Luo pöydässä olevat kortit, ja mahdollistaa niiden käsittelyn

-Card: Muuntaa kortit listaksi

-Player: Setuppaa pelaajan kerrallaan, pitää lukua pisteistä (sekä lopullisista ja kierroksella saaduista pöytäkorttien muodossa)

-PlayerHand: Luo käden pelaajalle

-PlayerMove: Ottaa vastaan pelaajan peliliikkeet, tarkistaa niiden laillisuuden

-Commands: Mahdollistaa eri käskyjen kutsumisen (Help, draw tms, load_game)

Gamestate: Toimii ylätiedostona eri tavoille aloittaa peli, ja pitää yllä pelin luomisen jälkeen tarvittavia arvoja (esimerkiksi tarvittavan määrän kortteja pelaajilla ja pelilaudalla, tämä tapahtuu luonnollisesti alaluokkien kautta. En lisännyt tätä UML mallinnukseen, koska en ole vielä 100% varma millä tavalla toteutan. Luultavasti päivitys tapahtuu jokaisen pelaajan vuoron jälkeen)

-NewGame: Luo uuden pelin, deckit, pelaajat

-LoadGame: Lataa uuden pelin, deckit, pelaajat

-SaveGame: Tallentaa pelin, deckit, pelaajat

Ei lisätä UML:

-GUI: Vastaa ”pelilaudan” luomisesta

-CardGraphics: Vastaa korttien luomisesta

-TextGraphcis: Vastaa tekstien luomisesta

-TestGame: Vastaa pelin testaamisesta

-Main: Vastaa pelin pyörittämisestä

-Ai: Pohjaversio AI:lle, kyseiseen tiedostoon tulee varmasti ylimääräisiä alaluokkia, mutta tällä hetkellä hyvin aukinainen osa ohjelmaa

Käyttötapakuvaus

Yleissuunnitelmaan liitetty kuva on klassinen alkutilanne pelistä, jossa on aloitettu new_gamella uusi peli. Aloittaessa uuden pelin pelaajalta kysytään muutama kysymys, joita tarvitaan pelin tekoon. Ensimmäisenä kysytään pelaajien määrä, jonka jälkeen käydään läpi yksi kerrallaan jokaisen pelaajan nimi, ja mahdollisesti onko pelaaja ihminen vai tietokone. Jokaisen pelaajan kohdalla NewGame kutsuu funktiota set_players, jonka kautta se suorittaa set_player, ja lisää tämä palauttaman arvon pelaajalistaan. Samalla jokaiselle pelaajalle annetaan pelaajanumero kutsumisen helpottamiseksi. Tämän jälkeen NewGame kutsuu funktiota set_deck, joka hakee Deck oliosta pakan jonka se arpoo Presetcardeista, ja asettaa jokaiselle kortille kolmen muuttujan listan, johon sisältyy sen maa, arvo kädessä ja arvo pöydällä. Ohjelma tämän jälkeen lisää jokaiselle pelaajalle luodusta deckistä 4 korttia, ja asettaa ne pelaajalle set_playerhandin kautta Player olion alaluokassa PlayerHand. Seuraavana on neljän kortin asettaminen pöydälle set_boardin kautta, joka kutsuu Board oliota. Luonnollisesti QtLiittymä, joka toteutetaan projektin loppuvaiheessa luo jokaisen kortin asetettaessa pöydälle uuden GraphicsItemin, ja GUI vastaa tekstikonsolin ja muun pois lukien korttien mallinnuksesta.

Pelin alkaessa pelaaja voi valita joko pelaavansa kortin, tai nostavansa sen. Jos pelaaja päättää pelata kortin, hän syöttää ensin kädestä pelattavan kortin, ja sen jälkeen yksitellen pöydässä olevat kortit, jotka hän haluaisi itselleen. Jos liike on laitoin, pelikonsoli ilmoittaa asiasta, ja pyytää tekemään uuden liikkeen. Nostettaessa kortti ohjelma kutsuu Boardia, joka lisää pöydälle uuden kortin, ja siirtää vuoron seuraavalle. Vuoron siirto tapahtuu silloin, kun pelaaja tai tietokone suorittaa onnistuneen siirron. Tämä on siis klassinen ohjelman kulku. Uuden kierroksen alkaessa pelilauta suorittaa käskyt uudestaan new_roundilla, jota en ole UML:llään mallintanut. Kyseinen komento sijaitsee Gamestatessa. Tällöin pakka, pöytä ja pelaajien kädet arvotaan uudestaan. Erikoistilanteet kuten mökki tms. on otettu huomioon kortteja pelatessa, ja ne lisäävät pisteitä pelaajalle sen mukaan, kun niitä ilmenee.

Algoritmit

Tärkein algoritmi projektin kannalta on kortin laillisuuden tarkistamiseen tarkoitettu algoritmi, jossa pelaajan tarvitsee vain ilmoittaa haluamansa kortti käytettäväksi, ja ohjelman pitää pystyä tulkitsemaan, onko liike laillinen vai ei. Olen saanut jo kyseisen algoritmin toimimaan ihmispelaajan kohdalla, mutta tietokoneversio osaa laskea vain yhden ja kahden kortin yhdistelmiä.

Ihmisversion tarkistus:

Ihmisversion tarkistusalgoritmi on suhteellisen yksinkertainen, sillä suurin työ tehdään luodessa jokainen kortti. Ihminen syöttää tekstikenttään ensin kortin, jonka haluaa pelata kädestään, jonka jälkeen häneltä kysytään yksi kerrallaan kortit, jotka hän haluaa ottaa pöydältä. Kyseiset kortit syötetään yksi kerrallaan. Kyseiset inputit käännetään korteiksi, ja luonnollisesti jos kortit ovat tuntemattomia, kyseisestä asiasta ilmoitetaan tässä vaiheessa. Jos kortit olivat oikeanlaisia, funktio palauttaa listan, jossa ensimmäisenä alkiona on kädestä syötetty kortti, ja toinen alkio on lista, jossa on pöydästä saatavat kortit. Muussa tapauksessa funktio palauttaan arvon False, joka tarkoittaa että prosessi epäonnistui.

Tämän jälkeen kyseinen lista viedään funktiolle, joka tarkistaa kyseisen peliliikkeen laillisuuden. Aluksi funktiossa tarkistetaan löytyvätkö annetut kortit pöydästä ja kädestä, ja jos näin ei ole, funktio palauttaa False ja ilmoittaa virheestä. Jos kuitenkin kortit ovat olemassa, funktio ottaa ensin käsikortista ”käsiarvon”, ja siirtyy laskemaan pöydältä nostettavien korttien määrän. Tämän jälkeen ohjelma lisää total nimiseen muuttujaan pöydästä valittujen korttien ”pöytäarvot”, ja vertaa onko käsiarvo yhtä suuri kuin total. Jos näin on, liike on laillinen, ja toiminto suoritetaan, muussa tapauksessa funktio aiheuttaa jälleen errorin, ja palauttaa Falsen.

Jos kyseisessä toimenpiteessä tapahtuu error, funktio antaa pelaajan antaa syötteensä uudestaan. Tällä hetkellä korttien olemassa olo tarkistetaan vasta kaikkien syötteiden jälkeen, mutta luultavasti vaihdan ohjelman toimimaan niin, että se tarkistaa korttien olemassaolon heti syötteen jälkeen, jotta ohjelman toiminta olisi sulavampi.

Tietokoneversion tarkistus:

Tietokoneen tarkistustoiminto on tällä hetkellä vielä aika WIP vaiheessa, mutta tämän hetkisen konseptin mukaan toiminto jakautuu kahteen vaiheeseen, tarkistukseen ja tämän jälkeiseen ”Min-Max” tyyppiseen toimintoon. Tarkoituksena on siis käydä jokainen tietokoneen kortti läpi, ja löytää kyseiselle kortille kaikki mahdolliset yhdistelmät pelilaudalta (kuten edellä mainitsin, tällä hetkellä kone löytää yhden ja kahden kortin yhdistelmiä). Käytännössä tämä tapahtuu niin, että aluksi otetaan ensimmäinen kortti kädestä, ja verrataan sitä ensimmäiseen korttiin pöydältä. Jos pöytäkortin pöytäarvo on enemmän kuin käsikortin käsiarvo, siirrytään toiseen pöytäkorttiin. Jos ne ovat yhtä suuret, lisätään funktion alussa tehtyyn ”viablemoves” listaan samalla tavalla kuin ihmispelaajan inputeilla tehty lista, ja siirrytään seuraavaan pöytäkorttiin. Jos kuitenkin pöytäkortin arvo on vähemmän kuin käsikortin, siirretään käsikortti, pöytäkortti, pöytäkorteista tehty lista ja tyhjä ”move” lista (johon säilötään mahdolliset useamman kortin liikkeet) toiselle funktiolle,

jonka tarkoitus on käydä läpi samalla tavalla kuin aiempi funktio mahdolliset yhdistelmät kyseiselle käsikortille ja lopuille korteille pöydässä.

Jos tällainen löytyy, kortti palauttaa move listan johon uusi korttiyhdistelmä on lisätty. Jos kuitenkaan tälläista yhdistelmää ei löydetty, funktio palauttaa tyhjän move listan, josta aiempi funktio tietää siirtyä seuraavaan pöytäkorttiin. Kun kaikki pöytäkortit on käyty läpi, siirrytään seuraavaan käsikorttiin.

Tarkoitus olisi tehdä ns. extendedlegalmoves funktioista (jota ensimmäinen funktio kutsuu kun on mahdollista löytää enemmän kuin yhden kortin yhdistelmä) looppaava kokonaisuus, jotta enemmän kuin kahden kortin yhdistelmät olisivat mahdollisia, mutta tämä ei valitettavasti tähän mennessä ole onnistunut. Olen kuitenkin varma, että tämä on mahdollista.

Kun jokainen mahdollinen korttiyhdistelmä on käyty läpi, ja lailliset löydetty, funktio palauttaa listan kaikista mahdollisista peliliikkeistä joita tietokone voi tehdä, ja vie ne min-max algoritmille. Min-max algoritmi käy läpi jokaisen yhdistelmän, ja pisteyttää ne niistä saatavien pisteiden mukaan. Jos kyseisestä liikkeestä ei saa suoraan pisteitä, se etsii yhdistelmän joka ottaa eniten kortteja pelaajalle pöydästä. Kyseinen konsepti on itselleni tuttu, koska teimme samalla periaatteella tammea pelaavan tekoälyn sähköpaja kurssilla.

Tietorakenteet

Kuten yleiskuvauksessa kerroin, tiedon varastoimiseen gamestatessa käytetään CSV tekstitiedostoja, joiden muodossa tallennan ja lataan pelin. Mahdollisia vaihtoehtoja olisi ollut lukuisia, mutta itselleni tämä on tuttu ja helppo tapa käsitellä infoa mitä ohjelman rakentamiseen tarvitaan.

Koska ohjelmassa on paljon tietoa, jota pitää päivittää usein, päädyin käyttämään lähes kaikissa tietorakenteissa pythonin omaa listaa. Tämän tilalla olisi voinut käyttää esimerkiksi sanakirjaa, joka olisi varmasti ollut myös hyvä tapa säilöä tietoa korteista, mutta en näe syytä vaihtaa listoista, koska ne ajavat tarkoituksen hyvin. Lähes kaikilla olioiden muuttujilla joissa käytän listaa pariton on yksittäinen kortti, ja parillinen pöytäkortti. Tästä poikkeuksena listat joilla kuvaan pöytää, pakkaa ja käsiä, joissa jokainen alkio eli lista[i] merkitsee yhtä korttia. Myöskin listoja käytettäessä len on mahtava tapa pitää kyseiset rakenteet oikean kokoisina.

Aikataulu

Aikataulutusta on vaikea hahmottaa, koska en ole ennen rakentanut näin laajaa projektia, ja kaikki tähän mennessä tekemäni työ on ollut hajanaista testailua yhteen tiedostoon, jotta hahmottaisin mitkä olisivat parhaita ratkaisuja. Tämän ansiosta jo osa tärkeistä ohjelman osista on valmiina, mutta niiden optimisointi vaatii aikaa. Aloitan luultavasti Deck oliosta, koska se on lähimpänä valmista kokonaisuutta. Veikkaisin, että tähän kuluu n. 5 h saada se toimimaan juuri halutulla tavalla. Deck osioon siis kuuluu mielessäni myös Card olio, ja PresetCards.

Tämän jälkeen olisi varmaan fiksua tehdä alkeellinen käyttöliittymä ohjelmalle, jotta loppuvaiheessa ei tarvitsisi alkaa vaihtamaan koodia saadakseen se yhteensopivaksi PyQt:n kanssa. Tällöin seuraava looginen vaihe olisi GUI ja siihen liittyvien kohtien ohjelmointi (esimerkiksi korttien piirtäminen pöydälle). Veikkaisin kokemattomuuteni takia, että tähän menisi yhteensä 10 h, joka jaettaisiin pienempiin väleihin ja suurin osa ajasta menisi wikiä lukiessa.

Player ja Board olioissa menee yhteensä oletettavasti 10 h myöskin, joista varmasti suurin osa ajasta kuluu Player oliota tehdessä. Tästä PlayerMove on jo hyvässä vaiheessa, periaatteessa täysin toimiva. Tämän jälkeen ns. peruspalaset ovat koossa, ja voidaan siirtyä Gamestaten ohjelmointiin, jota oletettavasti olen hahmotellut koko tämän ajan muiden olioiden rinnalla, johtuen niiden tiiviistä yhteistyöstä. NewGamessa ei pitäisi mennä kovin kauaa, LoadGamessa oletettavasti 3 h, ja SaveGamessa saman verran, riippuen kyseisen prosessin vaativuudesta (tiedostoon tallentaminen on itselleni vielä aika tuore konsepti).

Kaikkien näiden rinnalla olen olettavasti testaillut aktiivisesti, mutta laajemmat testit hahmotellaan tässä vaiheessa, ja PyQt ottaa suurimman huomion tässä vaiheessa, jossa samalla rakennan Commands tiedostoa, joka pyörittää tekstikenttään annettavia käskyjä. Tähän yhteis taakkaan veikkaisin menevän vähintäänkin yksi viikonloppu kokonaisuudessaan, mahdollisesti useampi, eli veikkaisin 18 h olevan aika realistinen aikamääre.

Tämän mukaan n. 50 h olisi aika realistinen määrä mitä kyseisen projektin rakentamiseen kuluisi. On vaikea arvioida tarkalleen, missä kohdissa nousee ongelmia, ja mitkä sujuvat oletettua nopeammin. Tarkoitukseni on tehdä projektiin vähintäänkin pieniä muutoksia joka päivä, ja muiden kurssien työmäärien vaihdellessa projektipanokseni myös vaihtelee.

Yksikkötestaussuunnitelma

Olen tottunut tekemään testauksia ohjelmia rakennettaessa lähes kokoajan kyseisessä oliossa, eli jokaista oliota rakennettaessa testaaminen on jatkuvaa. Varsinaisen testiolion rakennan varmaan kun Gamestate alkaa muodostua kokonaisuudessaan, ja tarkistamaan että kokonaisuus toimii halutulla tavalla.

Kuten yleissuunnitelmassa totesin, on ehdottoman tärkeää, että funktiot ja oliot toimivat mahdollisimman virheettömästi heti niitä rakentaessa, koska eri luokat ovat niin riippuvaisia toisistaan. Helppoja tarkistuksia on katsoa, että pakka on joka kerta eri järjestyksessä, kädessä on aina neljä korttia, pöydällä ei koskaan alle 4 (pois lukien pakan loppuessa), se että pelikierros osaa lopettaa itsensä, ja että pisteet kirjataan oikealla tavalla oikealla pelille. Loadgamessa tarkistetaan eri tekstitiedostoilla, että oikeisiin paikkoihin tulee oikeita kortteja. PyQt:n testaus onnistuu varmaan parhaiten pelaamalle ohjelmaa, ja katsomalla mitä milloinkin tapahtuu. Myös komentojen testaaminen on tärkeää, ja niiden suorittamat funktiot käydään samalla läpi.

Kirjallisuusviitteet ja linkit

Tällä hetkellä käytössä olevia lähteitä, tai lähteitä joita saatan mahdollisesti käyttää:

https://wiki.python.org/moin/

https://wiki.python.org/moin/PyQt/Tutorials

https://docs.python.org/3/library/random.html

Tammirobotin python koodi Sähköpaja kurssilta, jossa olin mukana

PyQt 5 tutoriaalit Youtubesta