Kuisma Joentakanen

PYGAME-PELIOHJELMOINTI

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Marraskuu 2010

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Tekniikka ja liiketalous, Kokkola	**Aika** 26.10.2010	**Tekijä/tekijät** Kuisma Joentakanen
Koulutusohjelma Tietotekniikka		
Työn nimi Pygame-peliohjelmointi		
Työn ohjaaja DI Kauko Kolehmainen		**Sivumäärä** 60 + 5 liitettä
Työelämäohjaaja		

Opinnäytetyön tarkoituksena oli selvittää, miten yksinkertaisten ja kaksiulotteisten pelien ohjelmointi onnistuu Python-ohjelmointikielen ja sen Pygame-moduulin avulla. Pygame rakentuu C-kielellä toteutetun SDL-multimediakirjaston päälle, ja sen avulla voidaan luoda kaksiulotteista grafiikkaa, hallita äänitehosteita ja vastaanottaa käyttäjän syötteitä.

Työssä tutustuttiin ensin Pythonin perusominaisuuksiin ja sen jälkeen varsinaiseen peliohjelmointiin. Lähteenä käytettiin ohjelmointioppaita ja virallista Pygamen dokumentaatiota. Työssä käydään Pygamen perusominaisuudet läpi viiden itse tehdyn esimerkkipelin kautta.

Pygame soveltuu yksinkertaisten pelien ohjelmointiin erittäin hyvin. Pelien tekeminen Pygamella on sekä hauskaa että helppoa.

Asiasanat
Python, Pygame, peliohjelmointi, tietokonepeli

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	**Date** 26.10.2010	**Author** Kuisma Joentakanen
Degree programme Information Technology		
Name of thesis Programming Games with Pygame		
Instructor Kauko Kolehmainen		**Pages** 60 + 5 Appendices
Supervisor		

Pygame is a module written for the Python programming language. Pygame runs on top of the Simple Directmedia Layer (SDL) programming library which is written in the C programming language. The purpose of the thesis was to examine the Pygame module and to write simple, two-dimensional computer games with it.

The method of carrying out the thesis was first to study the Python language and then go deep into the basics of Pygame. A total of five example games were written for this thesis. The basic features of Pygame were used and examined in those examples.

As the result of the thesis can be said that the Pygame module is very suitable for simple game programming. Pygame is both easy and fun to learn.

Key words

Python, Pygame, game programming, computer games

SISÄLLYS

1 JOHDANTO

Python on yksi tämän päivän suosituimmista ohjelmointikielistä. Python tukee useaa ohjelmointiparadigmaa ja sopii siten lähes minkälaiseen ohjelmointiin tahansa. Pythonilla tehdään mm. matkapuhelinsovelluksia, palvelinpuolen ohjelmointia ja tieteellistä laskentaa. Kielen on sanottu olevan syntaksiltaan yksinkertainen ja siten helposti opittavissa. Palvelinpuolen ohjelmoinnissa Pythonia käytetään usein yhtenä LAMP-kokonaisuuden osana. LAMP koostuu avoimen lähdekoodin ohjelmistoista, joita ovat Pythonin lisäksi Linux-käyttöjärjestelmäydin, Apache-webpalvelin ja MySQL-tietokantarajapinta.

Python on tulkattava ohjelmointikieli. Tulkki ajaa ohjelmaa reaaliajassa, eli ohjelmakoodia luetaan, käännetään tavukoodiksi ja suoritetaan lause kerrallaan. Tulkkaus tapahtuu joka kerta, kun ohjelmaa halutaan suorittaa, ja siksi tulkattavat kielet eivät pärjää nopeudessa käännetyille kielille. Suoritusnopeutta voidaan parantaa tekemällä esikäännettyjä tiedostoja, jotka voidaan suorittaa tulkilla saman tien. Esikäännetyn Python-tiedoston tunnistaa päätteestä pyc.

Pääasiallisina opetettavina ohjelmointikielinä koulussani ovat olleet Java ja C, jotka kumpikin ovat varsin vaikeita kieliä oppia. Opinnäytetyön idea syntyi kiinnostuksesta Python-kieltä kohtaan. Olin harrastunut Python-ohjelmointia jo jonkin verran ennen aiheen valintaa. Python vaikutti positiivisella tavalla erilaiselta ohjelmointikieleltä, josta on poistettu kaikki turha ja johon on jätetty vain se olennainen. Varsinainen peliohjelmoinnin teoria ei ollut minulle ennestään tuttua.

Opinnäytetyön tarkoituksena oli selvittää, miten yksinkertaisten ja kaksiulotteisten pelien ohjelmointi onnistuu Python-ohjelmointikielen ja sen Pygame-moduulin avulla. Pygame rakentuu C-kielellä toteutetun, matalan tason multimediakirjasto SDL:n päälle. Pygamen avulla Python-kielellä voi luoda kaksiulotteista grafiikkaa, käsitellä äänitiedostoja ja hallita käyttäjän syötteitä.

Luvussa kaksi kerrotaan ohjelmointikielten historiasta, ominaisuuksista ja luokittelusta. Luvussa kolme käydään läpi Python-ohjelmointikielen historiaa ja perusominaisuuksia sekä tehdään yksinkertainen, graafinen käyttöliittymä Tkinter-moduulilla. Luvussa neljä

käydään läpi tietokonepeligenret ja kerrotaan pelin perusrakenteista kuten moduuleista ja pelisilmukan toiminnasta. Luvussa viisi syvennytään varsinaiseen Pygame-peliohjelmointiin. Luvussa esitellään ilmaisjakelussa oleva valmis peli sekä käydään läpi Pygamen perusominaisuuksia viiden esimerkkiohjelman kautta.

Jussi Pekka Kasurisen Python 3 -ohjelmointi -kirjasta oli valtavasti hyötyä Pythonin opiskelussa. Kirja oli helppolukuinen, ja sen avulla pääsi hyvin perille kielen perusominaisuuksista. Olin jo aikaisemmin tutustunut Kasurisen Lappeenrannan yliopistolle kirjoittamiin, lyhyempiin Pythonia käsitteleviin oppaisiin. Al Sweigartin Invent Your Own Computer Games With Python -kirja sisältää puolestaan useita laadukkaita peliohjelmointiesimerkkejä ja ylipäätään hyviä vinkkejä peliohjelmointia ajatellen. Otin kirjan esimerkeistä paljon mallia omia esimerkkiohjelmia tehdessäni.

2 YLEISTÄ OHJELMOINTIKIELISTÄ

2.1 Mitä ohjelmointi on?

Yksinkertaisimmin ilmaistuna ohjelmointi on sitä, että ohjelmoija kirjoittaa käskyjä, jotka tietokone tulkitsee ja joiden perusteella tietokone lopulta tekee jotakin. Tietokoneohjelmassa määritellään kaikki yksinkertaisimmatkin toimenpiteet, jotka tietokoneen halutaan tekevän, oli se sitten keltaisen pallon piirtäminen ruudulle tai vaikka vain seuraavan käskyn odottaminen. Ohjelmakoodin tulee olla virheetöntä ja yksiselitteistä, sillä tietokone ei pysty arvailemaan, mitä ohjelmoija kenties tarkoitti, eikä siten myöskään korjaamaan koodia oikeanlaiseksi. Viallinen koodi voi johtaa siihen, ettei ohjelma toimi halutulla tavalla. (Kingsley-Hughes & Kingsley-Hughes 2005, 4.)

Ihmisen kirjoittama koodi tulee ensin kääntää tai tulkata konekielelle, ennen kuin tietokone pystyy sitä ymmärtämään. Konekieli koostuu biteistä eli ykkösistä ja nollista. Jotkin ohjelmointikielet, kuten esimerkiksi Visual Basic, vaativat oman kehitysympäristön, kun taas toisilla kielillä voi tehdä ohjelmia vaikka ihan tavallisella tekstinkäsittelyohjelmalla. (Kingsley-Hughes & Kingsley-Hughes 2005, 4.)

2.2 Ohjelmointikielten sukupolvet

Ensimmäiset ohjelmoitavat tietokoneet kehitettiin 1940-luvun lopussa. Koneiden arkkitehtuurista vastasi John Von Neumann, joka puolestaan pohjasi ajatuksensa Alan Turingin ideaan abstraktista koneesta. Neumannin koneissa keskusyksikkö vastasi muistiin tallennettujen ohjelmien suorittamisesta. Ensimmäistä ja ainoaa tällaista konetta kutsuttiin nimellä ENIAC ja se valmistui vuonna 1945. ENIAC koostui 18 000 elektroniputkesta, ja sitä ohjelmoitiin kytkentätauluun liitettävien johtojen avulla. (Vesanen 2006.)

Ohjelmointikielet voidaan jakaa viiteen sukupolveen. Ensimmäisen sukupolven kielet olivat konekieliä, eli ne kirjoitettiin tietokoneen muistiin binäärimuodossa. Konekielellä ohjelmointi oli erittäin vaikeaa ja virhealtista, ja 1950-luvulla kehitettiinkin symboliset konekielet, jotka luetaan toisen sukupolven kieliin. Näissä kielissä käskyille ja muistipaikoille

voitiin antaa tekstimuotoinen symboli, mikä helpotti muistamista. (Vesanen 2006.)

Kolmannen, neljännen ja viidennen sukupolven kielien määrittelemiseen ei ole yhtä ainoaa universaalia tapaa. Kolmannen sukupolven kieliin voidaan sanoa kuuluvan kaikki korkean tason kielet, jotka on kehitetty 1950-luvun lopulla tai sen jälkeen. Tähän kuuluvat myös nykypäivän oliopohjaiset kielet. Kolmannen sukupolven kielet ovat joko tulkattavia tai käännettäviä. (Vesanen 2006.)

Neljännen sukupolven kieliksi yleensä lasketaan1980-lvulta lähtien yleistyneet tietokantojen kyselykielet ja muut verkkoympäristöissä käytettävät kielet. Tämän sukupolven kielien syntaksi on helpompaa mutta vähemmän tehokasta kuin tyypillisten korkean tason kielten. Niinpä ne soveltuvatkin paremmin yksinkertaisempien ohjelmien luontiin. Viidennen sukupolven kielet ovat pääosin vielä kehitysasteella. Tämän polven kielien syntaksi on hyvin samankaltainen luonnollisten kielien kanssa, ja niitä käytetään pääasiassa tekoälyä vaativien sovellusten ohjelmointiin. (Vesanen 2006.)

2.3 Ohjelmointikielten historia

Vuonna 1957 IBM sai valmiiksi FORTRANin (Formula Translator), jota pidetään ensimmäisenä korkean tason ohjelmointikielenä. FORTRANilla oli hyvin suuri merkitys ohjelmointikielien kehityksessä, sillä se osoitti korkean tason ohjelmointikielien toteuttamisen mahdolliseksi myös käytännössä. FORTRANin ominaisuuksia olivat mm. tulostuksen ja syötön muotoilu, looginen if-lause, johon voitiin antaa algebrallisia vertailuoperaatioita, ja toistorakenne laskurimuuttujan suhteen do-lausekkeella. FORTRANilla ei voinut määritellä tietotyyppejä, vaan kirjaimilla I, J, K, L, M ja N alkavat muuttujat olivat kokonaislukuja ja muilla aakkosilla alkavat muuttujat liukulukuja. FORTRAN-kääntäjä teki konekieltä lähes yhtä nopeasti kuin kokenut ohjelmoija, ja kääntäjän hienosäätö veikin kielen suunnitteluajasta leijonanosan. FORTRANia käytetään edelleen jonkin verran tieteelliseen ja numeeriseen ohjelmointiin. (Vesanen 2006.)

1950-luvulla kehitettiin myös ensimmäiset funktionaaliset ohjelmointikielet. John McCarthyn ja Marvin Minskyn tekoälyprojektin tuloksena syntyi LISP (List Processing Language). LISP ei ole perinteinen, imperatiivinen kieli vaan se käsittelee vain kahdenlai-

sia tietorakenteita: atomeja ja listoja. Atomit ovat joko numeroita tai symboleja. Listoihin voi kuulua muita listoja tai atomeja. LISP oli ensimmäinen kieli, joka sisälsi roskien keruun. (Vesanen 2006.)

1960-luvun valtakieliä olivat FORTRAN, COBOL ja ALGOL. Kahta ensiksi mainittua käytettiin tieteelliseen laskentaan ja COBOLia talouspuolen sovelluksiin. 1960-luvulla kehitettiin BASIC (Beginner's All-purpose Symbolic Instruction Code) humanistisen alojen opiskelijoita varten. Kohderyhmänsä takia kielestä tehtiin yksinkertainen jättämällä mm. lohkorakenne pois. BASICia käytettiin yleisesti etäpääteyhteyksien kanssa. BASICin pohjalta on kehitetty uusia kieliä, joista tunnetuin lienee Visual Basic. (Vesanen 2006.)

Myös olio-ohjelmointi sai alkunsa samalla vuosikymmenellä. Ensimmäisenä oliokielenä voidaan pitää norjalaista Simulaa, joka nimensä mukaisesti kehiteltiin simulointitarkoituksiin. Tunnetuin versio tästä on Simula 67, joka pohjautui hyvin pitkälle ALGOL 60:een. Simulalla pystyi suorittamaan rinnakkain aliohjelmia, mistä sai alkunsa datan abstrahointi. Simulassa oli luokkien ja perinnän lisäksi roskienkerääjä. Simulan suosio jäi vaatimattomaksi. (Vesanen 2006.)

1970-luvulla pyrittiin kehittämään ohjelmointia yksinkertaisempaan, johdonmukaisempaan ja rakenteellisempaan suuntaan. Rakenteinen ohjelmointi (structured programming) olikin vuosikymmenen suosituin ohjelmointiparadigma. Vuonna 1971 sai alkunsa Pascal, joka suunniteltiin ohjelmointiopetusta varten. Pascal olikin suosituin opetuskieli yli kahden vuosikymmenen ajan. Varsinaiseen sovelluskehitykseen kieli oli kuitenkin liian köykäinen. (Vesanen 2006.)

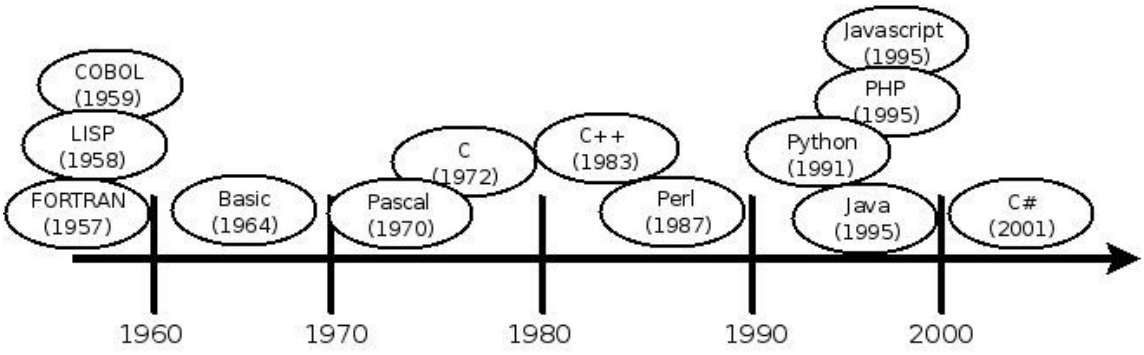
Vuonna 1972 julkaistiin C, joka on pysynyt suosiossa nykypäivään saakka. C suunniteltiin alun perin järjestelmäohjelmointiin. Pian UNIX-käyttöjärjestelmä kirjoitettiin uudelleen C:llä, ja tämä auttoi kielen leviämistä. C:ssä merkkijonoja käsitellään merkkitaulukkoina ja niitä hallitaan osoittimilla. Kielenä C on joustava, ja se kääntyy helposti konekielelle. C on vaikuttanut suuresti myöhempien kielien kehitykseen. (Vesanen 2006.)

1980-luvulla löi olio-ohjelmointi lopullisesti läpi. Bjarne Stroustrup kehitteli C-kielen pohjalta uutta oliopohjaista kieltä, jonka tulisi olla yhtä tehokas kuin esikuvansa. Olioiden käyttäytymiseen Stroustrup haki mallia Simula 67:stä ja Smalltalkista. C with classes -työ-

nimestä tuli lopulta C++. Se oli hyvin pitkälti yhteensopiva C:n kanssa, ja C-kieliset ohjelmat oli siten helppo muuntaa uudelle kielelle. C-yhteensopivuuden ja runsaan kääntäjätarjonnan myötä C++:sta tuli erittäin suosittu, ja suosio on säilynyt nykypäivään saakka, vaikkakin kieltä pidetään monimutkaisena. Vuosikymmenen lopulla saivat alkunsa myös useat skriptikielet, joista esimerkiksi Perl soveltui myöhemmin web-ympäristöjen CGI-ohjelmointiin. (Vesanen 2006.)

1990-luvun merkittävimpiä uusia tulokkaita olivat ainakin Java, Javascript, PHP ja Python. Javasta oli tarkoitus tulla luotettava sulautettujen järjestelmien ohjelmointikieli, mutta lopulta kieli löi itsensä läpi internetin yleistymisen myötä www-ohjelmoinnissa. Java muistuttaa hyvin paljon C++:aa, mutta tietyt C++:n ominaisuudet, kuten osoittimet ja moniperintä, jätettiin kielestä pois. Uusia ominaisuuksia ovat rinnakkaisen ohjelmoinnin tuli ja automaattinen roskien keruu. (Vesanen 2006.)

Vuonna 2000 Microsoft julkaisi C#-kielen, joka on osa .NET-konseptia. C# on puhtaampi oliokieli kuin esimerkiksi Java. Lisäksi kieli tukee mm. operaattorien ylikuormitusta (KUVIO 1). (Vesanen 2006.)



KUVIO 1. Merkittävimmät modernit ohjelmointikielet aikajanalla (Vesanen 2006.)

2.4 Ohjelmointikielten luokittelu

2.4.1 Ohjelmointiparadigmat

Ohjelmointiparadigma tarkoittaa ohjelmointitapaa. Ohjelmointikielet voidaan jakaa neljään pääparadigmaan: imperatiivinen ohjelmointi, olio-ohjelmointi, funktionaalinen ohjelmointi ja logiikkaohjelmointi. Imperatiivisessa ohjelmoinnissa muistia manipuloidaan muuttujien ja sijoituslauseiden avulla ja komennot suoritetaan peräkkäin. Suurin osa nykypäivän kielistä onkin ainakin jollakin tavalla imperatiivisia. Proseduraalinen ohjelmointi luetaan myös imperatiiviseen paradigmaan kuuluvaksi. Siinä ohjelma on jaettu pieniin aliohjelmiin. (Vesanen 2006.)

Olio-ohjelmointi luetaan omaksi paradigmakseen, vaikka useat oliokielet ovatkin luonteeltaan imperatiivisia. Olio-ohjelmoinnissa ohjelman toiminta riippuu aktiivisten tietoalkioiden (olio) vastaanottamista ja toisilleen lähettämistä viesteistä. Useat oliokielet tukevat rinnakkaisuutta, eli niissä voidaan määritellä käskyjä, jotka toimivat samanaikaisesti. (Vesanen 2006.)

Funktionaalisessa ohjelmoinnissa tietoa käsitellään matemaattisilla funktioilla eli funktiokutsuilla. Sijoituslauseita ja silmukoita ei ole lainkaan, ja tiedon tallentaminen perustuu rekursioon. Logiikkaohjelmoinnissa käytetään symbolista logiikkaa, eli ohjelma koostuu totuutta kuvaavista väitteistä. Muuttujat eivät mallinna muistipaikkoja vaan ne toimivat matemaattisten muuttujien tapaan. (Vesanen 2006.)

2.4.2 Ohjelmointikielien suosio nykyään

Ohjelmointikielien suosiosta on vaikea saada täysin paikkaansa pitävää tietoa, mutta esimerkiksi TIOBE Softwaren ylläpitämä tilasto (TAULUKKO 1) antaa hyvin osviittaa siitä, mitkä ohjelmointikielet nykypäivänä kiinnostavat. Tilasto on tehty suosituimpien hakukoneiden tulosten perusteella. Käytetyt hakukoneet ovat Google, Google Blogs, MSN, Yahoo!, Wikipedia ja Youtube web search. Hakukoneet on valittu www.alexa.com:in luokittelun perusteella. Hakusanana on käytetty seuraavaa:

+”<language> programming”

Hakuosumien määrä määrittelee kielen suosion. Lasketut osumat normalisoidaan tietyn matemaattisen kaavan mukaan virheiden poistamiseksi. Voidaan sanoa, että mitä suositumpi kieli on, sitä enemmän kielelle on tarjolla osaavia ohjelmoijia, hyviä ohjelmointiympäristöjä, kääntäjiä ja tulkkeja sekä lisäkirjastoja. Toisaalta tietyt, vähemmän suositut kielet soveltuvat tiettyyn käyttötarkoitukseen ns. mainstream-kieliä paremmin. Esimerkkinä on vaikkapa Ada, jota käytetään sotilastarkoituksiin. (TIOBE Software 2010a; TIOBE Software 2010b.)

TAULUKKO 1. Ohjelmointikielten suosio TIOBE:n mukaan 5/2010

Sijoitus v. 2010	Sijoitus v. 2009	Kieli	Osuus	Muutos vuodesta 2009
1	2	C	18,186 %	+2,06 %
2	1	Java	17,957 %	–1,58 %
3	3	C++	10,378 %	–0,69 %
4	4	PHP	9,073 %	–0,85 %
5	5	(Visual) Basic	5,656 %	–2,97 %
6	7	C#	4,779 %	+0,51 %
7	6	Python	4,097 %	–1,45 %
8	9	Perl	3,286 %	–0,24 %
9	11	Delphi	2,566 %	+0,24 %
10	39	Objective-C	2,363 %	+2,23 %
11	10	Ruby	2,094 %	–0,60 %
12	8	JavaScript	2,084 %	–1,46 %
13	12	PL/SQL	0,859 %	–0,24 %
14	13	SAS	0,732 %	–0,07 %
15	14	Pascal	0,728 %	–0,05 %
16	22	Lisp/Scheme/Clojure	0,651 %	+0,19 %
17	16	ABAP	0,650 %	–0,02 %
18	-	Go	0,640 %	+0,64 %
19	18	MATLAB	0,612 %	+0,09 %
20	20	Lua	0,493 %	+0,01 %

3 PYTHON

3.1 Pythonin historia

Python on verrattain tuore ohjelmointikieli. Guido Van Rossum aloitti kielen kehittämisen 1980-luvun lopussa. Rossumin tavoitteena oli tehdä vanhasta ABC-kielestä versio joka suoriutuisi myös virheidenkäsittelystä ja käyttöjärjestelmän kanssa kommunikoinnista ja joka olisi ulkoasultaan yksinkertainen ja selkeä. Rossum julkaisi Pythonin lähdekoodin (versio 0.9) vuoden 1991 helmikuussa. Jo tuossa vaiheessa Pythonin sisältämiä ominaisuuksia olivat perintä, poikkeuksienhallinta, funktiot ja ydin-tietotyypit kuten lista, luettelo ja merkkijono sekä moduulikirjastot. (Kasurinen 2009, 7.)

Virallinen 1.0 versio julkaistiin tammikuussa 1994. Sen tärkeimpiä lisäyksiä olivat funktiot lambda, map(), filter() ja reduce(). 16. lokakuuta 2000 julkaistun versio 2.0:n myötä Pythonin kehitystiimi otti käyttöön CVS-versiohallintaohjelmiston, ja lähdekoodit julkaistiin jatkossa SourceForge-sivuston kautta osoitteessa http://sourceforge.net/projects/python/. Tekijöiden mukaan CVS:n käyttöönotto nopeutti kielen kehitystä merkittävästi. Version uusia ominaisuuksia olivat mm. listan käsittely ja tehokkaampi roskienkeruu. (Kuchling & Zadka 2000; Python Software Foundation 2006.)

Versio 3.0 julkaistiin 3. joulukuuta 2008. Se oli ensimmäinen Python-jakelu, joka ei ollut taaksepäin yhteensopiva, eli jotkin vanhemmissa versioissa käytetyt tavat eivät enää toimineetkaan uudessa versiossa. Poistamalla tiettyjä tarpeettomia ja virheille alttiita ominaisuuksia pyrittiin kielen syntaksia muokkaamaan yksinkertaisempaan ja selkeämpään muotoon. Print-lause muuttui funktioksi. Print-funktiolle voidaan lisäksi määritellä rivin lopetus- ja rivitysmerkit. (Kasurinen 2009, 8.)

```
Vanha tapa:
print ”Tänään on tiistai”
Uusi tapa:
print(”Tänään on tiistai”)
```

Syötteitä vastaanotetaan vain ja ainoastaan input-käskyllä, joka korvasi vanhat input- ja raw_input-käskyt.

```
päivämäärä = input(”Mikä päivä tänään on?”)
```

(Kasurinen 2009, 8.)

Kahden eri tyyppisen muuttujan, kuten esimerkiksi merkkijonon ja kokonaisluvun tai listan ja tuplen vertailu loogisten operaattorien avulla palauttaa TypeError-virheen Boolean-arvon sijaan. Jakolasku palauttaa aina liukuluvun, oli jaettavana sitten pelkkiä kokonaislukuja tai sekä kokonais- että liukulukuja. Lisäksi tulostuksen muotoilu (%-muotoilu) tulee muuttumaan tulevissa versioissa. Python Software Foundation on vastannut kielen kehityksestä ja jakelusta vuodesta 2001 alkaen. (Kasurinen 2009, 8; Otero 2008.)

3.2 Pythonin ominaisuudet

Python on alustariippumaton, avoimen lähdekoodin tulkattava ohjelmointikieli, ja se tukee useita eri ohjelmointiparadigmoja, kuten olio-ohjelmointia, funktionaalista ohjelmointia ja proseduraalista ohjelmointia. Pythonin virallinen asennuspaketti koostuu tulkista, IDLE-lähdekoodieditorista ja debuggaustyökaluista. IDLE:n käyttö on vapaaehtoista, mutta ohjelmia suositellaan ajettavan virallisella tulkilla. (Kasurinen 2009, 2–11.)

Pythonissa muuttujan tietotyyppiä ei tarvitse etukäteen määritellä, eli ohjelmoijan ei ole välttämätöntä tietää, tallennetaanko muuttujaan esimerkiksi kokonaislukuja, tekstiä, listoja tai vastaavaa. Luodun muuttujan tyyppiä voidaan muuttaa kesken ohjelman suorituksen. Muuttujia voidaan luoda missä osassa koodia tahansa paitsi esimerkiksi toistorakenteissa ja loogisissa vertailuissa, joissa tulkin pitää tietää muuttujan arvo etukäteen. Muuttujien nimet ovat vapaasti valittavissa tiettyjä poikkeuksia, kuten esimerkiksi syntaksille varattuja sanoja, lukuun ottamatta. (Kasurinen 2009, 19–21.)

Pythonin kolmosversiossa syötteitä pyydetään input-funktion avulla. Input vastaanottaa tietoa pelkästään merkkijonomuodossa, joten numeroarvoja pyydettäessä joudutaan käyttämään tyyppimuunnoksia. Esimerkiksi merkkijonon ”250” kertominen kahdella tuottaa tuloksen ”250250”, eli merkkijonoa toistetaan kertoimen ilmoittaman määrän verran. Tyyppimuunnosfunktio palauttaa käyttäjälle arvon halutussa muodossa, kuten esimerkiksi kokonaislukuna. Tyyppimuunnosfunktio ei muuta muuttujan tyyppiä suoraan vaan tulos pitää

tallentaa uuteen muuttujaan. (Kasurinen 2009, 8.)

```
arvo = "1992"
int(arvo)
print(arvo*2)
```

Edellä esitetty esimerkki tulostaa arvon ”19921992”, koska tyyppimuunnettua arvo-muuttujaa ei tallennettu. Seuraavassa esimerkissä merkkijonomuodossa oleva arvo-muuttuja kierrätetään eli kokonaislukumuunnos tallennetaan sen päälle:

```
arvo = int(arvo)
print(arvo*2)
```

Tulokseksi tulee ”3984”, eli arvo-muuttujan muuntaminen kokonaisluvuksi onnistui, ja sille pystyy tämän jälkeen suorittamaan laskutoimituksia. Int:n lisäksi Pythonista löytyy useita muita tyyppimuunnosfunktioita. (Kasurinen 2009, 32–37.) Seuraavissa luvuissa syvennytään tarkemmin Pythonin ominaisuuksiin.

3.2.1 Sisennykset ja muu ulkoasu

Monista muista kielistä poiketen Pythonissa välilyönnit ja sisennykset vaikuttavat ratkaisevalla tavalla ohjelman toimivuuteen, ja siksi ohjelmointi kannattaa tehdä sellaisella editorilla, joka ymmärtää Pythonin syntaksia. (Kasurinen 2009, 38–41.)

Python-tulkki jakaa lähdekoodin loogisiin osiin sen mukaan, millä sisennystasolla lähdekoodirivi sijaitsee. (Kasurinen 2009, 40.) Seuraavassa esimerkissä print-käskyt suoritetaan vain silloin, kun muuttujan tila arvo on ”vapaa”:

```
if tila == "vapaa":
    print("Sali ei ole käytössä.")
    print("Voit mennä sisään.")
```

Jos jälkimmäisen tulostuslauseen sisennys nostetaan samaksi kuin if-lauseella, suoritetaan toinen print-lause aina tila-muuttujasta riippumatta:

```
if tila == "vapaa":
    print("Sali ei ole käytössä.")
print("Voit mennä sisään.")
```

Samalla sisennystasolla olevat rivit ovat siis loogisesti toisiaan seuraavia lauseita. If-lausetta seuraavat, alemmalle tasolle sisennetyt rivit kuuluvat vuorostaan if-lauseen koodiosioon. Sisennykset tulisikin tehdä aina käyttäen joko tabulaattoria tai välilyöntiä eikä niiden sekoituksia. Python-syntaksia tukevalla editorilla sisennyksien ei pitäisi aiheuttaa ongelmia. (Kasurinen 2009, 38–41.)

3.2.2 Merkkijonojen käsittely

Kasurinen (2009, 122–130) kuvaa merkkijonojen käsittelyä seuraavasti: Pythonissa on kaksi sisäänrakennettua merkkijonojen käsittelytapaa: leikkauksilla merkkijonot voidaan pilkkoa nopeasti ja tehokkaasti osiin ja valita osia myöhempää käyttöä varten, ja vastaavasti metodien avulla merkkijonoja on helppo muotoilla ja testata. Leikkauksien avulla määritellään osajoukkoja, jotka poimitaan merkkijonosta. Hakasuluissa ilmoitetaan, monenteenko merkkiin halutaan viitata:

```
merkkijono = "kolmivaihekilowattituntimittari"
print(merkkijono[3])
```

Ensimmäinen alkio on aina järjestysluvultaan 0, eli ”kolmosmerkkiin” viittaaminen antaa tulokseksi m-kirjaimen. Plusmerkkiset luvut laskevat alusta loppuun ja miinusmerkkiset toisinpäin. −1 siis viittaa merkkijonon viimeiseen merkkiin, ja −0 sen sijaan vastaa nollaa. Merkkijonon pituuden voi selvittää funktiolla len:

```
merkkijono = "tuhansittain"
pituus = len(merkkijono)
print("Sanan pituus on",pituus,"merkkiä.")
```

Useamman merkin leikkaus onnistuu siten, että hakasulkujen sisään sijoitetaan leikattavan alueen aloituspaikka, päätöspaikka ja siirtymäväli. Päätöspaikaksi asetetaan se merkki, jonka jälkeen leikkauksen halutaan päättyvän. Leikkaus voi alkaa mistä tahansa kohtaa merkkijonoa ja päättyä mihin tahansa kohtaan. Seuraavassa esimerkissä leikataan merkkijonosta viisi ensimmäistä merkkiä:

```
mjono = "autokorjaamo"
leikkaus = mjono[0:5]
```

Ohjelma tulostaa ”autok”. Seuraavassa esimerkissä leikataan merkkijonoa keskeltä:

```
mjono = "autokorjaamo"
leikkaus = mjono[3:10]
```

Tulostuksena on ”okorjaa”. Seuraavassa esimerkissä määritellään siirtymäväliksi kolme:

```
mjono = "korjaamopäällikkö"
leikkaus = mjono[0:15:3]
```

Merkkijonosta siis leikataan väliltä 0–15 joka kolmas merkki. Tuloksena saadaan ”kjmäl”. Merkkijono voidaan kääntää väärinpäin seuraavalla tavalla:

```
mjono = "ammattikorkeakouluopiskelija"
kääntö = mjono[::-1]
print(kääntö)
```

(Kasurinen 2009, 122–130.)

Leikkauksien kanssa on mahdollista päätyä IndexError-virhetilanteeseen, jos viitataan sellaiseen merkin paikkaan, joka on merkkijonon ulkopuolella. Kuitenkin sellaisten leikkausten kohdalla, joissa on määritelty alku- ja loppupiste, tulkki palauttaa vain tyhjän arvon tai leikkauksen alueelle kuuluvan osasen virhesanoman sijaan. (Kasurinen 2009, 122–130.)

Leikkauksia ja muokkausfunktioita (metodeja) yhdistelemällä voidaan merkkijonoja käyttää tehokkaasti hyödyksi. Merkkijonoja voidaan testata eri tavoin ja muuntaa helposti haluttuun muotoon, kuten esimerkiksi pikkukirjaimista isoiksi kirjaimiksi. Metodit palauttavat merkkijonon halutussa muodossa tai boolean-arvon True tai False sen mukaan, oliko testin tulos tosi vai ei. Seuraavassa on joitain esimerkkejä metodien käytöstä:

```
mjono = "limonadiautomaatti"

#testataan sisältääkö muuttuja pelkkiä numeroita
mjono1 = mjono.isalpha()

#muunnetaan isoiksi kirjaimiksi
mjono2 = mjono.upper()

#korvataan merkkijonosta syöte1 syötteellä 2
mjono3 = mjono.replace("onadi","ppari")
```

```
#Testataan alkaako merkkijono sanalla piste
mjono4 = mjono.startswith("piste")

print(mjono1)
print(mjono2)
print(mjono3)
print(mjono4)
```

Tulos:

```
True
LIMONADIAUTOMAATTI
limppariautomaatti
False
```

(Kasurinen 2009, 122–130.)

3.2.3 Valintarakenteet

If-else-rakenne perustuu loogisille väittämille. If-lauseeseen liitetty koodiosio ajetaan, jos väittämä on tosi (True). Mikäli kyseinen osio on epätosi (False), suoritetaan else-rakenteen osio jos sellainen on määritelty. Muuten ohjelmakoodin suorittamista jatketaan eteenpäin samalta tasolta. If-lauseita voidaan ketjuttaa peräkkäin elif-osion (muissa kielissä usein else if) avulla. Elif-osio on else-osion tapaan vapaaehtoinen, mutta kumpikin vaatii if-osion olemassaolon. Elif-osioita voi olla rajoittamattomasti.(Nikula & Vanhala 2010, 24.)

```
muuttuja = input("Anna jokin kokonaisluku:")
muuttuja = int(muuttuja)

if muuttuja > 5:
    print("Antamasi luku", muuttuja, "on suurempi kuin 5")
elif muuttuja == 5:
    print("Antamasi luku", muuttuja, "on yhtä suuri kuin 5")
else:
    print("Antamasi luku", muuttuja, "on pienempi kuin 5")
```

Edellä esitetyssä esimerkissä kysytään käyttäjältä kokonaislukua, joka tarkistetaan if-elif-else-rakenteen avulla. If-osiossa testataan, onko käyttäjän antama luku isompi kuin viisi. Mikäli se ei ole, ohjelma siirtyy elif-osioon, joka tarkastaa, onko luku yhtä suuri kuin 5. Mikäli luku ei ole kumpaakaan, täytyy sen luonnollisesti olla pienempi kuin viisi. Else-

osioon ei määritellä erikseen ehtoa. Elsen voisi korvata toisella elif-osiolla, joka tarkastaisi, onko annettu luku pienempi kuin viisi. (Nikula & Vanhala 2010, 24–26.)

If-if-else- ja if-elif-else-rakenteet eroavat toisistaan siinä, että koodi käy läpi kaikki if-väitteet, vaikka ne saisivat arvon tosi. Elif-väite käydään läpi ainoastaan siinä tapauksessa, jos edellinen if- tai elif-väite sai arvon epätosi. Rakenteesta poistutaan heti, jos elif-väite saa arvon tosi. (Nikula & Vanhala 2010, 27–28.)

3.2.4 Toistorakenteet

Toistorakenteita on kaksi, while ja for. While on näistä avoimempi ja vapaamuotoisempi, sillä se soveltuu sekä aloitus- että loppuehtoiseen käyttöön, eli toistoehto voidaan tarkastaa ennen alkua tai lopuksi. For sisältää whilea tarkemman syntaksin, ja se toimii aina aloitusehtoisesti. For-rakenteeseen kuuluu sijoittaa joko kierrosmäärä, jonka voi määritellä range-funktiolla, tai tietomuoto, jonka alkiot käydään läpi. Toistorakenteita ei suositella pantavan sisäkkäin kolmea enempää, mikä halutaan tehdä tehokasta ohjelmakoodia. (Kasurinen 2009, 59–67.)

```
ikasi = int(input("Kuinka vanha olet? "))
toiveika = int(input("Kuinka vanha haluaisit olla? "))

while ikasi < toiveika:
    ikasi+=1
    print("vanhenit vuodella, olet nyt", ikasi,"vuotta vanha")
```

Esimerkkiohjelma toteuttaa käyttäjän toiveen ja vanhentaa tätä haluttuun ikään asti. Ohjelma siis kysyy ensin käyttäjän ikää ja toiveikää, ja kasvattaa sen jälkeen ikasi-muuttujaa while-silmukan avulla. Silmukka pyörii niin kauan kuin muuttuja *ikasi* on muuttujaa *toiveika* pienempi. Silmukan sisällä kasvatetaan ikasi-muuttujan arvoa yhdellä. Ohjelma tulostaa seuraavaa:

```
Kuinka vanha olet? 13
Kuinka vanha haluaisit olla? 18
vanhenit vuodella, olet nyt 14 vuotta vanha
vanhenit vuodella, olet nyt 15 vuotta vanha
vanhenit vuodella, olet nyt 16 vuotta vanha
vanhenit vuodella, olet nyt 17 vuotta vanha
vanhenit vuodella, olet nyt 18 vuotta vanha
```

(Kasurinen 2009, 59–67.)

Seuraava esimerkkiohjelma tulostaa numerojoukon väliltä 10–15. Numerot ovat 10, 11, 12, 13 ja 14. Yläraja eli tässä tapauksessa numero 15 jää aina ulkopuolelle. Käytyään silmukan läpi ohjelma siirtyy seuraavaan kohtaan eli kommentin tulostamiseen. (Nikula & Vanhala 2010, 35.)

```
for i in range(10, 15):
    print(i)
print("Silmukka tuli tiensä päähän.")
```

(Nikula & Vanhala 2010, 35.)

Toistorakenteita voidaan ohjata break- ja continue-käskyillä. Break-käskyllä voidaan keskeyttää toistorakenne, jos enemmälle määrälle toistoja ei olekaan enää tarvetta. Tällainen tilanne voi tulla eteen, jos etsitään esimerkiksi tiettyä numeroa isosta joukosta. Break-käskyn jälkeen ohjelman suorittaminen jatkuu seuraavalta toistorakenteen jälkeiseltä loogiselta riviltä, eli break ohittaa myös mahdollisen else-osion. Seuraavassa esimerkissä break-käsky keskeyttää toistorakenteen, kun dollareita on kymmenen. (Nikula & Vanhala 2010, 36.)

```
for i in range(0, 100):
    print(i,"$")
    if i == 10:
            print("Kymppihän riittää, kiitosta vain.")
            break
else:
   print("Lompsani on pohjaton, antaa tulla vain!")
print("Loppu!")
```

Ohjelman tulostus:

```
0 $
1 $
2 $
...
9 $
10 $

Kymppihän riittää, kiitosta vain.
Loppu!
```

(Nikula & Vanhala 2010, 36.)

Siinä missä break-käskyllä voidaan lopettaa toistorakenteen suorittaminen, määrää continue-käsky sen, että toistorakenteen loppuosa jätetään välistä ja siirrytään suoraa seuraavalle kierrokselle. Seuraavassa esimerkissä ohjelma käskee käyttäjää kirjoittamaan jotakin. Ohjelma keskeytyy, jos käyttäjä kirjoittaa sanan ”lopeta”. Mikäli käyttäjän antama sana on viisi merkkiä tai vähemmän, siirtyy ohjelma kysymään käyttäjältä uutta syötettä. Jos taas sana on yli viisi merkkiä pitkä, siirrytään silmukan sisällä ensin print-kohtaan, joka mainitsee asiasta käyttäjälle, ja vasta sen jälkeen käyttäjä saa kirjoittaa uudestaan. (Nikula & Vanhala 2010, 37.)

```
while True:
    merkkijono = input("kirjoita jotakin: ")
    if merkkijono == "lopeta":
            break
    if len(merkkijono) < 6:
            continue
    print("kirjoitit yli 5 merkkiä pitkästi")
```

Ohjelma tulostaa seuraavaa:

```
kirjoita jotakin: mattijateppo
kirjoitit yli 5 merkkiä pitkästi
kirjoita jotakin: matti
kirjoita jotakin: seppokin
kirjoitit yli 5 merkkiä pitkästi
kirjoita jotakin: lopeta
```

(Nikula & Vanhala 2010, 37.)

3.2.5 Funktiot

Funktio on koodin osa, jota voidaan käyttää lukemattomia kertoja uudestaan. Funktiot voivat esimerkiksi vastaanottaa syötteitä, suorittaa niille haluttuja toimenpiteitä ja lopulta palauttaa jonkin laskutoimituksen arvon. Pythonissa funktion alkuun tulee aina avainsana def, jonka jälkeen kirjoitetaan funktion nimi, sulut ja niiden syötteet ja kaksoispiste. (Nikula & Vanhala 2010, 40–43.)

```
def moikkaa():
   print("moikkelis moi!")
   print("Tämä tulostus tulee funktion sisältä.")

moikkaa()
print("Funktio suoritettu.")
```

Koska funktiossa ei ole omia muuttujia, voidaan sitä kutsua omalla nimellään. Ohjelman suoritus alkaa funktion kutsumisella, jonka jälkeen suoritus siirtyy funktion sisään. Kun moikkaa()-funktion kaksi tulostuslausetta on saatu suoritettua, siirtyy suoritus pääohjelman seuraavaan kohtaan eli tässä tapauksessa viimeiseen tulostuslauseen. Funktiot tulee aina esitellä ennen pääohjelmaa, koska Python lukee ohjelmakoodia ihmisten tapaan ylhäältä alas eikä se siten voi kutsua funktiota, jota ei entuudestaan tiedä. (Nikula & Vanhala 2010, 40–43.)

Funktiokutsun sulkujen sisään voidaan määritellä parametreja eli arvoja, joita funktio voi työstää. Parametrit toimivat funktion sisällä normaalien muuttujien tapaan sillä erotuksella, että ne tulee määritellä funktiokutsussa funktion sisäosan sijaan. Parametrit erotellaan toisistaan pilkulla. (Nikula & Vanhala 2010, 40–43.)

```
def moikkaa(nimi, kenka):
   print("Moi " + nimi + "!")
   print("kengännumerosi on",kenka)
moikkaa("Irmeli",39)
```

Muuttujilla voi kutsua myös suoraan:

```
a = "Jorma-Kalevi"
b = 48
moikkaa(a,b)
```

Tulos:

```
Moi Jorma-Kalevi!
kengännumerosi on 48
```

(Nikula & Vanhala 2010, 40–43.)

Funktioiden käyttö ei kuitenkaan ole pakollista. Suurempien ohjelmien kohdalla funktiottomuus toki asettaisi rajoitteita mm. uudelleenkäytettävyyden suhteen. Pääfunktion voi nimetä vapaasti, mutta ”main”-nimen käyttö on järkevää ainakin silloin, jos koodia on tarkoitus esitellä henkilöille, jotka eivät tunne täysin kielen syntaksia. Pääfunktion hyödyllisyydestä Python-ohjelmoinnissa ollaan yleensä kahta mieltä. Toisten mielestä pääfunktion käyttö vain tekee koodista tarpeettoman monimutkaista, sillä tarkoituksena oli kehittää ohjelmointikieli ”yksinkertainen on paras” -periaatteella. Monissa oppimateriaaleissa ei nykyisin enää käytetäkään pääfunktiota. (Kasurinen 2009, 86–106.)

3.2.6 Tiedostojen käsittely

Ohjelmoitaessa tulee monesti tarve tallentaa esimerkiksi muuttujien arvot tai ohjelman asetukset omaan tiedostoonsa, jotta niitä ei tarvitse luoda jokaisella käynnistyskerralla uudestaan. Python-tulkki käsittelee tiedostojen sisältämää dataa kuten merkkijonoja. (Nikula & Vanhala 2010, 51.)

```
teksti = '''\
Perjantaina 24.9.2010

Rakas päiväkirja! Tänään tapahtui vaikka mitä...'''
tiedosto = open("paivakirja.txt", "w", encoding="utf-8")
tiedosto.write(teksti)
tiedosto.close()
tiedosto = open("paivakirja.txt", "r", encoding="utf-8")
while True:
    rivi = tiedosto.readline()
    if len(rivi) == 0:
      break
    print(rivi, end = "")
tiedosto.close()
```

Edellä esitetty ohjelma kirjoittaa tekstiä paivakirja-nimiseen tekstitiedostoon, joka sijaitsee samassa kansiossa lähdekooditiedoston kanssa. Aluksi luodaan teksti-muuttuja, johon tallennetaan haluttu päiväkirjamerkintä merkkijonomuodossa. Pitkän tekstin tallennukseen on hyvä käyttää kolmen heittomerkin syntaksia. Tiedosto avataan käskyllä open(), jonka sulkujen sisään laitetaan kolme parametria: tiedoston nimi, tila johon tiedosto avataan, tässä tapauksessa siis kirjoitustila write eli w, ja viimeisenä parametrina merkistökoodaus. Write()-komennolla kirjoitetaan teksti-muuttujan sisältämä merkkijono tiedosto-muuttujaan. Tiedosto suljetaan close()-komennolla. Sulkeminen varmistaa sen, että tiedoston lukeminen ja siihen kirjoittaminen eivät onnistu, ennen kuin tiedosto avataan uudelleen. Tiedostoon voidaan kirjoittaa ainoastaan merkkijono-tyyppistä tietoa, eli numerotietoa tallennettaessa tulee luvut muuntaa merkkijonoiksi str()-tyyppimuunnosfunktiolla. (Nikula & Vanhala 2010, 51–54.)

Tiedosto avataan uudelleen lukutilassa moodilla r (read) ja se luetaan rivi riviltä readline()-funktiolla, joka on sijoitettu while-silmukan sisään. Funktion palauttama merkkijono vastaa tiedoston yhtä fyysistä riviä. Silmukasta poistutaan, kun len()-funktio saa arvon 0, mikä tarkoittaa tyhjää riviä. (Nikula & Vanhala 2010, 51–54.)

Tiedostojen lukemiseen on olemassa myös muita funktioita. Read() palauttaa niin ikään tiedoston sisällön merkkijonona. Sulkujen sisään voidaan asettaa kokonaisluku, joka vastaa luettavien merkkien määrää, ja jos sulut jättää tyhjäksi, lukee funktio koko tiedoston. Readlines() palauttaa tiedoston sisällön yhtenä listana rivinvaihtomerkit mukaan lukien:

> ['Perjantaina 24.9.2010\n', '\n', 'Rakas päiväkirja! Tänään tapahtui vaikka mitä...']

(Nikula & Vanhala 2010, 54–55.)

3.2.7 Tietorakenteet

Pythonin sarjallisia muuttujatyyppejä ovat mm. lista, monikko (tuple), luokka ja sanakirja (dictionary). Listaa voidaan muokata jäsenfunktioiden eli metodien avulla, joiden käyttöä demonstroidaan seuraavassa esimerkissä:

```
ostoslista = ["maito", "leipä", "perunat", "kahvi"]

print("Listallani on", len(ostoslista), "tuotetta:")

for tuote in ostoslista:
   print(tuote, end = "\n")

print("\nUnohdin listasta omenat.\n")
ostoslista.append("omenat")
print("Nyt ostoslista näyttää tältä", ostoslista)
print("\nJärjestellään ostoslista:")
ostoslista.sort()

print(ostoslista)

print("\nEnsimmäisenä ostan tuotteen", ostoslista[0])
del ostoslista[0]

print("Nyt ostoslistalla on jäljellä", ostoslista)
```

Tulostus:

```
Listallani on 4 tuotetta:
maito
leipä
perunat
kahvi

Unohdin listasta omenat.

Nyt ostoslista näyttää tältä ['maito', 'leipä', 'perunat', 'kahvi', 'omenat']

Järjestellään ostoslista:
['kahvi', 'leipä', 'maito', 'omenat', 'perunat']

Ensimmäisenä ostan tuotteen kahvi
Nyt ostoslistalla on jäljellä ['leipä', 'maito', 'omenat', 'perunat']
```

Lista luodaan samaan tapaan kuten normaali muuttuja, mutta alkiot laitetaan hakasulkujen sisään ja ne erotellaan toisistaan pilkulla. Listan alkiot voidaan tulostaa normaalilla print-komennolla tai käyttää erikseen silmukkaa, kuten tässä tapauksessa for-silmukkaa ja range-funktiota. Alkioita voidaan lisätä metodilla append ja listan alkiot voidaan panna aakkosjärjestykseen metodilla sort. Del-metodilla voidaan poistaa alkioita järjestysnumeron perusteella. Sarjallisissa muuttujissa numerointi alkaa aina nollasta, eli poistamalla listasta alkion 0 poistetaan tässä tapauksessa aakkosjärjestyksen ensimmäinen alkio eli kahvi. Pois-

tettu alkio ei jää tyhjäksi vaan sen paikan ottaa järjestyksessä seuraava alkio jne. (Nikula & Vanhala 2010, 63–65.)

Edellä mainittujen metodien lisäksi on olemassa tukku muita metodeita, joista maininnan arvoisia ovat ainakin extend, insert, remove, pop, index, count ja reverse. Extend nimensä mukaisesti laajentaa olemassa olevaa listaa toisen listan alkioilla. Alkiot lisätään listan perälle. Insert-metodilla voidaan lisätä alkio tiettyyn listan kohtaan. Removella voidaan poista listasta alkioita arvon perusteella; metodi poistaa ensimmäisen täsmäävän alkion. Pop poistaa listasta alkion järjestysnumeron perusteella ja samalla palauttaa sen arvon. Index kertoo, missä kohden listaa etsitty arvo on. Count kertoo, kuinka monta kertaa etsitty arvo esiintyy listalla, ja reverse kääntää listan järjestyksen toisinpäin. (Nikula & Vanhala 2010, 66–67.)

Monikko-tietotyyppi on ulkoisilta ominaisuuksiltaan lähes samanlainen listan kanssa, mutta rakenteeltaan ne eroavat siinä, että Monikko on staattinen. Monikossa ei siis ole metodeja, eikä sitä voida muokata enää määrittelyn jälkeen. Monikko toimii siis ikään kuin kirjoitussuojattuna listana, jota voi hyödyntää esimerkiksi käsiteltäessä kuvatietoja tai koordinaatteja, joita ei haluta vahingossa hukattavan. Monikon käsittelyssä käytetään kaarisulkuja. (Kasurinen 2009, 131–138.)

```
arkki = ("opossumi", "rotta", "kurki")
print("Elukoita Nooan arkissa", len(arkki),"kpl")
uusi_arkki = ("delfiini", "tiikeri", arkki)
print("Uudessa arkissa on", len(uusi_arkki),"eläintä.")
print("Uuden arkin kaikki eläimet:", uusi_arkki)
print("Vanhasta eläintarhasta tuotuja ovat:", uusi_arkki[2])
print("Uuden eläintarhan viimeinen eläin on", uusi_arkki[2][2])
```

Tulostus:

```
Elukoita Nooan arkissa 3 kpl
Uudessa arkissa on 3 eläintä.
Uuden arkin kaikki eläimet: ('delfiini', 'tiikeri', ('opossumi', 'rotta', 'kurki'))
Vanhasta eläintarhasta tuotuja ovat: ('opossumi', 'rotta', 'kurki')
Uuden eläintarhan viimeinen eläin on kurki
```

(Nikula & Vanhala 2010, 72–73.)

Edellä esitetyssä esimerkissä luodaan ensin arkki-niminen monikko, johon tallennetaan kolme eläintä. Sen jälkeen luodaan toinen monikko nimeltä uusi_arkki, johon luodaan kaksi uutta eläintä sekä sisällytetään arkki-monikko. Arkki siis asettuu uusi_arkki-monikon kolmanneksi (järjestysnumeroltaan toiseksi) alkioksi, mikä käy ilmi tulostuksesta. Monikon alkioita voidaan käsitellä samaan tapaan kuten listan alkioita. Esimerkkiohjelman viimeisellä rivillä tulostetaankin uusi_arkki:n kolmannen alkion kolmas alkio. (Nikula & Vanhala 2010, 72–73.) Luokka on rakenteinen tietotyyppi, eli käyttäjä saa määritellä alkioita koskevat ominaisuudet kuten sen nimen, määrän ja laadun. Seuraavassa esimerkissä toteutetaan luokan luominen ja periyttäminen:

```
class Ihminen:
  etunimi = ""
  sukunimi = ""
  ika = 0
  ammatti = ""

class Supersankari(Ihminen):
    sankarinimi = ""
    erikoiskyky = ""

masav = Ihminen()
masav.etunimi = "Matti"
masav.sukunimi = "Virtanen"
masav.ika = 43
masav.ammatti = "sekatyömies"

hero1 = Supersankari()
hero1.etunimi = "Bruce"
hero1.sukunimi = "Banner"
hero1.ika = 35
hero1.ammatti = "tiedemies"
hero1.sankarinimi = "Hulk"
hero1.erikoiskyky = "valtava fyysinen voima"

print(masav.etunimi,masav.sukunimi,"on",
masav.ika,"vuotta vanha",masav.ammatti)

print("\nSupersankari",hero1.sankarinimi, "on henkilön", hero1.etunimi,
hero1.sukunimi, "alter ego","\nHahmon ikä on",hero1.ika,"vuotta",
"ja siviiliammatti",hero1.ammatti,"\nSupersankarin erikoiskyky on",hero1.erikoiskyky,end=".")
```

Tulostus:

```
Matti Virtanen on 43 vuotta vanha sekatyömies
Supersankari Hulk on henkilön Bruce Banner alter ego
Hahmon ikä on 35 vuotta ja siviiliammatti tiedemies
Supersankarin erikoiskyky on valtava fyysinen voima.
```

Aluksi luodaan luokka Ihminen, jolla on neljä jäsenmuuttujaa. Seuraavaksi luodaan luokka Supersankari, joka perii Ihminen-luokan ominaisuudet eli tässä tapauksessa neljä muuttujaa. Lisäksi Supersankari-luokalla on kaksi omaa jäsenmuuttujaa. Perittävän luokan nimi laitetaan uuden luokan sulkujen sisään. Uusi olio luodaan kaavalla olion nimi, yhtäsuuruusmerkki ja luokan nimi. Esimerkissä luodaan masav-niminen olio, jolle asetetaan arvot kaikkiin muuttujiin pistenotaatiolla. Tämän jälkeen luodaan hero1-niminen olio, joka koostuu kuudesta jäsenmuuttujasta. (Nikula & Vanhala 2010, 69–72.)

Sanakirja eroaa listasta siten, että siinä voidaan ”nimetä” alkioita määrittelemälle niille vastinpareja. Listan metodeita ei voi käyttää sanakirjan kanssa. Sanakirja pääsee oikeuksiinsa esimerkiksi silloin, kun halutaan tallentaa käyttäjätunnus ja sitä vastaava salasana tai (vaikka) ohjelman asetukset (tallennuksessa). (Kasurinen 2009, 131–138.)

```
legendat = {
"22" : "Dino Ciccarelli",
"66" : "Mario Lemieux",
"99" : "Wayne Gretzky",}

print("Numerolla 22 pelasi", legendat["22"])
legendat["17"] = "Jari Kurri"
del legendat["66"]
print("\nLegendojen listalla on",len(legendat),"merkintää.\n")
for avain, arvo in legendat.items():
                print("Pelinumeron {0} teki kuuluisaksi pelaaja nimeltä
{1}".format(avain,arvo))
if "17" in legendat:
    print("\nAvainta 17 vastaa arvo {0}".format(legendat["17"]))
```

Tulostus:

```
Legendojen listalla on 3 merkintää.

Pelinumeron 99 teki kuuluisaksi pelaaja nimeltä Wayne Gretzky
Pelinumeron 17 teki kuuluisaksi pelaaja nimeltä Jari Kurri
Pelinumeron 22 teki kuuluisaksi pelaaja nimeltä Dino Ciccarelli

Avainta 17 vastaa arvo Jari Kurri
```

(Nikula & Vanhala 2010, 74–75.)

Edellä esitetyssä esimerkissä luodaan legendat-niminen sanakirja, johon tallennetaan kolme avainta ja avainta vastaavaa arvoa. Avaimena toimii pelinumero ja arvona pelaajan nimi. Avain ja arvo erotetaan toisistaan kaksoispisteellä ja ne määritellään aaltosulkeiden sisään. Sanakirjasta voi hakea tietoa avaimen, mutta ei arvon perusteella. Ohjelmassa tulostetaan ensin arvo, joka vastaa avainta 22. Sen jälkeen lisätään ja poistetaan yksi uusi avain ja sitä vastaava arvo. Esimerkkiohjelman for-silmukka tulostaa kaikki avaimet ja niitä vastaavat arvot. Silmukan tulostuslauseessa loppuosassa käytetään metodia format(), jonka sulkujen sisään on määritelty, miten avaimet ja arvot halutaan tulostaa. Tässä esimerkissä halutaan tulostaa ensin avain ja sitä vastaa aaltosulkeiden sisällä oleva nolla. Arvoa puolestaan vastaa aaltosulkeiden sisässä oleva ykkönen. Esimerkkiohjelman lopussa on if-lause, joka tulostaa avainta 17 vastaavaan arvon, mikäli sellainen avain on olemassa. (Nikula & Vanhala 2010, 74–75.)

3.2.8 Kirjastot ja moduulit

Moduuli on funktioita sisältävä lähdekooditiedosto, joka voidaan ottaa käyttöön sisällyttämällä se koodiin. Pythonin mukana tulee suuri joukko valmiita moduuleita (ns. kirjastomoduuleita) eri käyttötarkoituksiin, mutta mikään ei tietenkään estä ohjelmoijaa tekemästä itse omia moduuleitaan. Moduulin sisällyttäminen tapahtuu import-komennolla. (Nikula & Vanhala 2010, 78.)

```
import math
print("Tehdään muutama korkeampaa matematiikkaa vaativa operaatio:")
print("e toiseen potenssiin on", math.exp(2))
print("Neliöjuuri 10:stä on", math.sqrt(10))
print("sin(2) radiaaneina on", math.sin(2))
print("Piin likiarvo on", math.pi)
```

Tulostus:

```
Tehdään muutama korkeampaa matematiikkaa vaativa operaatio:
e toiseen potenssiin on 7.38905609893
Neliöjuuri 10:stä on 3.16227766017
sin(2) radiaaneina on 0.909297426826
Piin likiarvo on 3.14159265359
```

(Nikula & Vanhala 2010, 78.)

Edellä esitetyssä esimerkkiohjelmassa käytetään math-moduulia, joka sisältää useita matemaattisia funktioita. Esimerkkiohjelmassa hyödynnetään funktioita exp(), sqrt(), sin() ja vakiota pi. Hyödyntäminen tapahtuu siten, että ensin kirjoitetaan moduulin nimi, sitten piste ja lopuksi käytettävän funktion nimi ja sen sulkujen sisään parametrit. Tätä kutsutaan pistenotaatioksi. Pistenotaation käyttö voi kuitenkin tulla pidemmän päälle työlääksi, ja siksi on olemassa myös toinen tapa, käsky ”from x import y”, jossa x tarkoittaa moduulia ja y moduulin funktiota. Math-moduulin sin-funktio voitaisiin siis sisällyttää koodiin komennolla from math import sin. Komennolla from x import * voidaan puolestaan tuoda kaikki moduulin funktiot ja vakiomuuttujat. Tätä tapaa on kuitenkin syytä käyttää harkitusti. Mikäli kaksi moduulia sisältää samannimisen funktion, jälkimmäinen funktio ylikirjoittaa ensimmäisen funktion eikä ylikirjoitettua funktiota voi enää jälkikäteen muokata. (Nikula & Vanhala 2010, 78–79.)

Pythonin nopeutta voidaan parantaa luomalla esikäännettyjä tiedostoja. Normaalisti Python-tulkki ensin kääntää lähdekooditiedoston konekielelle ja vasta sitten suorittaa ohjelman. Tämä tapahtuu joka kerta, kun lähdekooditiedostoa halutaan suorittaa. Esikäännetty tiedosto on siis käännetty konekieliseen muotoon jo valmiiksi, ja sen voi suorittaa saman tien. Esikäännetty Python-tiedosto merkitään pyc-tiedostopäätteellä. Pyc-tiedostoa ei voi enää jälkikäteen muuttaa. Esikäännetty tiedosto voidaan sisällyttää koodiin samalla tavalla kuten moduulit. (Nikula & Vanhala 2010, 79.)

```
def terve():
   print("Tämä tulostus tulee moduulin omaModuuli funktiosta 'terve'.")
def summaa(luku1, luku2):
   tulos = luku1 + luku2
   print("Laskin yhteen luvut", luku1, "ja", luku2)
   return tulos
```

Edellä esitetyssä esimerkissä luodaan ensin moduuli nimeltä omaModuuli.py. OmaModuuli sisältää kaksi funktiota, joista ensimmäinen tulostaa tervehdyksen ja jälkimmäinen laskee kaksi lukua yhteen. Sen jälkeen luodaan moduulinKaytto.py-ohjelma, johon omaModuuli sisällytetään import-käskyllä:

```
import omaModuuli

omaModuuli.terve()
eka = 5
toka = 6
yhdessa = omaModuuli.summaa(eka,toka)
print("eka ja toka ovat yhteensä", yhdessa)
```

OmaModuulin terve-funktiota kutsutaan ilman parametreja. Summaa-funktiolle annetaan kaksi parametria, eka- ja toka-muuttujat, ja tulos tallennetaan yhdessa-muuttujaan, joka tulostetaan seuraavalla rivillä. (Nikula & Vanhala 2010, 81–82.)

Kirjastomoduuleista mainitsemisen arvoisia ovat math-moduulin lisäksi random, datetime, time, urllib ja fractions. Random:in avulla voidaan generoida satunnaislukuja, ja datetimella napataan järjestelmän kellonaika- ja päivämäärätietoja ja tehdään niille laskutoimituksia. Time-moduulin avulla voidaan mitata esimerkiksi ohjelman suorittamiseen kuluvaa aikaa. Urllibillä voidaan lukea internetsivuja ja fractions-moduulilla puolestaan käsitellään murtolukuja. Kirjastomoduuliluettelo löytyy Pythonin internetsivuilta. (Nikula & Vanhala 2010, 82–85.)

3.2.9 Virheenkäsittely

Pythonissa virheen kiinni ottaminen ja virheestä toipuminen tapahtuu try-except-rakenteen avulla. Try-osioon määritellään virheille oletettavasti altis koodi ja except-osioon käsittelytapahtumat kiinni otettaville virheluokille. Virheluokkia on 35 erilaista, ja erilaisia varoi-

tuksia on yhdeksän. Tulkki käy ohjelmakoodin läpi normaaliin tapaan. Päästyään virheenkäsittelijän alkuun tulkki alkaa suorittaa virheenkäsittelijän sisään sijoitettua koodia. Mikäli kyseinen koodi ei aiheuta ongelmia, virheenkäsittelijää ei suoriteta ja ohjelma jatkuu normaalisti. Mikäli try-osion sisällä sattuu virhe, tulkki siirtyy kyseiselle virheelle luotuun virheenkäsittelijään ja try-osiossa olevan koodin suoritus keskeytyy. Samalla tulkki suorittaa except-osion koodia ja jatkaa taas try-osiosta seuraavalta riviltä. Jos virhetyypille ei ole määritelty except-osiota, tulkki antaa virheilmoituksen ja ohjelman suoritus loppuu samaan tapaan kuin sellaisen ohjelman, jolle ei ole tehty virheenkäsittelyrakennetta. (Kasurinen 2009, 110–118.)

Seuraavassa esimerkissä käytetään ValueError-virheluokkaa, joka keskeyttää ohjelman, jos käyttäjän antama vastaus ei ole kokonaisluku. Mikäli kyseistä virheluokkaa ei käytetä, except-osio keskeyttää ohjelman, oli virheenä mikä tahansa, ja tämä saattaa johtaa siihen, että ohjelmakoodissa oleva virhe jää ohjelmoijalta huomaamatta. Virheluokkia tulisi siis käyttää aina kun mahdollista. (Nikula & Vanhala 2010, 86–88.)

```
try:
    luku = int(input("Anna kokonaisluku: "))
    print("Annoit kokonaisluvun", luku)
except ValueError:
    print("Et antanut kunnollista kokonaislukua.")
```

Samaan except-osioon voidaan yhdistellä useita virheluokkia, mikäli niillä on yhteinen toipumismekanismi:

```
except (ValueError, TypeError):

    print("Jottain meni pielleen")
```

(Nikula & Vanhala 2010, 86–88.)

Siinä missä try-except-rakenteen tavoitteena oli estää kaatuminen, try-finally-rakenne huolehtii siitä, että ohjelma saadaan lopetettua hallitusti ja että dataa ei pääsisi häviämään. Finally-osioon kirjoitetut käskyt suoritetaan aina, keskeytyi ohjelma sitten tai ei. Kyseiset käskyt voivat esimerkiksi tallentaa ja sulkea keskeneräiset tiedostot ja katkaista verkkoyhteyden. (Kasurinen 2009, 110–118.)

Seuraava esimerkkiesiohjelma avaa tiedoston uutinen.txt ja lukee ja tulostaa sen rivi kerrallaan. Jokaisen lukukerran jälkeen ohjelma pysähtyy 2 sekunnin ajaksi, kunnes kaikki rivit on tulostettu. Finally-osiossa tiedosto suljetaan oikeaoppisesti, jotta se olisi taas muiden ohjelmien käytettävissä. (Nikula & Vanhala 2010, 89–90.)

```
import sys,time
tiedosto = open("uutinen.txt", "r", encoding="utf-8")
try:
    while True:
            rivi = tiedosto.readline()
            if len(rivi) == 0:
                    break
            time.sleep(2)
            print(rivi, end="")
finally:
    print("***Suljetaan tiedosto***")
    tiedosto.close()
```

Tulostus:

```
Esa Pakarinen
Reino Helismaa
Jorma Ikävalko
***Suljetaan tiedosto***
```

(Nikula & Vanhala 2010, 89–90.)

Pythonin with-avainsanan avulla saadaan virheenkäsittelyyn lisää selkeyttä. Seuraavassa esimerkissä try-osion sisään laitettu with-lohko ensin avaa tiedoston, lukee sen rivi kerrallaan ja tallentaa kunkin rivin sisällön listaan, jonka jälkeen tiedosto suljetaan automaattisesti ilman close-komentoa, tapahtui sitten virhe tai ei. Mikäli tiedostoa ei löydy, kutsutaan IOError-virheluokkaa ja ohjelma keskeytyy. (Nikula & Vanhala 2010, 90–91.)

```
import sys
lista = []
try:
    with open("luvut.txt", "r", encoding="utf-8") as tiedosto:
            while True:
                    rivi = tiedosto.readline()
                    if len(rivi) == 0:
                            break
                    rivi = rivi[0:-1]
                    lista.append(int(rivi))
                    print(lista)
except IOError:
   print("Tiedostoa ei voitu avata.")
   sys.exit(-1)
print("Tiedoston käsittely suoritettu onnistuneesti.")
print("Kiitos ohjelman käytöstä.")
```

Tulostus:

```
[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
Tiedoston käsittely suoritettu onnistuneesti.
Kiitos ohjelman käytöstä.
```

(Nikula & Vanhala 2010, 90–91.)

3.2.10 Graafinen käyttöliittymä

Pythonille on saatavissa enemmän käyttöliittymäkirjastoja kuin millekään muulle ohjelmointikielelle. Eräs näistä kirjastoista on Tkinter, joka kuuluu Pythonin vakiopaketteihin. Tkinter ei pohjaudu Pythoniin, vaan Tcl-kielen Tk-käyttöliittymätyökaluihin. Tkinterin avulla graafisten käyttöliittymien luonti on varsin helppoa.

Tulkin ikkunassa ajettavista ohjelmista poiketen Tkinter perustuu widget-komponentteihin ja niiden tapahtumapohjaiseen toimintaan. Tkinter-käyttöliittymä koostuu peruskomponenteista, kuten painikkeista, ikkunoista, syöte- ja tekstikentistä sekä manuaalisesti luoduista valikoista. (Kasurinen 2009, 200.)

Tapahtumapohjaisessa toiminnassa ohjelma odottaa, että käyttäjä ensin tekee jotain, kuten esimerkiksi painaa painiketta, minkä jälkeen kyseiseen komponenttiin kytketty tapahtuma käynnistyy. Ohjelman suoritus loppuu vasta lopettamispainikkeen painamiseen tai käyttöjärjestelmän määräyksestä. Lähdekoodissa käyttöliittymä ja sen widgetit määritellään päätasolle ja toiminnat toteutetaan funktioina. Toimiakseen Tkinter-kirjasto vaatii, että jokaiselle peruskomponentteihin nimetylle tapahtumalle on oltava funktio. Tämä voidaan kiertää luomalla ns. dummy-funktioita, jotka eivät tee oikeasti mitään. Käyttöliittymää luotaessa pitää ensin ottaa käyttöön Tkinter-moduuli:

```
from tkinter import *
```

Sen jälkeen luodaan pohjaikkuna-komponentti:

```
ikkuna = Tk()
```

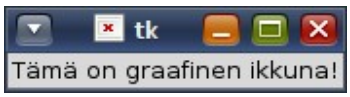
Pohjakomponentin päälle voidaan luoda uusia komponentteja, kuten painikkeita, tekstikenttiä ja valikoita. Seuraavassa on yksinkertaisen graafisen ohjelman lähdekoodi:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from tkinter import *

ikkuna = Tk()
tekstiruutu = Label(ikkuna, text = "Tämä on graafinen ikkuna!")
tekstiruutu.pack()
ikkuna.mainloop()
```

Ohjelmassa Label-komponentti nimeltä tekstiruutu on sijoitettu pohjaikkunaan ikkuna. Komponentin jäsenfunktion pack kutsuminen ilmoittaa tulkille, että komponentin määrittely on valmis. ”Pakkaamattomia” komponentteja ei piirretä ruudulle. Ohjelma käynnistetään mainloop-komennolla. Tuloksena syntyy pieni ikkuna (KUVIO 2).



KUVIO 2. Graafinen ikkuna

Käyttöjärjestelmän vastuulle jäävät muut toiminnot, kuten ikkunan koon muuttaminen ja ikkunan ulkoasu. Sekä pääikkuna että koko ohjelma voidaan lopettaa painamalla ”Sulje”-painiketta. (Kasurinen 2009, 200–216.)

4 TIETOKONEPELIT

4.1 Mikä on tietokonepeli?

Tietokonepeli käyttää tavallista tietokoneohjelmaa enemmän hyväksi tiedon syöttämiseen ja vastaanottamiseen tarkoitettuja laitteita, kuten esimerkiksi näppäimistöä, hiirtä, peliohjainta tai vaikka kuulokemikrofonia. Lisäksi pelit toimivat muita ohjelmia nopeammin ja vaativat enemmän suoritinaikaa. Tietokonepeliä voisi vaikka verrata tekstinkäsittelyohjelmaan, jolla kirjoitetaan tekstiä, muokataan tekstin ulkoasua sekä lisätään kommentteja ja lopulta teksti tallennetaan omaan tiedostoonsa. Pelissä sen sijaan on päivitettävä näyttöruutu niin monta kertaa sekunnissa kuin mahdollista, vihollisten pitää reagoida pelihahmon liikkeisiin, pelaamisen päätavoitteen eli maalin on oltava saavutettavissa ja mikä tärkeintä, pelaamisen täytyy olla hauskaa. (Baker, Campbell & Collins 2002, 2.)

Pelin tärkein ominaisuus on tietysti pelattavuus. Hyvän pelattavuuden peli on sellainen, jossa pelaaja tuntee pystyvänsä vaikuttamaan asioihin ja saavuttamaan pelin tavoitteen ja jonka pelaaminen on mukavaa. Huono pelattavuus voisi koostua esimerkiksi seuraavista tekijöistä: pelihahmo ei tottele pelaajan käskyjä, tekoäly tekee epäloogisia ratkaisuja, peli kaatuu tai peli sisältää sellaisia virheitä, joiden takia pelin tavoitetta ei voi saavuttaa, kuten esimerkiksi loppuvastustajaa ei voi kukistaa millään keinolla. (Baker ym. 2002, 3–5.)

Pelin käyttöliittymän tulisi olla nopea ja selkeä käyttää, eli pelaajan tulisi päästä hyvin vaivattomasti pelaamaan. Pelin tulisi olla ”helppo oppia, mutta vaikea hallita”, eli pelaaja oppisi suhteellisen nopeasti liikuttamaan pelihahmoa ja etenemään pelissä, mutta ylimääräisten tasojen avaaminen tai erikoistemppujen tekeminen vaatisi jo enemmän harjoittelua ja vaivannäköä. Kyseisten ominaisuuksien pitäisikin olla riittävän palkitsevia, jotta pelaaja todella haluaa nähdä vaivaa niiden saavuttamiseen. Pelin kontrollien, eli pelihahmon liikkeiden ohjaamisen eri näppäimillä, tulisi olla määritelty siten, että näppäimiä painaakseen pelaajan ei tarvitse jatkuvasti muuttaa asentoaan tai otettaan peliohjaimesta. (Baker ym. 2002, 3–5.)

Myös juonella voi olla iso merkitys pelin menestyksen kannalta. Juoni tarkoittaa pelin tapahtumien kulkua, eli sitä mitä tehdään, miksi tehdään ja miten. Esimerkiksi seikkailupe-

lissä taustatarinana voisi olla pelihahmon isän murha, joka pelaajan tulee sitten kostaa. Kaikissa peleissä juonta ei tietenkään tarvita. Esimerkiksi urheilupeleissä pelaaja saa vapaasti valita pelimuodot ja joukkueet eikä varsinaista ennalta määrättyä tavoitetta ole. (Baker ym. 2002, 5–6.)

4.2 Pelityypit

Tietokonepelit voidaan luokitella seuraaviin pääkategorioihin:

Räiskintäpeleihin (Shooter) kuuluvat sekä ensimmäisen (First-person shooter, FPS) että kolmannen (Third-person shooter, TPS) persoonan räiskinnät ja esimerkiksi sivusta vieritettävät ammuskelut. FPS- ja TPS-peleissä on realistista 3D-grafiikkaa, joka vaatii tehokasta tietokonetta pyöriäkseen sujuvasti. Räiskintäpeleissä pääpaino on ammuskelulla ja juoni jää monesti sivurooliin. FPS-peleissä pelitapahtumat kuvataan pelihahmon silmistä käsin ja ruudulla näkyy yleensä vain hahmon käyttämä ase ja esimerkiksi energiatason kuvaaja. TPS-peleissä kamera seuraa pelitapahtumia hahmon takaa siten, että hahmon tekemät liikkeet ovat selvästi nähtävissä. (Baker ym. 2002, 2–5.)

Strategiapelit voidaan luokitella kahteen ryhmään: reaaliaikaiset strategiat (Real-time strategy, RTS) ja vuoropohjaiset strategiat (Turn-based strategy, TBS). Reaaliaikaisissa strategiapeleissä kaksi tai useampaa puolta ottaa mittaan toisistaan reaaliajassa, eli kukin pelin osapuoli pystyy liikuttamaan joukkoja ja keräämään resursseja samanaikaisesti, aivan kuten oikeassakin elämässä. RTS-peleissä pelaaja joutuukin ohjaamaan monta asiaa yhtä aikaa, koska peliä ei voi pysäyttää tarkastellakseen tilannetta yläilmoista käsin. Vuoropohjaisissa strategiapeleissä siirrot tehdään vuoron perään kuten shakissa ja aikaa on käytössä miltei rajattomasti. (Baker ym. 2002, 2–5.)

Simulaatioissa pyritään mallintamaan tosielämää mahdollisimman tarkasti. Lentosimulaattorilla lentäminen voi olla ihan yhtä haastavaa kuin oikeankin koneen ohjaaminen. Pilkkisimulaatiossa kalaparvet käyttäytyvät realistisesti ja saantiin vaikuttavat useat eri asiat, kuten vaikkapa sää ja valoisuus. (Baker ym. 2002, 2–5.)

Tasohyppelypeleissä (Platformer) pelihahmoa ohjataan pitkin tasoja, jotka sisältävät erinäisiä esteitä, kuten esimerkiksi kuoppia, joihin putoamalla pelihahmo menettää energiaa tai jopa henkensä. Peli on yleensä jaettu useisiin kenttiin, jotka läpäisemällä pääsee seuraavaan kenttään. Viimeisessä kentässä vastassa saattaa olla loppuvihollinen, jonka kukistamalla peli läpäistään. 80- ja 90-luvulla tasohyppelyt olivat vielä pääosin kaksiulotteisia ja sivusuunnassa eteneviä, mutta nykyään tasohyppelypelit ovat jo yleensä kolmiulotteisia ja niissä eteneminen on vapaampaa. (Baker ym. 2002, 2–5.)

Puzzlepeleissä pelaajan on tarkoitus ratkoa erinäisiä ongelmia, kuten esimerkiksi koota pyramidi useasta erimuotoisesta kappaleesta. Puzzlepeleissä näyttävä peligrafiikka on sivuroolissa. (Baker ym. 2002, 2–5.)

Seikkailupeleissä (Adventure) on yleensä ennalta hyvin tarkkaan määritelty juoni. Pelaajan ohjaama hahmo joutuu selvittämään erinäisiä mysteereitä, kuten esimerkiksi henkilön katoamista. Pelissä edetään tutkimalla ympäristöä ja keskustelemalla peliin kirjoitettujen hahmojen kanssa. (Baker ym. 2002, 2–5.)

Roolipelit (Role-playing games, RPG) eroavat seikkailupeleistä siten, että niissä pelihahmoa voi yleensä muokata hyvinkin paljon mielensä mukaan ja hahmolle kertyy kokemuspisteitä suoritettujen pientehtävien mukaan. (Baker ym. 2002, 2–5.)

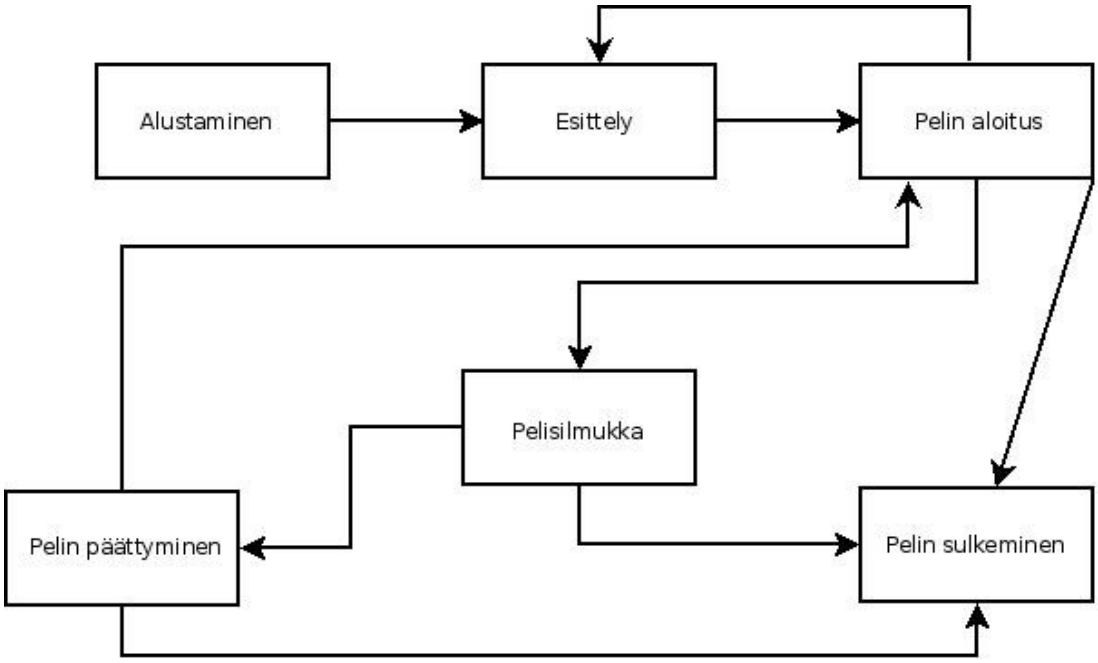
Korttipeleihin voidaan laskea sekä pienet ajantappopelit, kuten pasianssi, että verkossa pelattavat uhkapelit, kuten Texas Hold'em. Pasianssin kaltaisissa yksinpeleissä tietokone hoitaa korttien jakamisen ja pisteen laskun ja pelaajan harteille jää pelkkä korttien siirtely. (Baker ym. 2002, 2–5.)

Urheilupelit voivat olla hyvin lähellä simulaatioita, eli pelitapahtumat ja hahmojen käyttäytyminen on mallinnettu hyvin tarkasti tosielämän mukaan. Yleensä kuitenkin urheilupelien kehittämisessä pyritään tasapainoon realismin ja pelattavuuden suhteen eli tekemään pelejä, jotka on helppo oppia mutta vaikea hallita. Esimerkiksi jalkapallopelissä pelaaja voi halutessaan ohjata koko pelin ajan yhtä tiettyä kentällä olevaa pelaajaa tai perinteiseen tapaan sitä pelaajaa, jolla on pallo tai joka on lähimpänä vastapuolen pallollista pelaajaa. (Baker ym. 2002, 2–5.)

4.3 Pelin rakenne

4.3.1 Moduulit

Tietokonepeli voidaan jakaa rakenteeltaan kuuteen moduuliin (KUVIO 3). Alustaminen- ja esittely-moduulit vastaavat peliohjelman käynnistymisestä. Ne suoritetaan vain kerran. Pelin aloitus -moduuliin kuuluu esimerkiksi peliasetusten, kuten vaikeustason, säätö. Pelisilmukka vastaa pelilogiikan suorittamisesta. Pelin päättyminen -moduuli antaa pelaajalle palautteen pelin tuloksesta, ja siitä siirrytään takaisin esittely-moduuliin. Peli lopetetaan kutsumalla Pelin sulkeminen -moduulia. (Rautiainen 2002, 32.)



KUVIO 3. Pelin rakenne (Rautiainen 2002, 32.)

- Alustaminen vastaa pelin asettamisesta alkutilaan. Sen vastuulla on muistin varaaminen, globaalien muuttujien alustaminen, hakutaulukoiden luonti ja esimerkiksi grafiikka- ja äänitiedostojen lataaminen. Alustamisen aikana ruudulla pyörii yleensä alkuintro pelaajaa viihdyttämässä. Pelkkä tyhjä ruutu voisi saada pelaajan luulemaan, että peli on kaatunut. (Rautiainen 2002, 32–34.)

- Esittely johdattaa pelaajan pelin maailmaan valottamalla taustatarinaa sekä opastamalla pelihahmon ohjaamisessa ja pelin tavoitteiden saavuttamisessa. Esittely-sekvenssin tulisi olla ohitettavissa, ettei sitä tarvitse katsoa joka ikinen kerta kun pelin aloittaa. (Rautiainen 2002, 32–34.)

- Pelin aloitus nollaa pelimuuttujat ennen pelin alkua. Näitä ovat esimerkiksi pelaajan pistemäärä, panosten lukumäärä tai vaikeustaso. (Rautiainen 2002, 32–34.)

- Pelisilmukka on moduuleista tärkein. Sen vastuulla on peliympäristön luonti, objektien hallinta, pelaajan syötteiden käsittely ja ruudun päivitys. (Rautiainen 2002, 32–34.) Pelisilmukasta on lisää tietoa luvussa 4.3.2.

- Pelin päättyminen -moduuli kertoo pelaajalle pelin päättyneen ja pelaajan saaman tuloksen. Pelin loppumisesta voidaan kertoa monella tavalla. Joissakin peleissä riittää esimerkiksi pelkkä ”GAME OVER” -ruutu, kun taas toisissa on syytä näyttää loppuanimaatio tai tarkat tilastot ja pistemäärä. Pelaajan pitäisi myös päästä aloittamaan uusi peli tai poistua pelistä kokonaan. (Rautiainen 2002, 32–34.)

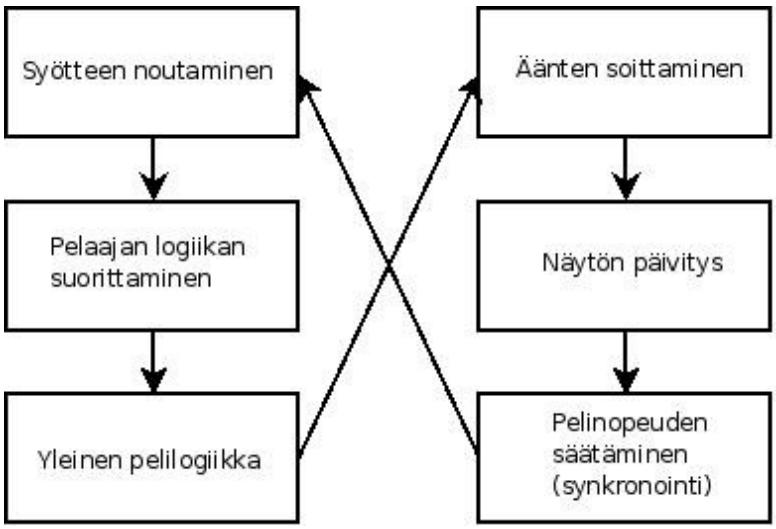
- Pelin sulkeminen -moduulin tehtävä on vapauttaa Alustaminen-moduulissa varatut resurssit ja sammuttaa pelisovellus ilman muistivuotoja. Ennen lopetusta on myös syytä kysyä pelaajalta, tallennetaanko peli. (Rautiainen 2002, 32–34.)

4.3.2 Pelisilmukka

Pelin selkäranka on pelisilmukka (Game loop). Pelisilmukkaa toistetaan siihen saakka, kunnes pelaaja läpäisee pelin tai lopettaa pelaamisen. Silmukkaa voitaisiin kuvata pseudokoodilla seuraavasti:

```
while(pelaaja ei ole lopettanut)
    kerää syöte()
    pelin logiikka()
    liikuta hahmoja()
    piirrä grafiikka()
```

Tyypillisen pelisilmukan (KUVIO 4) vastuulla ovat ainakin seuraavat kolme asiaa: syötteen vastaanotto, objektien päivittäminen ja animaatiokehyksen (animation frame) muuntaminen rasterigrafiikaksi. Pelisilmukka voidaan jakaa kuuteen lohkoon (KUVIO 4).(Rautiainen 2002, 34–35.)



KUVIO 4. Tyypillinen pelisilmukka (Rautiainen 2002, 35.)

- Syötteen noutaminen -lohko lukee pelaajan käyttämältä syöttölaitteelta, kuten esimerkiksi peliohjaimelta, syötearvoja, jotka se muuntaa sellaiseen muotoon, että pelihahmoa voidaan ohjata. Syöttölaitetta luetaan jokaisella kierroksella tietyssä kohdassa. Yleensä tämä tehdään silmukan alussa, jotta viimeisin painallus vaikuttaisi seuraavaan näytön näkymään. Liian raskas silmukka voi johtaa siihen, että käyttäjän tekemät painallukset eivät aina rekisteröidy ja pelin ohjaaminen muuttuu vaikeaksi. Tämä voidaan estää tekemällä syötteen lukeminen useassa eri silmukan kohdassa ja laskemalla painallusten summa, kun objekteja päivitetään. Toinen tapa on puskuroida painallukset erillisellä säikeellä, joka vastaa syötelaitteen lukemisesta ja tapahtumien jonohallinnasta. (Rautiainen 2002, 35–37.)

- Pelaajan logiikan suorittamiseen kuuluvat pelihahmon tekemät liikkeet eli siirtyminen paikasta toiseen ja esimerkiksi aseen laukaiseminen. Lohko tarkastaa pelin tilan, jotta tiedetään tehdyt muutokset. (Rautiainen 2002, 35–37.)

- Yleisen pelilogiikan vastuulla on tekoälykoodin suoritus. Tekoälykoodissa on määritelty pelin ”säännöt” eli se, kuinka objektit reagoivat pelihahmon liikkeisiin. Pelimaailman objektit käsitellään ja niiden sijainnit asetetaan lohkossa yksitellen. Samaan aikaan piirretään hahmo puskuriin. Muita toimenpiteitä voi olla esimerkiksi törmäysten tunnistaminen, jolla selvitetään, onko kaksi tai useampia hahmoja tai objekteja törmännyt toisiinsa. Tunnistamisessa voidaan käyttää esimerkiksi suorakulmion mallista rajalaatikkoa (bounding box), joka rajaa objektit. Jos laatikot leikkaavat, on tapahtunut törmäys. Yhtä hahmoa kohden voidaan käyttää useita rajalaatikoita, joita pystyy myös suurentamaan tai pienentämään. (Rautiainen 2002, 35–37.)

- Äänten soittaminen -lohkon vastuulla on tausta- ja käyttöliittymä-äänten suorittaminen. Musiikin ja ääniefektien soittaminen voidaan hoitaa joko tässä lohkossa tai Syötteen noutaminen -lohkossa. Esimerkiksi ammuskelupelissä lohko voi kutsua funktiota, jonka vastuulla on aseen ammuksen luonti ja ammukselle ominainen ääni. (Rautiainen 2002, 35–37.)

- Näytön päivitys alkaa taustagrafiikan piirtämisestä. Tausta voi vaihdella pelkästä yksivärisestä seinästä aina monimutkaisiin ja näyttäviin graafisiin tekstuureihin. Pelihahmot ja objektit piirretään taustan päälle. Grafiikka luodaan erilliseen puskuriin, ja kun kaikki piirtäminen on saatu tehtyä, tulostetaan puskuri ruudulle. Illuusio animaatiosta syntyy, kun sama toistetaan useasti peräkkäin ja hahmojen paikkaa ruudulla vaihdetaan. (Rautiainen 2002, 35–37.)

- Pelinopeuden säätämistä tarvitaan, jotta peli toimisi samaan tapaan sekä uudemmilla että vanhemmilla koneilla. Pelinopeutta voidaan muuttaa siten, että silmukan alussa tallennetaan aloitusaika, joka vähennetään silmukan loppuajasta. Mikäli tulos on määriteltyä aikaa pienempi, voidaan peliä hidastaa keskeyttämällä suoritus erotuksen pituiseksi ajaksi. (Rautiainen 2002, 35–37.)

5 PYGAME

5.1 Pygamen perusteet

Pygame on alustariippumaton moduuli Python-ohjelmointikielelle. Pygame toimii C-kielellä toteutetun Single DirectMedia Layerin (SDL) päällä. SDL on matalan tason multimediakirjasto, jonka avulla voi luoda kaksiulotteista (2D) grafiikkaa, käsitellä ääniä ja hallita hiirtä, näppäimistöä, peliohjaimia ja esimerkiksi dvd-asemaa. Pete Shinners aloitti Pygamen kehittämisen syksyllä 2000, kun edellinen Pythonin ja SDL:n yhdistävä projekti, Mark Bakerin johtama PySDL lopetettiin. Shinnersin mielestä PySDL muistutti liikaa C-kieltä, ja hän pyrkikin luomaan moduulin, joka hyödyntäisi Pythonin korkean tason tietorakenteita ja olisi syntaksiltaan yhtä yksinkertainen ja selkeä kuin Python. Versio 1.0 julkaistiin keväällä 2001. (Shinners 2002.)

5.2 OilWorker-peli

Testasin Will Alvarezin tekemää OilWorker-peliä, joka on ladattavissa ilmaiseksi www.pygame.org-sivustolta. Sivustolta löytyy satoja Pygamella tehtyjä pelejä, joista iso osa on julkaistu GNU GPL -lisenssillä. Sen myötä kuka tahansa saa kopioida, muuttaa ja jakaa eteenpäin peliä ja sen lähdekoodia. Myös kaupallinen hyödyntäminen on sallittu. Pelin tuorein versio on 0.8, eikä tekijä ole julkaissut uudempaa versiota yli kahteen vuoteen.

5.2.1 Pelin idea

OilWorker on 1980-luvun alussa ilmestyneen PipeMania-pelin klooni, eli pelien ulkoasu ja pelimekaniikka ovat melkein identtiset. Genreltään Oilworker on puzzlepeli. Pelin tarkoituksena on rakentaa öljyputki porakaivon ja säiliön välille öljy-yhtiön toimeksi antamana. Mitä pitempi putki on, sitä enemmän pelaaja saa pisteitä. Öljyputki koostuu yksittäisistä palasista, jotka pelaaja asettaa maahan yksi kerrallaan. Peli on jaettu useisiin tasoihin, joita suorittamalla pääsee aina uuteen tasoon, kunnes tasot loppuvat kesken. Kullekin tasolle on asetettu minimitavoite asennettujen putkien suhteen, ja tavoite kasvaa taso tasolta. Myös

putken rakentamiseen käytettävissä oleva aika vähenee taso tasolta. Ensimmäisellä tasolla aikaa on roimasti mutta lopulta pelaaja joutuu toimimaan hyvin nopeasti.

Jos aika loppuu tai putki ei yllä määränpäähän tai pelaaja ei ole käyttänyt minimimäärää putkia, peli päättyy ja pelaajan keräämä pistemäärä lisätään ennätyslistalle (high scores), jos listalla ei ole jo kymmentä parempaa tulosta ennestään. Sen jälkeen peli palaa päävalikkoon (KUVIO 5). Päävalikosta voi aloittaa uuden pelin, jatkaa vapaavalintaisesta jo läpäistystä tasosta, katsoa parhaimpia tuloksia, lukea peliohjeet ja sammuttaa pelin.

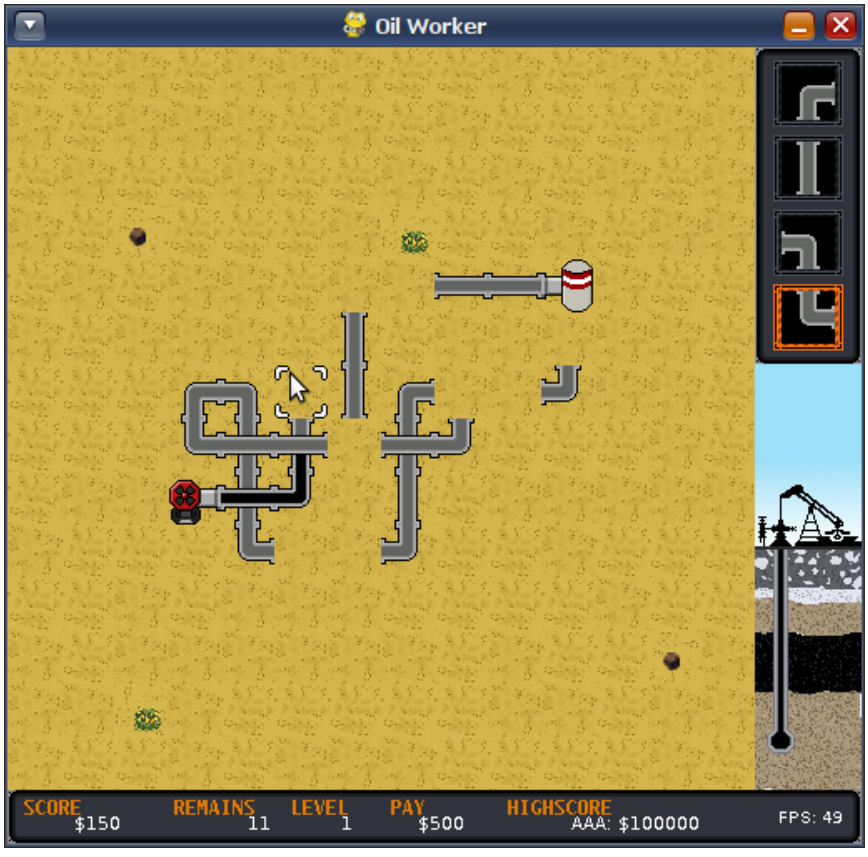
KUVIO 5. OilWorker-pelin päävalikko

5.2.2 Grafiikka

Graafiselta ulkoasultaan peli on varsin yksinkertainen. Miljöönä on yleensä aavikko, jossa ei juuri kasvustoa ole. Grafiikka muodostuu .XPM- ja .PNG-tiedostopäätteisistä kuvatiedostoista. Pelialue koostuu neljästätoista pysty- ja vaakaruudusta. Yhteen ruutuun voi laittaa yhden palasen putkea, ellei ruudussa ole esimerkiksi eläimen fossiilia, isoa kiveä tai jo-

tain muuta estettä. Peli-ikkuna on kooltansa vakio (516 x 509 pikseliä), eikä peliä voi pelata kokoruudussa. Peli-ikkunan alaosassa näkyy pelaajan pistemäärä, minitavoitteesta puuttuvien putkien lukumäärä, pelin taso, putkiurakoitsijan saama palkka sekä ennätyslistan paras tulos ja sen tekijän nimikirjaimet (KUVIO 6). Alaoikealla näkyy kuvataajuus (frames per second) eli näytölle sekunnissa piirrettävien kuvien määrä. Sen yläpuolella on animoitu lämpömittarin mallinen öljyputki. Kun putken sisus on kokonaan musta, lähtee öljy pumppautumaan maan sisuksista pelaajan rakentamaan öljyputkeen.

Öljyn eteneminen pelaajan rakentamassa putkessa näkyy niin ikään mustana. Oikealla ylhäällä näkyy pelaajan seuraavaksi käytettävissä olevat putkenpalaset. Pelaaja voi asetella peliruudulle noista neljästä alimmaisen. Kun alimmainen on käytetty, siirtyy toiseksi alimmainen sille paikalle jne.



KUVIO 6. Peli-ikkuna

5.2.3 Äänitehosteet

Pelissä on useita taustamusiikkeja, jotka vaihtelevat tason mukaan. Putkenpalasen asentamisesta kuuluu aina porauksen ääni. Ennen öljyn pumppauksen aloittamista kuuluu hälytyssireeni. Öljyn virratessa yksittäisen putkenpalasen läpi kuuluu kassakoneen ääni merkiksi siitä, että pelaaja saa pisteitä (rahaa). Tason läpäisyn merkiksi kuuluu ihmisten hurraahuutoja ja pelin päättyessä buuausta.

5.2.4 Kontrollit

Päävalikossa liikutaan nuolinäppäimillä ja valinta hyväksytään Enterillä. Muuten peliä ohjataan hiirellä. Putkenpalasten asentamiseen riittää yksi klikkaus halutussa ruudussa. Pelin voi pysäyttää painamalla Enteriä ja päävalikkoon pääsee takaisin painamalla Esciä. Putkessa valuvan öljyn nopeutta voi lisätä painamalla välilyöntiä.

5.2.5 Yhteenveto pelistä

Mielestäni OilWorker on varsin mainio peli. Olen aina pitäny PipeMania-pelin klooneista, ja OilWorker ei tee poikkeusta. Pelin pelaaminen on äärimmäisen yksinkertaista ja toimivaa; putkenpalaset asentuvat juuri sinne minne pitääkin. Grafiikka on mielestäni sopivan hyvää, ja aavikko näyttää juuri siltä miltä pitääkin. Putkessa kulkeva öljy on animoitu hyvin. Miinuspisteitä antaisin putkenpalasen asennusäänestä ja kassakoneen kilinästä sekä pelin ajoittaisesta kaatuilusta.

5.3 Pygame-ohjelmointi

5.3.1 Pygamen asennus

Pygame-moduuli ei sisälly Pythonin perusasennuspakettiin vaan se tulee ladata ja asentaa erikseen. Asennuspaketin voi ladata osoitteesta www.pygame.org/download.shtml. Pygame

on saatavilla Windowsille, Linuxille ja Mac OS X:lle. Linux-jakeluilla Pygamen löytää pakettivarastosta, mutta tuoreimman version joutuu yleensä lataamaan ja asentamaan manuaalisesti. Pygamen asennuksen onnistumisesta pääsee parhaiten selville kirjoittamalla tulkkiin käskyn ”import pygame” ja painamalla Enter. Jos tulkki ei ilmoita mitään, asennus onnistui.

Seuraavissa luvuissa käydään läpi Pygamen ominaisuuksia viiden eri esimerkkiohjelman/-pelin kautta. Esimerkkien tekoon olen käyttänyt Geany-ohjelmointiympäristöä sekä Pythonin versiota 3.0.1 ja Pygamen versiota 1.9.1.

5.3.2 Piirto

Ensimmäinen esimerkkiohjelma piirtää ikkunaan valkoiselle taustalle mustan ympyrän, vihreän kolmion ja punaisen suorakulmion (KUVIO 7). Ohjelman lähdekoodi on liitteessä 1.

KUVIO 7. Kuvioita pienessä ikkunassa

Ennen varsinaisen ohjelmoinnin aloittamista tulee lähdekooditiedostoon sisällyttää tarvittavat moduulit:

```
import pygame, sys
from pygame.locals import *
```

Ensimmäisellä rivillä sisällytetään moduulit pygame ja sys. Toisella rivillä sisällytetään lisäksi pygame.locals-moduulista kaikki sen sisältämät ominaisuudet. Näitä ovat mm. vakiomuuttujat QUIT ja K_ESCAPE, joita käytetään myöhemmissä esimerkeissä. Pygame-moduuli tulee vielä alustaa komennolla *pygame.init()*.

Sisällyttämisen ja alustamisen jälkeen päästään luomaan varsinaista grafiikkaa. Set_mode()-metodilla luodaan ikkuna, johon kuviot on tarkoitus piirtää:

```
Ikkuna = pygame.display.set_mode((250, 200), 0, 32)
```

Set_mode()-metodi koostuu kolmesta eri parametrista: ensimmäinen, tuple-tyyppiä oleva parametri kertoo ikkunan leveyden ja korkeuden pikseleissä. Esimerkkiohjelma on siis 250 pikseliä leveä ja 200 pikseliä korkea. Toisen parametrin arvoksi voidaan asettaa yksi tai useampi seuraavista vaihtoehdoista: 0, FULLSCREEN, DOUBLEBUF, HWSURFACE, OPENGL, RESIZABLE ja NOFRAME. Mikäli arvoksi asetetaan nolla, pysyy ikkunan koko samana kaikissa tilanteissa, eikä sen kokoa voida muuttaa. RESIZABLE-arvo taas sallii ikkunan koon muuttamisen ja FULLSCREEN-arvo määrittää ohjelman käynnistymään koko ruudussa. Kolmas parametri on värisyvyys, joka on esimerkissä asetettu 32-bittiseksi. Set_caption()-metodilla voidaan ikkunalle antaa nimi:

```
pygame.display.set_caption('Erilaisia kuvioita')
```

Seuraavaksi luodaan neljä värimuuttujaa:

```
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
```

Kaikki neljä muuttujaa ovat tuple-tietotyyppiä. Pygamessa käytetäänkin tupleja varsin usein listojen sijaan, mikä nopeuttaa ohjelman suorittamista. (Sweigart 2009, 271.) Muuttujille määritellään RGB-värimallin mukaiset arvot asteikolla 0–255. Värimuuttujia ei ole tietenkään pakko käyttää, mutta niiden avulla ohjelmakoodi selkeytyy hiukan ja ohjelmoijan työmäärä vähenee. Tausta maalataan kokonaan valkoiseksi fill()-funktion avulla:

```
Ikkuna.fill(WHITE)
```

Esimerkkikuviot piirretään kaikki samaan tapaan. Ympyrän piirtoon käytetään draw.circle()-funktiota, suorakulmion piirtoon draw.rect()-funktiota ja niin edelleen. Piirrettävälle ympyrälle pitää antaa viisi parametria: pinta, johon ympyrä piirretään, ympyrän väri, keskipisteen koordinaatit (x,y), ympyrän säde ja piirin paksuus pikseleinä. Jos jälkimmäisen arvon asettaa nollaksi, täytetään ympyrä valitulla värillä.

```
pygame.draw.circle(Ikkuna, BLACK, (75, 100), 50, 0)
```

Suorakulmiossa on on kolme parametria; pinta, väri ja pisteiden koordinaatit;

```
pygame.draw.rect(Ikkuna, RED, (20, 155, 200, 25))
```

Monikulmiossa parametreja on neljä; pinta, väri, pisteiden koordinaatit ja paksuus, joka jätetään ympyrän tavoin nollaksi;

```
pygame.draw.polygon(Ikkuna, GREEN, ((220,50),(220,125),(140,125)), 0)
```

Varsinainen piirto tapahtuu vasta, kun funktiota display.update() on kutsuttu. Tämä kannattaa tehdä vasta loppuvaiheessa eikä jokaisen yksittäisen graafisen elementin jälkeen, sillä grafiikan piirtäminen on verrattain hidas operaatio. Funktiota on kutsuttava joka kerta, kun halutaan päivittää ruutua. Lähdekoodin lopussa on vielä yksinkertainen pelisilmukka, joka ainoa tehtävä on pitää ohjelma käynnissä siihen asti, kun käyttäjä haluaa sen sulkea.

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
```

Seuraavissa esimerkeissä käytetään hyödyksi jo läpikäytyä ohjelmakoodia ja lisätään joitakin uusia ominaisuuksia vanhan lisukkeeksi.

5.3.3 Animaatio

Toinen esimerkkiohjelma liikuttaa valkoista ympyrää mustalla taustalla vasemmalta oikealle, kunnes ympyrä on kokonaan kuvaruudun ulkopuolella (KUVIO 8). Sen jälkeen animaatio alkaa taas alusta. Ohjelman lähdekoodi on liitteessä 2. Lähdekoodin alkuosa on lähes

identtinen edellisen esimerkin kanssa, paitsi tässä esimerkissä tarvitaan myös moduulia time, jonka avulla voidaan hallita ja mitata aikaa. Lisäksi ikkunan koko on 400 x 200 kuvapistettä.

KUVIO 8. Vasemmasta reunasta liikkeelle lähtenyt valkoinen ympyrä

Tässä esimerkissä tarvitaan vain kaksi värimuuttujaa, musta ja valkoinen. Lisäksi luodaan muuttuja x, joka on ympyrän keskipisteen sijainti x-akselilla ennen sen liikkeelle lähtöä:

```
x = –50
```

Kun ympyrän säde on samat 50 kuvapistettä, lähtee ympyrä liikkeelle ikkunan ulkopuolelta. Edellisessä esimerkissä pelisilmukalla ei ollut vielä varsinaista toimintaa, mutta tässä esimerkissä silmukka vastaa sekä piirtämisestä että ajan käytöstä. Tämä tapahtuu siten, että tarvittavat funktiot sijoitetaan while-silmukan sisään:

```
while True:
    for event in pygame.event.get():
            if event.type == QUIT:
                    pygame.quit()
                    sys.exit()
    ruutu.fill(BLACK)
```

Peli-ikkuna, jonka nimi on tässä tapauksessa ruutu, maalataan mustaksi jo tutuksi tulleella tavalla. Sen jälkeen piirretään valkoinen ympyrä:

```
pygame.draw.circle(ruutu, WHITE, (x, 50), 50, 0)
x+=1
if x>450:
        x=-50
```

Ympyrä piirretään vaaka-akselilla kohtaan x ja pystyakselilla 100 kuvapisteen päähän yläreunasta. Seuraavat kolme riviä määrittävät sen, miten ympyrä tulee liikkumaan vaaka-akselilla. X-arvo kasvaa yhdellä niin kauan, kunnes se on yli 450 kuvapistettä eli ympyrä on siirtynyt ikkunan oikealle puolelle kokonaan näkymättömiin. Sen jälkeen ympyrä siirretään takaisin alkupisteeseen −50. Peli-ikkuna tulee päivittää jokaisen siirtymän jälkeen, ja tästä vastaa update()-funktio:

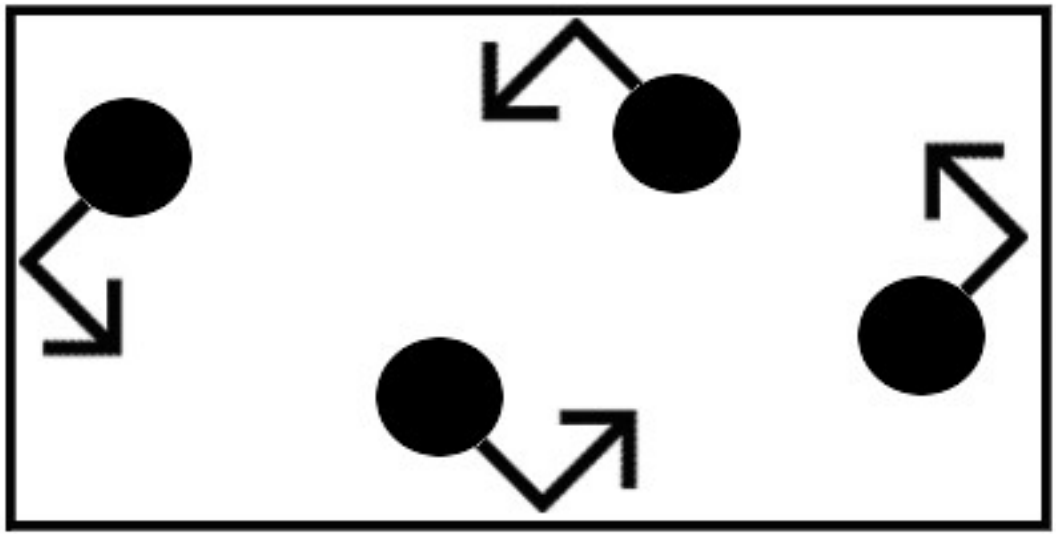
```
pygame.display.update()
time.sleep(0.005)
```

Time.sleep()-funktio vastaa siitä, että animaatio etenee ennalta määritellyllä nopeudella eikä tietokoneen nopeuden mukaan. Uudemmilla ja tehokkaammilla koneilla animaatio olisi paljon nopeampi kuin vanhemmilla koneilla, ja näin peli näyttäytyisi ihan erilaisina eri tietokonekokoonpanon omistaville pelaajille. Esimerkissä funktio on asetettu pysäyttämään animaation 5 millisekunnin ajaksi. Silmukka siis toistaa *maalaa tausta-piirrä ympyrä-pysähdy*-toimintoa loputtomiin. Ilman joka kerta toistuvaa ikkunan taustan maalaamista valkoinen ympyrä piirtyisi aina edellisen päälle ja tuloksena olisi mustalla taustalla kulkeva valkoinen raita.

Seuraavassa esimerkissä jatketaan animaation merkeissä, mutta enää liike ei saa jatkua ikkunan ulkopuolelle vaan kohteen tulee pysyä ennalta määritetyllä pelialueella.

5.3.4 Törmäys

Kolmas esimerkkiohjelma liikuttaa rantapalloa peli-ikkunan sisällä (KUVIO 10). Rantapallo ei voi poistua pelialueen ulkopuolelle kuten edellisen esimerkin ympyrä. Kohdatessaan seinän eli ikkunan äärirajan ottaa pallo kimmokkeen ja jatkaa matkaa uuteen suuntaan, kunnes törmää seinään taas uudelleen (KUVIO 9). Ohjelman lähdekoodi on liitteessä 3.



KUVIO 9. Kimmoke seinästä

Lähdekoodin alkuosa on identtinen edellisen esimerkin kanssa. Peli-ikkuna on saman kokoinen (400 x 200) ja värimuuttujia tarvitaan vain yksi, musta. Pallon liikkeen mahdollistamiseksi luodaan kaksialkioinen lista ”nopeus”, johon määritellään pallon siirtymä x- ja y-akselilla. Pallo siirtyy jokaisella pelisilmukan kierroksella 2 pikseliä vaaka- ja pystyakselilla.

KUVIO 10. Törmäilevä rantapallo

Ennen pelisilmukkaa tulee vielä luoda pallo-muuttuja, johon ladataan GIF-muodossa oleva kuva rantapallosta:

```
pallo = pygame.image.load("ball.gif")
ballrect = pallo.get_rect()
```

Ball.gif-kuvatiedosto sijaitsee samassa alikansiossa lähdekoodin kanssa, joten kansiopolkua ei tarvitse määritellä erikseen. Seuraavalla rivillä luodaan muuttuja ”ballrect”, johon tallennetaan get.rect()-funktion avulla pallon sijainti. Pygamen erityisominaisuus on Rect-objekti, jota käytetään monikulmioiden pinta-alojen manipulointiin ja tallennukseen. Rect-objekti koostuu arvoista top, left, bottom ja right eli ylä, vasen, ala, oikea (KUVIO 11). Rect-objekteja voidaan myös luoda jo olemassa olevista Rect-objekteista tai muuttujista, joiden nimestä löytyy sana ”rect”. Pelisilmukan sisällä palloa liikutetaan Rect.move()-metodin avulla:

```
ballrect = ballrect.move(nopeus)
```

Rect.move()-metodilla siis siirretään ballrectin sijaintia nopeus-listalla olevien arvojen verran. Uusi sijainti tallennetaan saman vanhan ballrect-muuttujan päälle. Toistaiseksi rantapallo kulkee vasta yhteen suuntaan, oikealle alaviistoon. Pallo saadaan muuttamaan suuntaa seuraavalla tavalla:

```
if ballrect.left < 0 or ballrect.right > leveys:
    nopeus[0] = -nopeus[0]
if ballrect.top < 0 or ballrect.bottom > korkeus:
    nopeus[1] = -nopeus[1]
```

Ensimmäisellä rivillä tarkastetaan kaksi asiaa: onko Rect-objektin left-arvo pienempi kuin nolla ja onko Rect-objektin right-arvo suurempi kuin ikkunan leveys. Mikäli jompikumpi väittämistä pitää paikkansa, nopeus-listan ensimmäisen alkion arvo 2 muutetaan miinusmerkkiseksi (–2), eli pallo kulkee samalla nopeudella, mutta vastakkaiseen suuntaan. Sama tarkastelu tehdään myös y-akselin suhteen, top- ja bottom-arvojen kanssa.

Lopuksi tulee vielä maalata ikkunan tausta, piirtää pallo ruudulle ja päivittää ruutu:

```
ruutu.fill(BLACK)
ruutu.blit(pallo, ballrect)
pygame.display.flip()
time.sleep(0.01)
```

Pallo piirretään ikkunaan blit()-funktion avulla. Ensin mainitaan piirron kohde eli ruutu, sitten blit ja sulkuihin pallo-muuttuja ja Rect-objekti ballrect, joka siis sisältää pallon kulloisetkin koordinaatit ja mitat. Animaatio ”uinuu” 10 millisekuntia joka kierroksella.

Seuraavissa esimerkeissä ei enää olla pelkän esiohjelmoidun toiminnan varassa vaan pelaaja pääsee liikuttamaan pelihahmoa sekä hiiren että näppäimistön avulla.

5.3.5 Hiiri- ja näppäimistöohjaus

Neljäs esimerkkiohjelma päästää pelaajan ohjaamaan kahta kalaa. Siikaa (pitkulainen kala) ohjataan nuolinäppäimillä ja lahnaa (leveä kala) hiirellä (KUVIO 12). Pelin lähdekoodi on liitteessä 4. Lähdekoodin alkuosa on taas hyvin samanlainen edeltävien esimerkkien kanssa.

KUVIO 12. Kalat vedessä

Kahdessa edellisessä esimerkissä käytettiin time.sleep()-funktiota, mutta tässä esimerkissä luodaan erityinen Clock-objekti:

```
mainClock = pygame.time.Clock()
```

Siinä missä time.sleep()-funktio hidasti animaatiota tietyn millisekuntimäärän verran, voidaan Clock-objektilla hallita pelin kuvanopeutta (framerate) eli sitä, kuinka monta kertaa kuva päivittyy yhden sekunnin aikana. Peli-ikkunan koko on tässä esimerkissä 640 x 360 kuvapistettä, ja peli käyttää mustan taustan sijasta merellistä taustakuvaa:

```
tausta = "meri.jpg"
background = pygame.image.load(tausta).convert()
```

Kuvatiedosto meri.jpg ladataan ensin muuttujaan nimeltä tausta. Sen jälkeen tausta-muuttuja muutetaan vielä convert()-funktion avulla background-nimiseksi muuttujaksi. Tämän pitäisi hivenen nopeuttaa taustojen piirtämistä. Seuraavaksi luodaan kaksi muuttujaa edellisistä esimerkistä tutulla tavalla:

```
siika = pygame.image.load("siika.png")
rect1 = siika.get_rect()

lahna = pygame.image.load("lahna.png")
rect2 = lahna.get_rect()
```

Kummallekin kalalle luodaan oma Rect-objekti. Ennen pelisilmukkaa luodaan myös muuttujat näppäimistöpainalluksille ja pelinopeudelle:

```
moveLeft = False
moveRight = False
moveUp = False
moveDown = False

MOVESPEED = 8
```

Muuttujat moveLeft, moveRight, moveUp ja moveDown ovat loogisia tietotyyppejä, eli ne voivat olla vain joko False- tai True-tilassa. MOVESPEED-muuttuja toimii kuten 3. esimerkin muuttuja nopeus, eli arvo 8 on kalan siirtymä kuvapisteinä. Pelisilmukan sisällä määritellään ensin näppäinkomennot:

```
if event.type == KEYDOWN:
        if event.key == K_LEFT:
                moveRight = False
                moveLeft = True
        elif event.key == K_RIGHT:
                moveLeft = False
                moveRight = True
        elif event.key == K_UP:
```

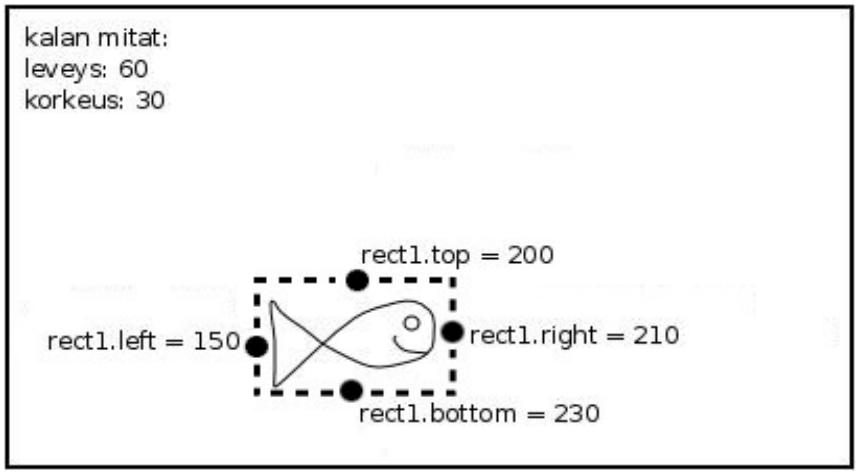
```
                    moveDown = False
                    moveUp = True
                elif event.key == K_DOWN:
                    moveUp = False
                    moveDown = True

            if event.type == KEYUP:
                if event.key == K_ESCAPE:
                    pygame.quit()
                    sys.exit()
                if event.key == K_LEFT:
                    moveLeft = False
                elif event.key == K_RIGHT:
                    moveRight = False
                elif event.key == K_UP:
                    moveUp = False
                elif event.key == K_DOWN:
                    moveDown = False
```

Pygamessa tapahtumankäsittelystä (event handling) vastaa event-moduuli. Tässä esimerkkiohjelmassa käytetään KEYDOWN- ja KEYUP-tapahtumia eli määritellään, mitä tapahtuu silloin kun näppäin on painettuna alas ja kun näppäin päästetään vapaaksi. Siian ohjaamisen käytetään siis nuolinäppäimiä, joita vastaavat vakiomuuttujat K_LEFT, K-RIGHT, K_UP ja K_DOWN. Vasenta nuolinäppäintä painamalla moveLeft-muuttuja saa arvon tosi ja vastakkaiseen suuntaan vievä moveRight arvon epätosi. Sama pätee muihin nuolinäppäimiin: siikaa ei voi ohjata yhtä aikaa sekä alas että ylös tai sekä vasemmalle että oikealle. K_ESCAPE-vakiomuuttuja vastaa Esc-näppäimen painallusta. Peli päättyy kun pelaaja on painanut ja vapauttanut Esc-näppäimen. Näppäimistöohjaus tapahtuu seuraavasti:

```
if moveDown and rect1.bottom < korkeus:
        rect1.top += MOVESPEED
if moveUp and rect1.top > 0:
        rect1.top -= MOVESPEED
if moveLeft and rect1.left > 0:
        rect1.left -= MOVESPEED
if moveRight and rect1.right < leveys:
        rect1.right += MOVESPEED
```

Jos pelaaja painaa nuolinäppäintä alas ja siian rect1.bottom-arvo on ikkunan korkeutta pienempi, kasvatetaan rect1.top-arvoa kahdeksalla kuvapisteellä. Liike pysähtyy, kun rect1.bottom on yhtä suuri kuin korkeus (KUVIO 11). Kolme muuta nuolinäppäintä toimivat samaan tapaan.



KUVIO 11. Rect-objekti ja sen koordinaatit

Lahna-kalan hiiriohjaus on tehty seuraavasti:

```
sijainti = pygame.mouse.get_pos()
sijx = sijainti[0]
sijy = sijainti[1]
sijx -= lahna.get_width()/2
sijy -= lahna.get_height()/2
rect3 = rect2.move(sijx,sijy)

pygame.mouse.set_visible(False)
```

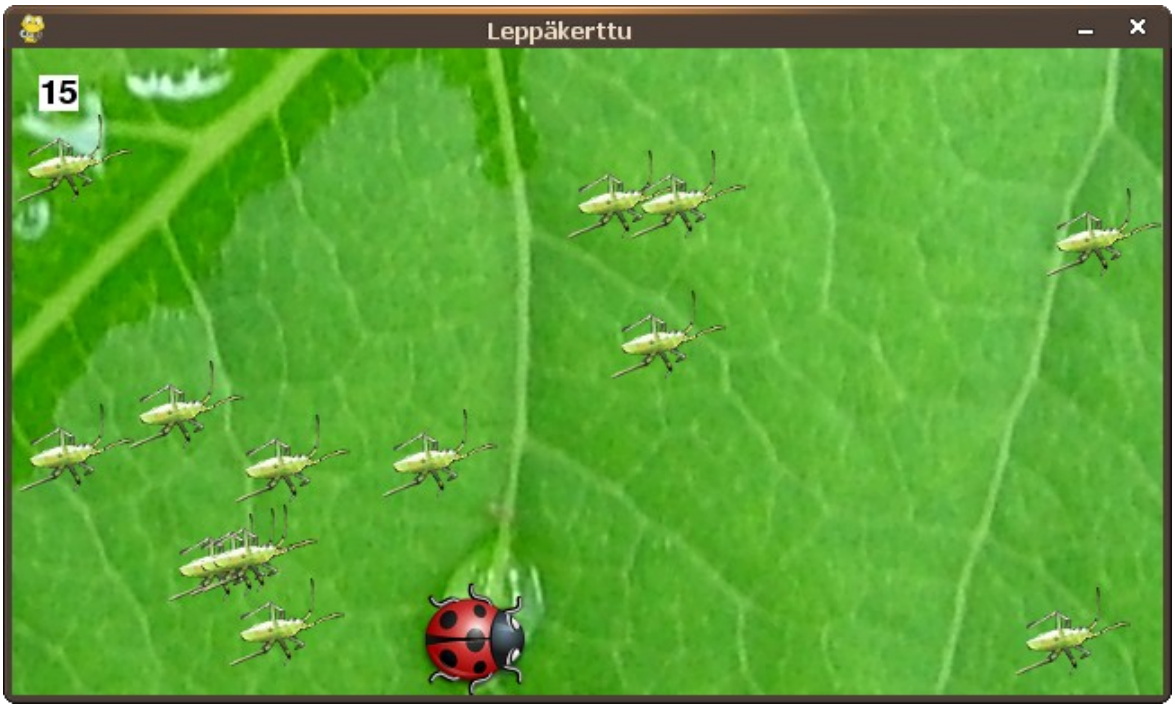
Hiiren sijainti eli sen x- ja y-arvot tallennetaan sijainti-listaan mouse.get_pos()-funktiolla. X- ja y-arvot erotellaan omiksi muuttujikseen ja get_width()- ja get_height()-metodeilla saadaan selville lahna-kuvan leveys ja korkeus, jotka jaetaan ensin kahdella ja vähennetään vielä x- ja y-sijainneista. Lopuksi sijainnit tallennetaan uuteen rect3-objektiin. Näin saadaan hiiren osoitin lahnan keskelle. Hiiriosoitin piilotetaan mouse.set_visible()-funktion avulla.

Lopuksi vielä piirretään tausta ja kalat ruudulle ja asetetaan pelin ruudunpäivitysnopeudeksi 40:

```
screen.blit(background,(0,0))
screen.blit(siika, rect1)
screen.blit(lahna, rect3)
pygame.display.update()
mainClock.tick(40)
```

5.3.6 Äänitehosteet, musiikki ja päällekkäiset objektit

Viidennessä esimerkkiohjelmassa ohjataan kirvoja syövää leppäkerttua. Jokaisesta syödystä kirvasta saa yhden pisteen, ja pistemäärä näkyy vasemmassa yläkulmassa (KUVIO 13). Pelin lähdekoodi on liitteessä 5. Leppäkerttua ohjataan nuolinäppäimillä, ja syöminen tapahtuu, kun leppäkerttu ja kirva koskettavat toisiaan.



KUVIO 13. Kirvat ja leppäkerttu

Pelissä tarvitaan satunnaislukuja, ja niinpä lähdekoodin sisällytetään edellistä esimerkeistä tuttujen kirjastojen lisäksi satunnaislukukirjasto random. Leppäkerttu ja kirva luodaan samaan tapaan kuten edellisen esimerkin kalat:

```
kerttu = pygame.image.load("bug.png")
rect1 = kerttu.get_rect()

kirva = pygame.image.load("kirva_pieni.png")
```

Yksittäiselle kirvalle luodaan Rect-objekti silmukassa:

```
otokkamaara = []
for i in range(20):
    otokkamaara.append(pygame.Rect(random.randint(0,leveys-64),
    random.randint(0,korkeus-49),64,49))
```

Otokkamaara on lista, johon arvotaan for-silmukan avulla 20 Rect-objektia. Myöhemmässä vaiheessa lähdekoodia luodaan kirvoja, jotka käyttävät hyödykseen näitä objekteja eli x- ja y-arvoja. Randint()-funktio arpoo x-akselin sijainnin nollan ja ikkunan leveydestä (640 kuvapistettä) vähennetyn kirvan leveyden (64) avulla. Samalla logiikalla luodaan myös y-akselin koordinaatit eli korkeudesta (360) vähennetään kirvan korkeus (49).

```
laskuri = 0
syotavat = 60
syodytOtukset = 0
```

Muuttujat laskuri, syotavat ja syodytOtukset luodaan myöhempää käyttöä varten. Liikemuuttujat ja pelinopeus toimivat samalla tavalla edellisen esimerkin kanssa. Ääniasetukset luodaan seuraavalla tavalla:

```
pickUpSound = pygame.mixer.Sound("glups.wav")
pygame.mixer.music.load('wind.wav')
pygame.mixer.music.play(-1, 0.0)
musicPlaying = True
```

Pygame.mixer-moduuli vastaa lyhyiden ääniefektien soittamisesta ja pygame.mixer.music vuorostaan taustamusiikin soittamisesta. Mixer.Sound()-konstruktorin avulla luodaan pickUpSound-muuttuja, joka soi aina, kun leppäkerttu saa syötyä kirvan. Taustamusiikkina toimiva wind.wav-tiedosto ladataan mixer.music.load()-konstruktorin avulla. Taustamusiikin soittaminen alkaa komennolla play(). Sulkujen sisällä on kaksi parametria, joista ensimmäinen määrittää sen, kuinka monesti musiikki halutaan soittaa ensimmäisen soiton päälle. Arvo 5 tarkoittaisi sitä, että musiikki soisi kuudesti peräkkäin. Arvo -1 tarkoittaa sitä, että taustakappale toistetaan niin kauan kuin peli pyörii. Jälkimmäinen parametri kertoo, mistä kohdin kappaleen soitto halutaan aloittaa. Arvolla 0.0 kappale soi siis aina kokonaan. Pistemäärän fontti ja sijainti määritellään seuraavasti:

```
basicFont = pygame.font.SysFont(None, 28)
textRect = (15,15,50,150)
pistemaara = "0"
```

Fontti tallennetaan basicFont-muuttujaan. BasicFont käyttää käyttöjärjestelmän omia fontteja (SysFont). Sulkujen sisällä olevat kaksi parametria ovat fontin nimi ja koko. Jos halutaan käyttää esimerkiksi Arial-fonttia, tulee se kirjoittaa sulkujen sisään lainausmerkeissä eli string-tyyppisenä. TextRect-objektiin on tallennettu pistemäärän sijainti (15 kuvapistettä reunoista) ja mitat. Pistemaara-muuttuja alustetaan string-tyyppiseksi, koska se helpottaa tulostamista.

Pelisilmukan alkupuoli on identtinen edellisen esimerkin kanssa, eli pelihahmoa ohjataan samoilla näppäimillä, ja pelin voi keskeyttää Esciä painamalla. Tausta ja leppäkerttu piirretään ikkunaan tutulla tavalla:

```
screen.blit(background,(0,0))
screen.blit(kerttu, rect1)
```

Peli-ikkunaan piirretään lisää kirvoja seuraavalla tavalla:

```
laskuri += 1
if laskuri >= syotavat:
        # lisää ruokaa
        laskuri = 0
        otokkamaara.append(pygame.Rect(random.randint(0,leveys-64),
        random.randint(0,korkeus-49),64,49))
```

Laskuri-muuttuja alustettiin lähdekoodin alussa nollaksi. Pelisilmukan pyöriessä laskuria kasvatetaan aina yhdellä, ja jos se kasvaa suuremmaksi tai yhtä suureksi kuin syotavat-muuttuja, laskuri nollataan ja otokkamaara-listaan arvotaan yksi uusi kirvan sijainti lisää. Leppäkertun ja kirvojen törmääminen tarkastetaan seuraavasti:

```
for f in otokkamaara[:]:
        if rect1.colliderect(f):
                otokkamaara.remove(f)
                syodytOtukset = syodytOtukset+1
                pistemaara = str(syodytOtukset)

                if musicPlaying:
                        pickUpSound.play()
```

Otokkamaara-lista käydään läpi for-silmukan avulla. Rect.collliderect()-funktio palauttaa arvon tosi, jos kaksi Rect-objektia ovat päällekkäin, tässä tapauksessa siis leppäkerttu ja satunnaisesti luotu kirva. Leppäkertun kohtaama kirva poistetaan listasta ja syodytOtukset-muuttujaa kasvatetaan yhdellä. Tämän jälkeen int-tyyppiä oleva syodytOtukset muunnetaan str-funktion avulla tekstimuotoon ja tallennetaan pistemaara-muuttujaan. Jokaisella syöntikerralla soitetaan pickUpSound-äänitiedosto, jos pelaaja ei ole mykistänyt pelin ääniä. Tämä varmistetaan if musicPlaying -ehdolla. Pistemäärän ruudulle piirtämistä varten luodaan text-muuttuja:

```
text = basicFont.render(pistemaara, True, BLACK, WHITE)
```

Text-muuttuja käyttää aikaisemmin luotua basicFont-fonttia. Sulkujen sisällä on määritelty, mitä piirretään (pistemaara) ja käytetäänkö reunojenpehmennystä sekä tekstin ja taustan väri.

```
for f in otokkamaara:
        screen.blit(kirva, f)

pygame.mouse.set_visible(False)

screen.blit(text, textRect)

pygame.display.update()
mainClock.tick(60)
```

Pelisilmukan lopuksi hoidetaan varsinainen piirto. Ruudunpäivitysnopeudeksi on asetettu 60 ruutua sekunnissa. Edeltävän viiden esimerkin kautta käytiin läpi Pygame-moduulin perusasiat.

6 YHTEENVETO

Opinnäytetyön tarkoituksena oli selvittää, miten yksinkertaisten ja kaksiulotteisten pelien ohjelmointi onnistuu Python-ohjelmointikielen ja sen Pygame-moduulin avulla. Pygame on selvästi tehty saman periaatteeen mukaan kuin itse Python-kieli, eli yksinkertainen on kaunista. Grafiikan piirto ja kuvatiedostojen käsittely on sen verta helppoa, että aloitteleva Pygame-ohjelmoija saa nopeasti näkyvää jälkeä aikaan. Pelihahmon sijainnin määrittäminen Rect-objektin avulla on niin ikään loogista ja toimivaa.

Pygamen kenties suurin heikkous on siitä kertovien laadukkaiden ohjelmointioppaiden vähäinen määrä. Pelkän virallisen dokumentaation avulla ei ole vielä kovin helppoa päästä peliohjelmoinnissa alkuun. Dokumentaatio olisi kaivannut lisää havainnollistavia esimerkkejä. Internetistä löytyy jonkin verran pienehköjä oppaita, mutta alkua pidemmälle ei niistä ole juuri iloa. Paras Pygamea käsittelevä peliohjelmointiopas mielestäni oli Al Sweigartin e-kirja Invent Your Own Computer Games With Python. Kirjassa käytiin peliohjelmoinnin perusperiaatteet varsin kattavasti läpi, ja ohjelmointiesimerkit olivat monipuolisia ja hyvin selitettyjä.

Pygame-peliohjelmoinnista kiinnostuneen on ensin hyvä opetella Pythonin perusteet, sillä peliohjelmoinnissa tarvitaan tavallisen ohjelmoinnin tapaan silmukoita, tietorakenteita ja muita kielen perusominaisuuksia.

LÄHTEET

Baker, Steve, Campbell, Ben & Collins, Mark. 2002. Linux Game Programming. Saatavissa: http://site.ebrary.com/. Luettu: 10.5.2010.

Kasurinen, Jussi. 2009. Python 3 -ohjelmointi. Helsinki: WSOY.

Kingsley-Hughes, Adrian & Kingsley-Hughes, Kathie. 2005. Beginning Programming. Saatavissa: http://site.ebrary.com/. Luettu: 10.5.2010.

Kuchling, A.M & Zadka, M. 2000. What's New in Python 2.0. Python Software Foundation. Www-dokumentti. Saatavissa: http://www.python.org/download/releases/2.0/new-python.htm. Luettu 26.5.2010.

Nikula, U & Vanhala, E. 2010. Python3 -ohjelmointiopas. Lappeenrannan teknillinen yliopisto. Pdf-tiedosto. Saatavissa: https://oa.doria.fi/handle/10024/63381. Luettu: 10.9.2010.

Otero, C. 2008. Python 3 Primer, Part 1: What's new. IBM developerWorks. Www-dokumentti. Saatavissa: http://www.ibm.com/developerworks/linux/library/l-python3-1/. Luettu 25.5.2010.

Python Software Foundation. 2006. Python History. Www-dokumentti. Saatavissa: http://svn.python.org/view/*checkout*/python/trunk/Misc/HISTORY. Luettu: 1.2.2010.

Rautiainen, L. 2002. Tietokonepelit: historia ja ohjelmistoprosessi. Joensuun yliopiston tietojenkäsittelytieteen laitos, tietojenkäsittelyn pro gradu -tutkielma. Pdf-tiedosto. Saatavissa: http://bonecode.com/downloads/gradu.pdf. Luettu: 31.5.2010.

Shinners, P. 2002. Python Pygame Introduction. Www-dokumentti. Saatavissa: http://www.pygame.org/docs/tut/intro/intro.html. Luettu: 27.5.2010.

Sweigart, A. 2009. Invent Your Own Computer Games With Python. Pdf-tiedosto. Saatavissa: http://inventwithpython.com/IYOCGwP_book1.pdf. Luettu: 2.6.2010.

TIOBE Software. 2010a. TIOBE Programming Community Index for May 2010. Www-dokumentti. Saatavissa: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Luettu: 26.5.2010.

TIOBE Software. 2010b. TIOBE Programming Community Index Definition. Www-dokumentti. Saatavissa: http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm. Luettu: 26.5.2010.

Vesanen, A. 2006. Ohjelmointikielten historiaa. Oulun yliopisto, tietojenkäsittelytieteiden laitos. Www-dokumentti. Saatavissa: http://www.tol.oulu.fi/kurssit/okp/Luennot/OKP_Historia.html. Luettu 17.5.2010.

```
1.  #!/usr/bin/python
2.  # -*- coding: UTF-8 -*-
3.
4.  import pygame, sys
5.  from pygame.locals import *
6.
7.  # Pygamen alustus
8.  pygame.init()
9.
10. # Ikkunan luominen
11. #((leveys, korkeus), lisäasetukset, värisyvyys bitteinä)
12. Ikkuna = pygame.display.set_mode((250, 200), 0, 32)
13. pygame.display.set_caption('Erilaisia kuvioita')
14.
15. # Värit
16. BLACK = (0, 0, 0)
17. WHITE = (255, 255, 255)
18. RED = (255, 0, 0)
19. GREEN = (0, 255, 0)
20.
21. # Maalaa ikkunan taustan valkoiseksi
22. Ikkuna.fill(WHITE)
23.
24. # Ympyrä
25. # (kohde, väri, (sijainti x, sijainti y), säde, ulkoreunan paksuus)
26. pygame.draw.circle(Ikkuna, BLACK, (75, 100), 50, 0)
27.
28. # Suorakulmio
29. # (kohde, väri, (sijainti x, sijainti y, leveys, korkeus)
30. pygame.draw.rect(Ikkuna, RED, (20, 155, 200, 25))
31.
32. # Suorakulmainen kolmio
33. # (kohde, väri, (piste1, piste2, piste3), ulkoreunan paksuus)
34. pygame.draw.polygon(Ikkuna, GREEN, ((220,50),(220,125),(140,125)), 0)
35.
36. # Piirretään kaikki ylläoleva Ikkunaan
37. pygame.display.update()
38.
39. # Pelisilmukka
40. while True:
41.     for event in pygame.event.get():
42.         if event.type == QUIT:
43.             pygame.quit()
44. sys.exit()
```

```
1.  #!/usr/bin/python
2.  # -*- coding: UTF-8 -*-
3.
4.  import pygame, sys, time
5.  from pygame.locals import *
6.
7.  # Pygamen alustus
8.  pygame.init()
9.
10. # Luodaan ikkuna
11. leveys = 400
12. korkeus = 200
13. ruutu = pygame.display.set_mode((leveys, korkeus), 0, 32)
14. pygame.display.set_caption('Animaatio')
15.
16. # Värit
17. BLACK = (0, 0, 0)
18. WHITE = (255,255,255)
19.
20. #Ympyrän lähtöpaikka x-akselilla
21. x = -50
22.
23. # Pelisilmukka, joka liikuttaa valkoista ympyrää x-akselilla
24. while True:
25.     for event in pygame.event.get():
26.         if event.type == QUIT:
27.             pygame.quit()
28.             sys.exit()
29.     ruutu.fill(BLACK)
30.     pygame.draw.circle(ruutu, WHITE, (x, 100), 50, 0)
31.     x+=1
32.     if x>450:
33.         x=-50
34.     pygame.display.update()
35.     time.sleep(0.005)
```

```
1.  #!/usr/bin/python
2.  # -*- coding: UTF-8 -*-
3.
4.  import pygame, sys, time
5.  from pygame.locals import *
6.
7.  pygame.init()
8.
9.  # Luodaan ikkuna
10. leveys = 400
11. korkeus = 250
12. ruutu = pygame.display.set_mode((leveys, korkeus), 0, 32)
13.
14. pygame.display.set_caption('Reunoihin törmäilevä rantapallo')
15.
16. # Värit
17. BLACK = (0, 0, 0)
18.
19. nopeus = [2, 2]
20.
21. pallo = pygame.image.load("ball.gif")
22. ballrect = pallo.get_rect()
23.
24. while True:
25.     for event in pygame.event.get():
26.         if event.type == QUIT:
27.             pygame.quit()
28.             sys.exit()
29.
30.     ballrect = ballrect.move(nopeus)
31.
32.     if ballrect.left < 0 or ballrect.right > leveys:
33.         nopeus[0] = -nopeus[0]
34.     if ballrect.top < 0 or ballrect.bottom > korkeus:
35.         nopeus[1] = -nopeus[1]
36.
37.     ruutu.fill(BLACK)
38.     ruutu.blit(pallo, ballrect)
39.     pygame.display.flip()
40.     time.sleep(0.01)
```

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.
4.  import pygame, sys
5.  from pygame.locals import *
6.
7.  # Pygamen alustus
8.  pygame.init()
9.  mainClock = pygame.time.Clock()
10.
11. # Luodaan ikkuna
12. leveys = 640
13. korkeus = 360
14. screen = pygame.display.set_mode((leveys, korkeus),0,32)
15. pygame.display.set_caption('Lahna ja Siika')
16.
17. tausta = "meri.jpg"
18. background = pygame.image.load(tausta).convert()
19.
20. siika = pygame.image.load("siika.png")
21. rect1 = siika.get_rect()
22.
23. lahna = pygame.image.load("lahna.png")
24. rect2 = lahna.get_rect()
25.
26. # Luodaan liikemuuttujat
27. moveLeft = False
28. moveRight = False
29. moveUp = False
30. moveDown = False
31.
32. MOVESPEED = 8
33.
34. while True:
35.     for event in pygame.event.get():
36.         if event.type == QUIT:
37.             pygame.quit()
38.             sys.exit()
39.
40.         if event.type == KEYDOWN:
41.             if event.key == K_LEFT:
42.                 moveRight = False
43.                 moveLeft = True
44.             elif event.key == K_RIGHT:
45.                 moveLeft = False
46.                 moveRight = True
47.             elif event.key == K_UP:
48.                 moveDown = False
49.                 moveUp = True
50.             elif event.key == K_DOWN:
```

```
51.                                     moveUp = False
52.                                     moveDown = True
53.
54.                     if event.type == KEYUP:
55.                             # Esc-nappia paimalla ohjelma sammuu
56.                             if event.key == K_ESCAPE:
57.                                     pygame.quit()
58.                                     sys.exit()
59.                             if event.key == K_LEFT:
60.                                     moveLeft = False
61.                             elif event.key == K_RIGHT:
62.                                     moveRight = False
63.                             elif event.key == K_UP:
64.                                     moveUp = False
65.                             elif event.key == K_DOWN:
66.                                     moveDown = False
67.
68.             if moveDown and rect1.bottom < korkeus:
69.                     rect1.top += MOVESPEED
70.             if moveUp and rect1.top > 0:
71.                     rect1.top -= MOVESPEED
72.             if moveLeft and rect1.left > 0:
73.                     rect1.left -= MOVESPEED
74.             if moveRight and rect1.right < leveys:
75.                     rect1.right += MOVESPEED
76.
77.             sijainti = pygame.mouse.get_pos()
78.             # keskittää kursorin kalan puoliväliin
79.             sijx = sijainti[0]
80.             sijy = sijainti[1]
81.             sijx -= lahna.get_width()/2
82.             sijy -= lahna.get_height()/2
83.             rect3 = rect2.move(sijx,sijy)
84.
85.             pygame.mouse.set_visible(False)
86.
87.             screen.blit(background,(0,0))
88.             screen.blit(siika, rect1)
89.             screen.blit(lahna, rect3)
90.             pygame.display.update()
91.             mainClock.tick(40)
```

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.
4.  import pygame, sys, time, random
5.  from pygame.locals import *
6.
7.  # set up pygame
8.  pygame.init()
9.  mainClock = pygame.time.Clock()
10.
11. # Luodaan ikkuna
12. leveys = 640
13. korkeus = 360
14. screen = pygame.display.set_mode((leveys, korkeus),0,32)
15. pygame.display.set_caption('Leppäkerttu')
16.
17. tausta = "lehti.jpg"
18. background = pygame.image.load(tausta).convert()
19.
20. kerttu = pygame.image.load("bug.png")
21. rect1 = kerttu.get_rect()
22.
23. kirva = pygame.image.load("kirva_pieni.png")
24. otokkamaara = []
25. for i in range(20):
26.     otokkamaara.append(pygame.Rect(random.randint(0,leveys-64),
27.     random.randint(0,korkeus-49),64,49))
28.
29. laskuri = 0
30. syotavat = 60
31.
32. # Luodaan liikemuuttujat
33. moveLeft = False
34. moveRight = False
35. moveUp = False
36. moveDown = False
37.
38. MOVESPEED = 3
39.
40. # Ääniasetukset
41. pickUpSound = pygame.mixer.Sound("glups.wav")
42. pygame.mixer.music.load('wind.wav')
43. pygame.mixer.music.play(-1, 0.0)
44. musicPlaying = True
45. syodytOtukset = 0
46.
47. # Luodaan fontti
48. basicFont = pygame.font.SysFont(None, 28)
49.
50. BLACK = (0, 0, 0)
```

```
51.         WHITE = (255, 255, 255)
52.         RED = (255, 0, 0)
53.         GREEN = (0, 255, 0)
54.
55.         # pistemäärän sijainti
56.         textRect = (15,15,50,150)
57.
58.         # pelaajan pisteet String-muuttujana
59.         pistemaara = "0"
60.
61.         # Pelisilmukka
62.         while True:
63.             for event in pygame.event.get():
64.                 if event.type == QUIT:
65.                     pygame.quit()
66.                     sys.exit()
67.
68.                 if event.type == KEYDOWN:
69.                     if event.key == K_LEFT:
70.                         moveRight = False
71.                         moveLeft = True
72.                     elif event.key == K_RIGHT:
73.                         moveLeft = False
74.                         moveRight = True
75.                     elif event.key == K_UP:
76.                         moveDown = False
77.                         moveUp = True
78.                     elif event.key == K_DOWN:
79.                         moveUp = False
80.                         moveDown = True
81.
82.                 if event.type == KEYUP:
83.                     # Esc-nappia paimalla ohjelma sammuu
84.                     if event.key == K_ESCAPE:
85.                         pygame.quit()
86.                         sys.exit()
87.                     if event.key == K_LEFT:
88.                         moveLeft = False
89.                     if event.key == K_RIGHT:
90.                         moveRight = False
91.                     if event.key == K_UP:
92.                         moveUp = False
93.                     if event.key == K_DOWN:
94.                         moveDown = False
95.
96.
97.             if moveDown and rect1.bottom < korkeus:
98.                 rect1.top += MOVESPEED
99.             if moveUp and rect1.top > 0:
100.                rect1.top -= MOVESPEED
```

```
101.            if moveLeft and rect1.left > 0:
102.                    rect1.left -= MOVESPEED
103.            if moveRight and rect1.right < leveys:
104.                    rect1.right += MOVESPEED
105.
106.            # Piirrä tausta
107.            screen.blit(background,(0,0))
108.            # Piirrä leppäkerttu ruudulle
109.            screen.blit(kerttu, rect1)
110.
111.            laskuri += 1
112.            if laskuri >= syotavat:
113.                    # lisää ruokaa
114.                    laskuri = 0
115.            otokkamaara.append(pygame.Rect(random.randint(0,leveys-64),
116.            random.randint(0,korkeus-49),64,49))
117.
118.            # Tarkista onko leppäkerttu törmännyt kirvaan
119.            for f in otokkamaara[:]:
120.                    if rect1.colliderect(f):
121.                            otokkamaara.remove(f)
122.                            syodytOtukset = syodytOtukset+1
123.                            pistemaara = str(syodytOtukset)
124.
125.                            if musicPlaying:
126.                                    pickUpSound.play()
127.            text = basicFont.render(pistemaara, True, BLACK, WHITE)
128.
129.
130.            # piirrä kirvoja
131.            for f in otokkamaara:
132.                    screen.blit(kirva, f)
133.
134.            pygame.mouse.set_visible(False)
135.
136.            # piirrä pistemäärä ruudulle
137.            screen.blit(text, textRect)
138.
139.            pygame.display.update()
140.            mainClock.tick(60)
```