Patrik Ahvenainen / 013326292
patrik.ahvenainen@helsinki.fi
Ohjaaja: Joel Rybicki

Yleinen noppapeli

Suunnitteludokumentti

Ohjelmoinnin harjoitustyö
Tietojenkäsittelytieteen laitos
Helsingin yliopisto
Palautuspäivä: 11.04.2010

1 Ohjelman yleisrakenne

1.1 Yleistä

Peli toteutetaan python sovelluksena. Käytetty kieliversio on 2.5, jotta toteutettu sovellus olisi helposti ajettavissa tietojenkäsittelytieteen laitoksen koneilla. Pelin suunnittelun lähtökohtana on suunnitella helposti muunneltava ja laajennettava ohjelma, jossa käyttöliittymä ja pelimoottori toteutetaan erillisinä osina. Peliin tulisi siis kyetä suunnittelemaan toinen, esimerkiksi tekstipohjainen käyttöliittymä. Luokkien suunnittelussa pyritään välttämään turhaa tiedon toistoa. Toisaalta pyritään tarjoamaan monipuoliset luokat, joiden avulla kyetään toteuttamaan hyvin toisistaan eroavia pelejä. Pelin sääntöjen rajoittamattomuuden vuoksi peliin ei toteuteta kurssisuorituksen yhteydessä tekoälyä.

Pelin luokkien attribuutit toteutetaan pythonin *@property*-ominaisuuksina, jotta niiden arvojen asettamiselle voidaan tarjota turvallisempi vaihtoehto. Pythonin 2.5 versiossa tämä ei välttämättä tee koodista yhtään luontevampaa ja edellyttää erillisten get, ja set -metodien luomista ja niiden yhdistämistä muotoon

attribuutti = property(get_attribuutti, set_attribuutti),

jossa sekä get_attribuutti ja set_attribuutti on määritelty aikaisemmin. Kaikki pelin luokat periytyvät pythonin object-luokasta tai toisesta, siitä jo perityneestä luokasta.

1.2 Luokkasuunnittelua: Peli-luokka

Peli-luokka on pelin pelimoottorin tärkein luokka. Sen luomiskutsulla luodaan myös kaikki pelin käyttämät luokat, joita tarvitaan peli käynnistyessä. Peli-luokka on oikeastaan lista peliin kuuluvista muilla luokista sekä sisältää tiedon pelikierroksesta ja pelivuorosta.

Metodit

- `def __init__(self, pelaajat, asetustiedosto):`
  - Luo listan pelaajista. Lataa tämän jälkeen asetustiedostosta listan pelissä käytettävistä yhdistelmistä ja luon niiden perusteella pelaajille tulostaulukot. Luo peliin asetustiedostossa määritellyn määrän noppia.
- `def get_Pelaajalista(self):`
  `def get_Pelivuoro(self):`
  `def get_Pelikierros(self):`
  `def get_Yhdistetelmälista(self):`
  `def get_Heitto(self):`
  `def get_Noppalista(self):`
  - Palauttaa alaviivan jälkeisen attribuutin.
- `def Pelikierros_etene(self):`
  - Siirtää pelikierrosta yhdellä eteenpäin ja heittää noppaa ensimmäisen pelaajan kohdalla. Heitto-luokan ilmentymä luodaan aina kuin pelikierros tai pelivuoro etenee. Jos kyseessä on viimeinen kierros pelin loppumisesta ilmoitetaan ja pelin voittajan pisteet lisätään top-10 -listalle, jos pistemäärä on tarpeeksi suuri. Pelikierros tallennetaan Pelikierros-kokonaislukuun.

- `def Pelivuoro_etene(self):`
  - Yrittää edistää pelivuoroa siirtämällä pelivuoroa seuraavalle pelaajalle ja kasvattamalla Pelivuoro-muuttujaa. Jos kyseessä on viimeinen pelaaja, kutsutaan metodia Pelikierros_etene().

1.3 Luokkasuunnittelua: Asetukset-luokka

Peli-luokan jälkeen toiseksi tärkein luokka on varmaankin Asetukset-luokka. Tässä luokassa määritellään pelisäännöt. Peliasetukset luodaan lukemalla tiedostosta Peliasetukset. Peliasetuksiin luetaan eri yhdistelmien lisäksi oletusarvot seuraaville muuttujille: kierrosten lukumäärä, noppien maksimisilmäluku ja heittovuorojen lukumäärä yhdellä pelivuorolla. Näiden kolmen muuttujien arvoja voidaan muuttaa ennen pelin alkua (pelikierros 0).

Metodit

- `def __init__(self, tiedoston_nimi, peli):`
  - Luo asetuksiin tarvittavat muuttujat. Muuttujien arvot luetaan peli-asetuksista kutsumalla metodia set_Yhdistelmalista_tiedostosta(...).
- `def get_Kierrosten lukumaara(self):`
  `def get_Moppien_maksimisl(self):`
  `def get_Heittovuorojen_lkm(self):`
  `def get_Asetuskoodi`
  - Palauttaa alaviivan jälkeisen attribuutin.
- `def set_Yhdistelmalista_tiedostosta(self, tiedoston_nimi):`
  - Lukee Peliasetukset-tiedostosta käytettävät yhdistelmät sekä kierrosten lukumäärän, noppien maksimisilmäluvun ja heittovuorojen lukumäärän yhdellä pelivuorolla.
- `def set_Kierrosten lukumaara(self, Uusi_kierrosten_lkm):`
  `def set_Noppien_maksimisl(self, Uusi_noppien_maksimisl):`
  `def set_Heittovuorojen_lkm(self, Uusi_heittovuorojen_lkm):`
  - Muuttaa alaviivan jälkeisen muuttujan arvoa jälkimmäisen parametrin (kokonaisluku) arvoksi, jos parametri on hyväksyttävällä välillä oleva kokonaisluku.

1.4 Luokkasuunnittelua: Noppa-luokka

Noppapelissä tärkeitä tekijöitä ovat tietenkin nopat. Pelin toteutuksessa nopalla ei itse asiassa ole montaa metodia ja attribuuttia. Nopalla on tieto vain neljästä asiasta: maksimisilmäluvustaan, silmäluvustaan, edellisestä silmäluvustaan, sekä lukitsemisstatuksestaan. Edellinen silmäluku-tieto nopalla on siksi, että se mahdollistaa edellisen siirron perumiseen, jos peliin halutaan toteuttaa "Peru"-toiminto. Tämän harjoitustyön toteutuksessa sitä toimintoa ei toteuteta.

Metodit

- `def __init__(self):`
  - Pelin luomisen yhteydessä luodaan peliasetuksista luettu määrä noppia. Nopat luodaan myös muutettaessa noppien maksimisilmälukua tai noppien lukumäärää.
- `def get_Silmaluku(self):`
  `def get_Lukossa(self):`
  - Palauttaa alaviivan jälkeisen attribuutin. "Lukossa" on totuusarvo.

- `def Heita(self):`
  - `Vaihtaa noppien silmalukua ja päivittää edellisen silmäluvun, jos noppa ei ole lukittu. Lukitun nopan heittäminen ei aiheuta mitään toimintoja.`
- `def Lukitse(self):`
  - `Muuttaa nopan lukossa-arvon todeksi (true).`
- `def Avaa(self):`
  - `Muuttaa nopan lukossa-arvon epätodeksi (false).`
- `def toggle_Lukossa(self):`
  - `Muuttaa nopan lukossa-arvon todeksi (true), jos se oli epätosi (false) ja epätodeksi (false), jos se oli tosi (true).`

1.5 Luokkasuunnittelua: Heitto-luokka

Oleellisesti noppiin liittyy myös tieto kaikkien noppien silmälukujen ominaisuuksista. Yhdessä kaikkien noppien silmäluvut muodostavat heittovuoro-tiedon kanssa heiton. Heitto-luokan ilmentymä luodaan kun ensimmäinen pelaaja on heittänyt noppia. Heitto-luokka tarjoaa Yhdistelmät-luokalle jotain hyödyllisiä tietoja noppien silmäluvuista, kuten kirjaston (dict) noppien silmäluvuista, m:n nopan suorista sekä n:stä kappaleesta noppia, joilla on sama silmäluku (n samaa). Lisäksi Heitto-luokasta saadaan kaikkien noppien silmälukujen summa (sattuma).

Metodit

- `def __init__(self, peli):`
  - `Heittovuoro luodaan pelin ensimmäisen pelaajan heittäessä noppaa.`
- `def get_Heittovuoro(self):`
  `def get_Sattuma(self):`
  `def get_Silmaluvut(self):`
  `def get_n_samaa(self):`
  `def get_m_sarja(self):`
  - `Palauttaa alaviivan jälkeisen attribuutin.`
- `def Uusi_Heittovuoro(self):`
  - `Tätä metodia kutsutaan pelaajan heittäessä ensimmäisellä heitto-vuorollaan noppaa. Heiton attribuuteista Heittovuoron arvoksi asetetaan yksi ja muiden attribuuttien arvot luetaan noppien silmä-luvuista.`
- `def Heittovuoro_etene(self):`
  - `Heiton attribuuteista Heittovuoron arvoa kasvatetaan yhdellä ja muiden attribuuttien arvot luetaan noppien silmäluvuista. Jos kyseessä on viimeinen heittovuoro, ei tehdä mitään.`

1.6 Luokkasuunnittelua: Yhdistelmä-luokka

Eri yhdistelmille voidaan asettaa erilaisia perusteita siitä, miten pisteet kyseiselle yhdistelmälle lasketaan. Jos yhdistelmä ei täytä kriteerejä, tulee yhdistelmän arvoksi nolla. Yhdistelmälle voidaan asettaa rajoituksiksi noppien maksimi/minimilukumäärä sekä noppien maksimi/minimisilmäluku. Perus-yhdistelmien pisteet lasketaan yhdistelmän koodia käyttäen.

Metodit

- `def __init__(self, asetukset):`

        - Yhdistelmät luodaan peliasetusten lataamisen yhteydessä ja listataan pelin yhdistelmälistaan.
    - def get_Yhdistelmannimi(self):
        - Palauttaa Yhdistelmän nimen (string).
    - def Laske_arvo(self):
        - Palauttaa yhdistelmän laskusääntöjen mukaisen pistearvon. Pistearvo voi riippua noppien lukumäärästä, yhdistemälle määritetyistä noppien maksimi/minimilukumäärästä ja maksimi/minimisilmäluvusta sekä noppien maksimisilmäluvusta.

1.7 Luokkasuunnittelua: Pelaaja-luokka

Pelaaja-luokka on luotu laajennettavuutta silmällä pitäen. Nykyinen Pelaaja-luokka sisältää pelaajan tiedoista ainoastaan nimen sekä tiedon pelaajan tulostaulukosta.

Metodit

- def __init__(self, pelaajan_nimi, peli):
    - Pelaajat luodaan peliasetusten lataamisen jälkeen ja niille annetaan nimi sekä niille luodaan tulostaulukko.
- def get_Nimi(self):
  def get_Tulostaulukko(self):
    - Palauttaa alaviivan jälkeisen attribuutin.
- def set_Nimi(self):
    - Muuttaa pelaajan nimeä.

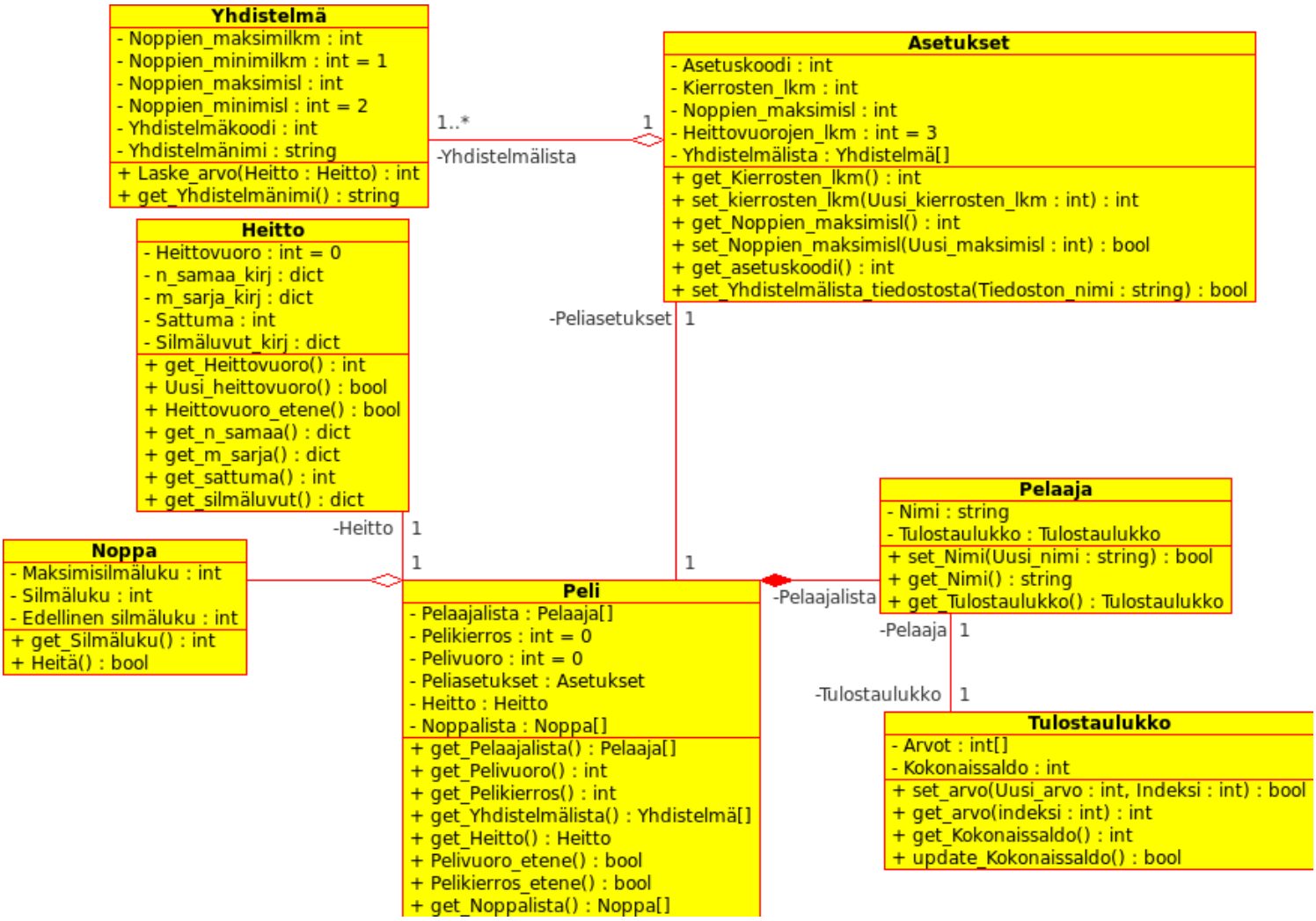
1.7 Luokkasuunnittelua: Tulostaulukko-luokka

Tulostaulukkoon kerätään pelaajien pisteet. Tulostaulukossa ei ole mitään tietoa siitä, mistä yhdistelmästä mikäkin pistemäärä tulee. Kokonaissaldo päivitetään aina muuttaessa yhtä arvoa.

Metodit

- def __init__(self, peli, pelaaja):
    - Tulostaulukko pelaajalle luodaan aina uuden pelin aloittamisen yhteydessä sekä asetuksia muuttaessa.
- def get_Arvo(self, indeksi):
  def get_Kokonaissaldo(self):
    - Palauttaa alaviivan jälkeisen attribuutin.
- def set_Arvo(self, indeksi, Uusi_arvo):
    - Indeksin mukaisen arvo-muuttujan arvoa päivitetään uudella arvolla, jos vanha arvo on nolla. Jos vanha arvo on jo olemassa, mitään ei tehdä.

2 UML-kaavio pelimoottorista



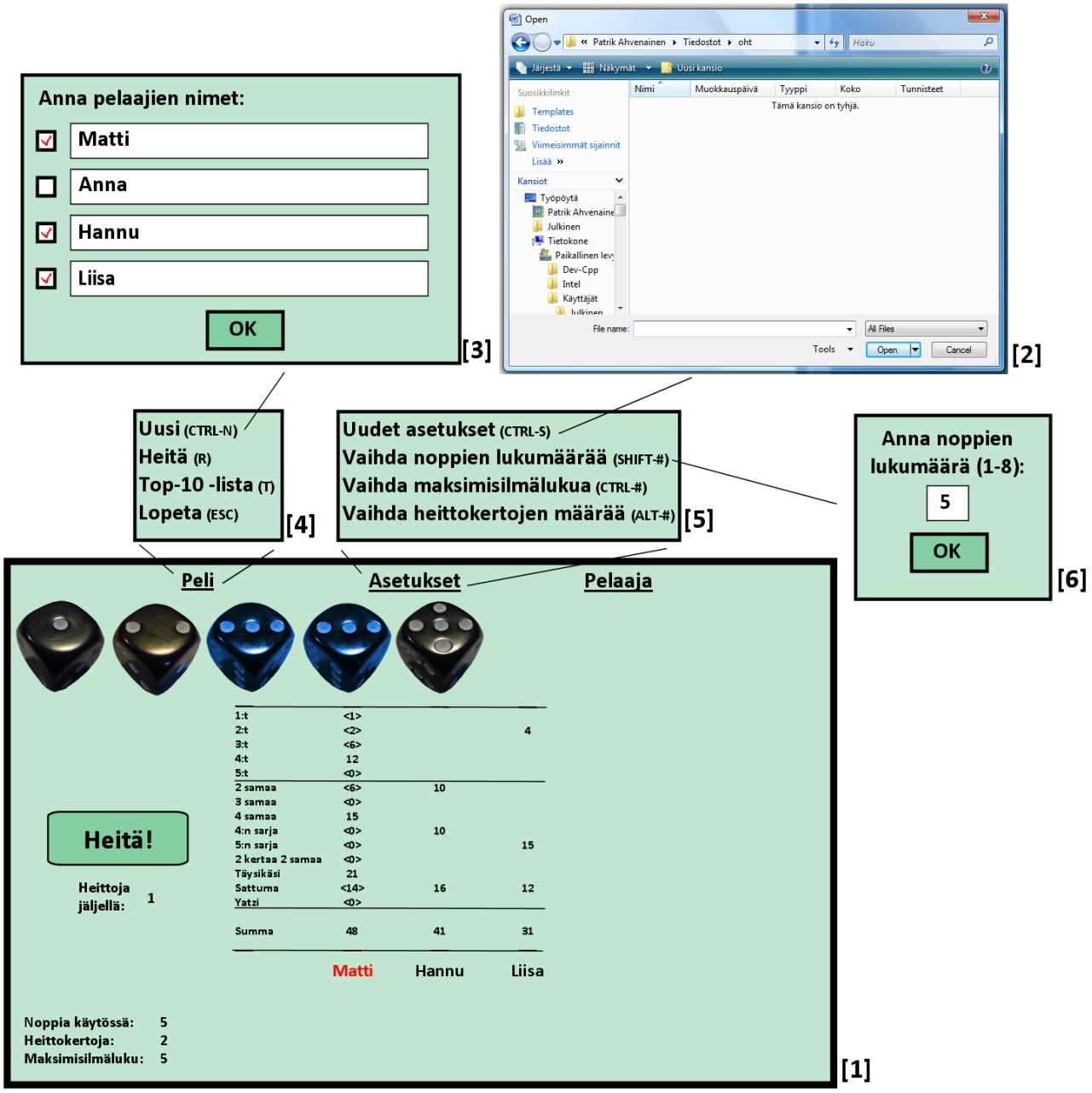
Kuva 1. UML-kaavio yleisen noppapelin peruskomponenttien keskenäisistä suhteista. Luokkien attribuutit ovat merkitty yksityisiksi (-). Käytännössä tätä rajoitusta ei tulla toteuttamaan, mutta luokkien käyttöön toteutetaan "turvallisia" metodeja, joiden avulla on tarkoitus muokata luokkien attribuutteja.

3 Käyttöliittymän suunnittelua

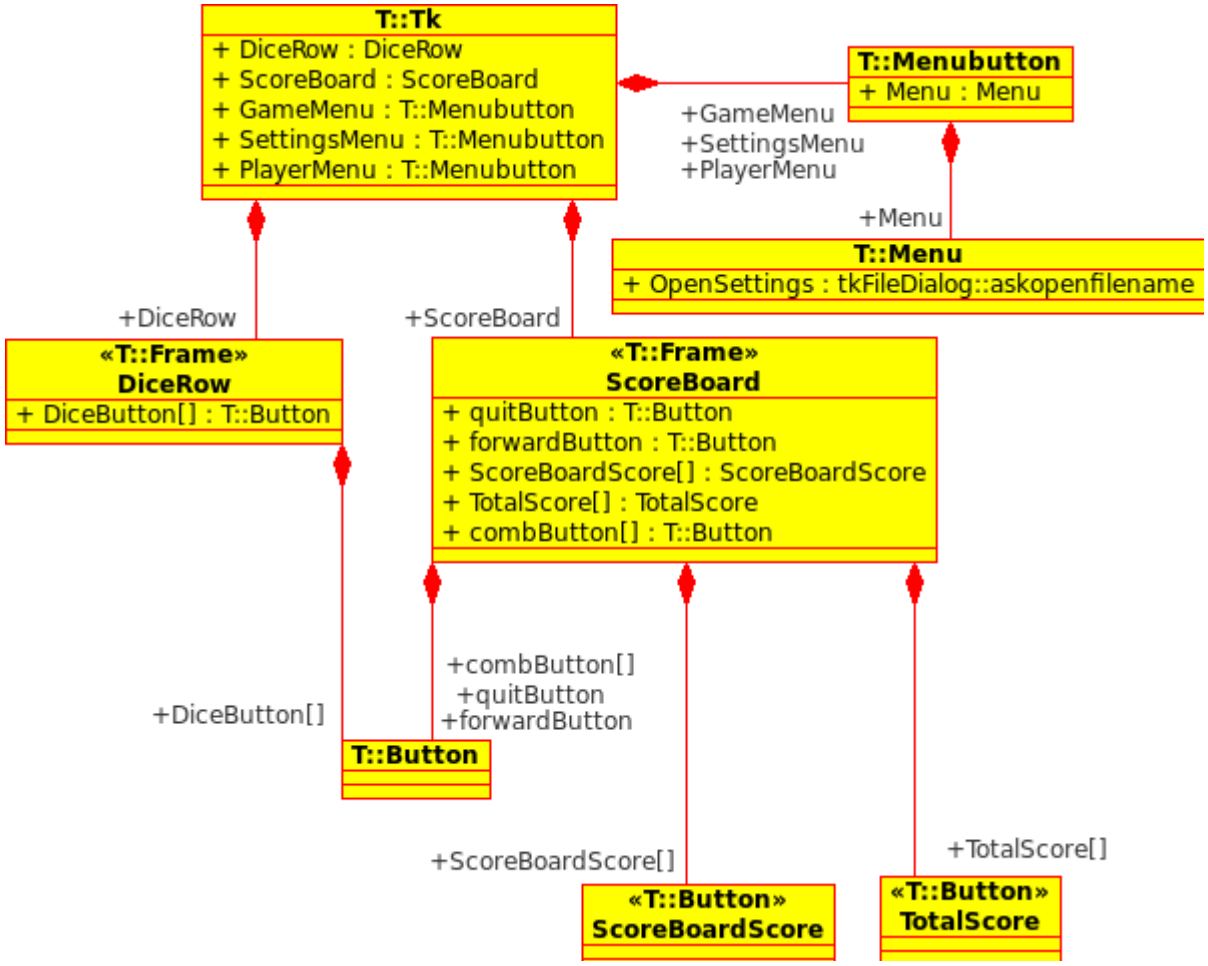
Käyttöliittymä toteutetaan pelimoottorista erillisenä komponenttina. Käyttöliittymä toteutetaan Tkinter-moduulin avulla. Käyttöliittymässä käytetään mahdollisimman paljon valmiita moduuleita, joiden toimivuus on testattu kaikissa käyttöjärjestelmissä.

Pelin päänäyttö [1] (Kuva 2) toteutetaan Frame-widgettinä. Siinä olevat nopat toteutetaan Button-widgettinä. Samoihin widgetteihin perustuvat myös Heitä-nappula, pelaajien tulostaulukkoiden pisteet yksittäisille yhdistelmille ja pika-asetusten lukumäärät (päänäytön vasen alanurkka). Jäljellä olevien

heittojen lukumäärä sekä sekä kaikki muu teksti toteutetaan Label-widgettinä. Peli- ja Asetukset -valikot [4][5] toteutetaan Menu ja MenuButton -widgettejen avulla. Pelaajien nimien kirjoitus-valikko [3] toteutetaan Entry-widgetin avulla. Noppien lukumäärän kyselyyn käytetään Spinbox-widgettiä. Peli-asetustiedoston avaamiseen [2] käytetään tkFileDialog.askopenfilename-ikkunaa.



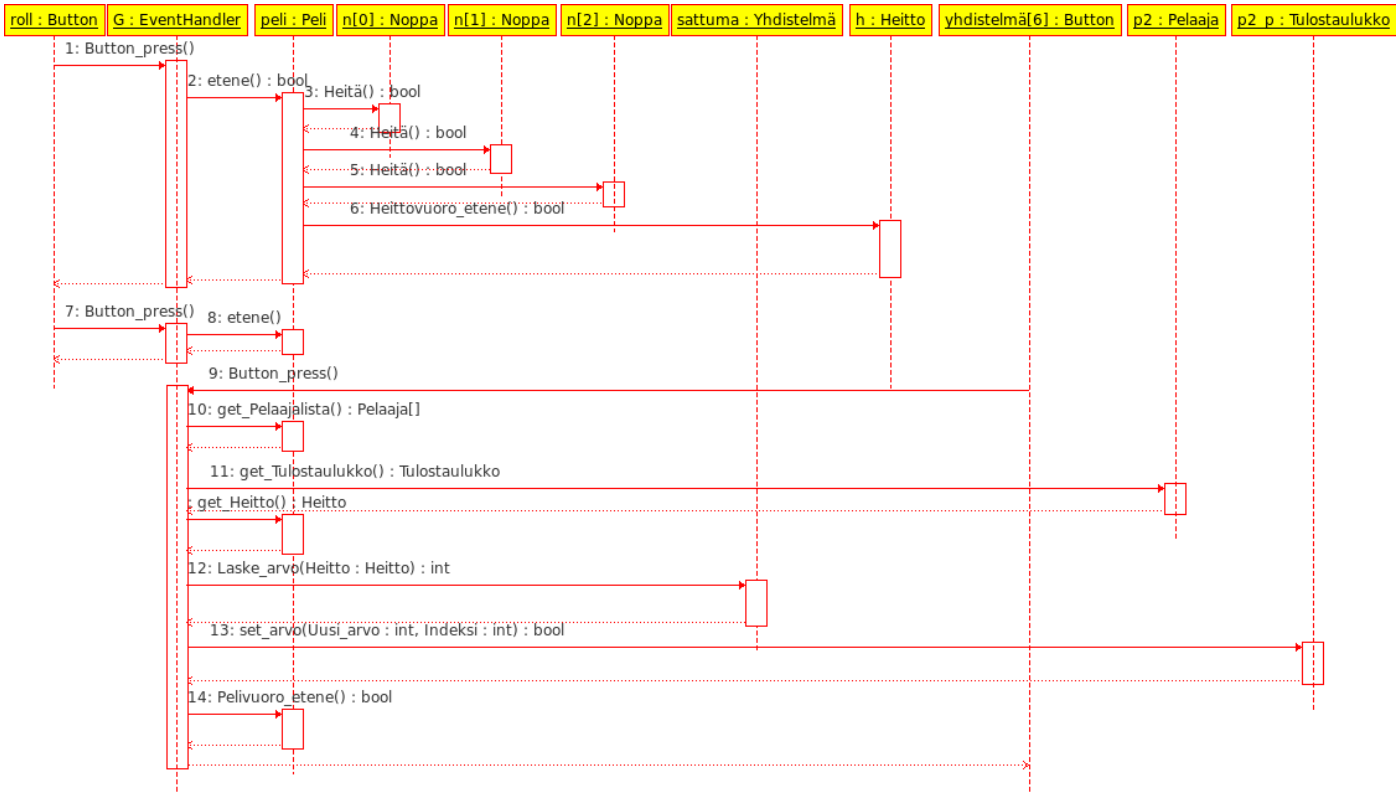
Kuva 2. Suunnitelma yleisen noppapelin graafiselle käyttöliittymälle. [1] Pelin päänäyttö. [2] Peliasetustiedoston valintaikkunan ulkonäkö riippuu käyttöjärjestelmästä. [3] Peliin osallistuvien pelaajien nimeäminen tapahtuu uuden pelin yhteydessä. [4] Peli-valikosta voi aloittaa uuden pelin ja lopettaa nykyisen sekä katsella top-10 listaa. [5] Asetukset-valikosta voi vaihtaa peliasetustiedostoa sekä noppien lukumäärää, maksimisilmälukua sekä heitto-vuoron heittokertojen määrää. Tämä onnistuu vain pelikertojen välissä. [6] Pikanäppäinten lisäksi Asetukset-valikon kolmea alinta asetusta voi muuttaa valintaikkunan kautta.



Kuva 3. Luokkakaavio graafisen käyttöliittymän luokkien suhteista. T viittaa Tkinter-moduliin.

Tarkemmin pelin graafisen käyttöliittymän luokkia on esitelty luokkakaaviossa (Kuva 3). Pelin pääikkunaan liittyy valikon Menubuttoneiden lisäksi kaksi Frame-objektia, jotka sisältävät pelin nopat ja pelin pistetaulukon. Valikon kautta avautuu uusia ikkunoita joidenkin valintojen tekemiseen (Kuva 2). Nämä ovat pääosin melko yksinkertaisia valikkoja, joiden toiminta palautuu suoraan pelimoottorin metodiksi.

4 Sekvenssikaavio pelivuoron etenemisestä



Kuva 4. Sekvenssikaavio yhdestä pelivuorosta. Tapahtumakulku tässä sekvenssikaaviossa on seuraava. Pelaaja painaa "Heitä" -nappulaa (1). Pelin etene-metodi (2) heittää noppia (3-5) ja siirtää heittovuoroa eteenpäin (6). Tässä vaiheessa noppien silmäluvut vaihtuvat. Kyseisessä esimerkkipelissä ei ole kuin yksi heittokerta heittovuoroa kohti, joten kun pelaaja yrittää heittää uudestaan (7), ei tapahdu mitään (8). Kun pelaaja valitsee jonkin yhdistelmän painamalla tätä lukuarvonappulaa (9), päivitetään pelaajan (10) tulostaulukon (11) arvo heiton avulla lasketulla arvolla (12). Kun tämä lukuarvo on asetettu (13) siirretään pelivuoroa seuraavalle pelaajalle (14).