

Juha-Matti Ojala

PYTHONIN KÄYTTÖ PELIOHJELMOINISSA

Opinnäytetyö

KESKI-POHJANMAAN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

Joulukuu 2011

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Tekniikan ja liiketalouden yksikkö	Aika Joulukuu 2011	Tekijä/tekijät Juha-Matti Ojala
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi Pythonin käyttö peliohjelmoinnissa		
Työn ohjaaja DI Kauko Kolehmainen		Sivumäärä 39 + 28
Työelämäohjaaja		
<p>Työn tavoitteena oli tutkia, kuinka Python-ohjelmointikieltä käyttämällä voidaan kehittää OpenGL-piirtoa hyödyntävä tietokonepeli. Tarkoituksena ei ollut paneutua peliohjelmoinnin syvällisimpiin ongelmiin.</p> <p>Sovelluskehitys toteutettiin Python-ohjelmointikielellä. Käytettyjä laajennuksia olivat pygame ja PyOpenGL. Työ rajattiin työkalujen esittelyyn ja lopputuotokseen. Työkalujen esittely rajattiin pythonin, pygamen ja PyOpenGL:n esittelyihin. Lopputuotoksen esittely jaettiin suunnittelu- ja toteutusvaiheisiin.</p> <p>Tuloksena syntyi OpenGL-piirtoa käyttävän tietokonepelin prototyyppi. Johtopäätös oli, että Pythonin nopeus on riittävää peliohjelmointiin, mutta alemman tason ohjelmakoodia vaaditaan alustariippumattomuuden takaamiseksi.</p>		
Asiasanat peliohjelmointi, pygame, PyOpenGL, Python		

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	Date December 2011	Author Juha-Matti Ojala
Degree programme Information Technology		
Name of thesis Using Python in game programming		
Instructor Kauko Kolehmainen		Pages 39 + 28
Supervisor		
<p>The goal of this thesis was to research the possibilities of Python programming language in developing a computer game that utilizes OpenGL rendering. The goal was not to go deep into the more complex problems of the game programming field.</p> <p>Software development was done with the Python programming language, including a couple of necessary modules, such as pygame and PyOpenGL. The thesis was divided into introduction of the tools used in the development process and introduction of the final product. The introduction of the final product was further subdivided into planning and implementation phases.</p> <p>The final product that was developed for this thesis was a prototype of a basic game idea. The conclusion was that the Python programming language is sufficiently effective in game programming however, it may require an accompanying low-level programming language, such as the C-language for ensuring platform independence.</p>		
Key words game programming, pygame, PyOpenGL, Python		

TIIVISTELMÄ
ABSTRACT
SISÄLLYS

1 JOHDANTO	1
2 PYTHON	2
2.1 Perussyntaksi	2
2.2 Modulaarisuus	3
2.3 Moduuleja peliohjelmointiin	6
2.4 Virhe käsittely	7
3 PYOPENGL	9
3.1 Matriisit	9
3.2 Tekstuurit ja valaistus	10
4 PYGAME	11
4.1 Näyttöraja pinta	11
4.2 Tapahtumien käsittely	12
4.3 Prosessointinopeuden rajoitus	13
5 PROTOBALL	14
5.1 Vaatimukset	14
5.2 Peliobjektit	14
5.3 Graafiset tehosteet	15
5.3.1 Salamaefekti	15
5.3.2 Pallon törmäystarkistus ja pyöritys	17
5.4 Suunnittelu	18
5.4.1 Tasodatan formaatti	19
5.4.2 Luokkarakenne	19
5.4.3 Pelielementtien vuorovaikutus	22
5.5 Toteutus	26
5.5.1 Pelisilmukka	26
5.5.2 Alustukset	27
5.5.3 Syötteet ja tilapäivitykset	30
5.5.4 Piirto	32
5.5.5 Kenttäeditori	32
5.6 Tehokkuustestaus	35
6 YHTEENVETO	37
LÄHTEET	38
LIITTEET	

1 JOHDANTO

Python-ohjelmointi on ollut harrastukseni jo jonkin aikaa, joten tuntui luontevalta että opinnäytetyö käsittelee Python-ohjelmointia. Kun tähän lisäsi kiinnostuksen peliohjelmointia kohtaan, aihe alkoi hahmottua. Myös näytönohjaimen käyttö ja 3-ulotteinen piirto peliohjelmoinnissa olivat jo pidemmän aikaa askarruttaneet mieltäni, ja taustatutkimuksen jälkeen selvisi, että PyOpenGL tarjoaa rajapinnan juuri siihen tarkoitukseen.

Työn tarkoituksena on tutkia, kuinka Python-ohjelmointikieltä käyttämällä voidaan luoda grafiikkakiihdytystä käyttävä tietokonepeli. Rajapintana näytönohjaimen valjastamiseen ohjelmoijan käyttöön toimii PyOpenGL. Pelimoottorin perustarpeet, kuten näytön ja tapahtumien hallinnan, hoitaa pygame. Tarkoituksena on tehdä yksinkertaisesta peli-ideasta prototyyppiversio, jossa peliohjelmoinnin peruselementit ovat paikallaan. Tarkoitus ei ole tutkia tarkemmin peliohjelmoinnin alahaaroja. Tavoitteena on ymmärtää OpenGL:n toiminnallisuutta, kehittää ohjelmointitaitoja ja hankkia valmiuksia peliohjelmointiin.

Työ rajattiin työkalujen esittelyyn ja lopputuotokseen. Työn painotus on lopputuotoksen läpikäynnissä.

2 PYTHON

Python on proseduraalinen, funktionaalinen ja oliopohjainen ohjelmointikieli. Kieltä voidaan siis käyttää hyvin eri tavoilla, lyhyistä script-tyyppisistä ohjelmista laajempiin oliopohjaisiin sovelluksiin. Python on tulkattava. Sitä voidaan ajaa interaktiivisesti, käyttäen esimerkiksi IDLE ympäristöä, joka tulee Python-asennuspaketin yhteydessä.

Python on dynaaminen, eli muuttujalle ei määritellä tietotyyppiä, eikä sitä tarvitse esimääritellä. Pythonissa muuttujat ja funktiot ovat olioviitteitä. Käskeyllä `x = "A"` tehdään viittausoperaatio. Muistiin luodaan str-tyyppinen olio, johon `x` viittaa. Jos seuraavalla rivillä suoritetaan `x = "B"`, roskienkeruu vapauttaa muistin paikassa, missä `"A"` sijaisi, sillä siihen ei enää viittaa mikään. Is-operaattoria käyttämällä voidaan tarkastella muuttujien ominaisuuksia (KOODI 1). Kun kahteen muuttujaan sijoitetaan sama sisältö, ne viittaavat samaan muistialueeseen, paitsi kokoelmien yhteydessä.

```
>>> a = "hello"
>>> b = "hello"
>>> a is b
True
>>> a = (1, 2)
>>> b = (1, 2)
>>> a is b
False
```

KOODI 1. Is-operaattori

Pythonissa tietotyyppien muistitilaa ei ole rajattu, vaan muistia varataan sitä mukaa kuin muuttujan tilavaatimus kasvaa. (Summerfield 2010, 15–23; Hall & Stacey 2009, 17; Python Community 2011.)

2.1 Perussyntaksi

Yhteenlaskuoperaattori on Pythonin kattavin aritmeettinen operaattori, sillä se toimii numeroiden, merkkijonojen ja kokoelmien yhteydessä (KOODI 2).

```
>>> print "a" + "b", 5+2, [1,2,3] + [4,5,6]
ab 7 [1, 2, 3, 4, 5, 6]
```

KOODI 2. Yhteenlaskuoperaattori

Yhdistettyä sijoitus- ja aritmeettisoperaattoria käytetään inkrementointiin, koska Pythonissa ei ole muista ohjelmointikielistä tuttua inkrementaatio-operaattoria. Tällöin muuttujan sisältöä ei muuteta, vaan muuttujaolio viittaa uuteen muistialueeseen. Potenssioperaattorilla voidaan suorittaa sekä potenssilaskut että neliöjuuren ottamisen, ilman ulkoisten kirjastojen tuomista ohjelmaan, mikä on eroavaisuus monista muista ohjelmointikielistä. (Summerfield 2010, 31; Hall & Stacey 2009, 54.)

Pythonissa ei ole C-kielestä tuttua switch-case-haarautumisominaisuutta vaan kaikki haarautuminen tehdään if-lauseilla. Elif-käsky vastaa C:n else if -käskyä. Elif-käsky ajetaan, mikäli yksikään aikaisempi if- tai elif-käsky ei ole toteutunut. (Summerfield 2010, 27.)

C-kielessä for-silmukalla voi ajaa tietyn määrän kierroksia, mutta Pythonissa for-silmukka iteroi läpi kokoelman olioita (KOODI 3) ja suorittaa aliohjelmanlohkon yhtä monta kertaa kuin kokoelmassa on alkioita. Jokaisella kierroksella for ja in avainsanojen välissä määritelty muuttuja viittaa uuteen sekvenssissä olevaan alkioon. Tietyn kierrosmäärän voi käydä läpi käyttämällä range-funktiota, joka palauttaa listan kokonaislukuja. (A Byte of Python 2009, 40–41.)

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]: print i
```

KOODI 3. For-silmukka

2.2 Modulaarisuus

Pythonissa funktiomäärittelyyn käytetään def-avainsanaa, jota seuraa funktion nimi ja parametrikenttä. Kun funktio on määritelty, sitä voidaan kutsua. Koska Python on dynaaminen, voi argumentteina olla mitä tahansa muuttujatyyppejä. Funktiomäärittelyssä voidaan käyttää oletusarvoja. Oletusparametrit tulee aina

määritellä muiden parametrien jälkeen. Funktiokutsun yhteydessä ei ole tällöin pakko määritellä kaikkia argumentteja.

Funktion parametrimäärittelyyn voidaan lisätä myös lista tai sanakirja, joilla on mielivaltainen pituus (KOODI 4). Ne tulee aina määritellä aivan viimeisenä. Listalle käytetään syntaksia `*args` ja sanakirjalle `**kwargs`. Funktiokutsun yhteydessä voidaan tällöin käyttää listaa, jolla on mielivaltainen pituus. Mielivaltaista argumenttien määrää käyttämällä, voidaan funktioista tehdä huomattavasti geneerisempiä. (Summerfield 2010, 173–175.)

```
#avg function
def avg(*args):
    return 1.*sum(args) / len(args)
>>> avg(5,73,26,3,45)
30.399
```

KOODI 4. Funktion parametrien mielivaltainen määrä

Koristelijafunktiolla voidaan muuttaa jonkin funktion suorittamaa ohjelmakoodia muuttamatta funktiota itseään. Tämä on mahdollista, koska Pythonissa funktiot ovat olioita. Funktiota voidaan käsitellä muuttujan tavoin ja tehdä funktiokutsu myöhemmin (KOODI 5).

```
def decorate(f):
    def wrapper(*args):
        print "before function"
        res = f(*args)
        print "after function"
        return res
    return wrapper
@decorate
def hello(): print "hello"

>>> hello()
before function
hello
after function
```

KOODI 5. Koristelijafunktio

`@decorate` on lyhenne operaatiosta `hello = decorate(hello)`. Decorate-funktion sisällä on määritelty toinen funktio, jossa alkuperäiseen funktioon voidaan lisätä

ohjelmakoodia. Koska Pythonin parametrikentissä voidaan käyttää mielivaltaista argumenttien määrää, koristelijafunktiolla voidaan muokata mitä tahansa toista funktiota. Koristelija voi olla myös luokka, jolloin koristelijalle voi syöttää argumentteja, tehden siitä geneerisemmän. (Summerfield 2010, 356–360.)

Pythonissa luokka määritellään syntaksilla `class Person`. Luokasta voidaan ottaa instanssi `person = Person()` sijoituksella. Luokalla on jäsenmuuttujia, attribuutteja, jotka ovat normaaleja muuttujia, mutta kuuluvat luokalle. Luokan konstruktorilla määritellään olion luonnin yhteydessä tehtävät alustukset.

Pythonissa luokilla on erityismetodeita, joilla voidaan tehdä mm. attribuuttien alustus ja operaattorien ylimäärittelyt (KOODI 6). Metodit toimivat funktioiden tavoin, mutta luokan sisällä, muokaten tai käyttäen luokan omia attribuutteja. Metodien ja konstruktorin määrittelyssä ensimmäinen parametri on aina olioon itseensä viittaava muuttuja. Kun luokka on määritelty, se perii tietyt erityismetodit object-luokalta. Kun luokasta otetaan instanssi, ensimmäiseksi Python ajaa `__new__()`-käskyn ja sen jälkeen konstruktorin `__init__()`. Operaattorien ylimäärittelyyn eli olioiden vertailuun on olemassa liuta valmiita erityismetodeita. Person-luokan olioiden vertailua ei ole määritelty, ennen kuin vertailulle luodaan säännöt. Esimerkiksi `person1 + person2` -vertailun tekeminen vaatii yhteenlaskuoperaattorin ylimäärittelyn. Siihen on olemassa `__add__(self, other)` -käsky. (Payne 2010, 92; Summerfield 2010, 239–242.)

```
def __add__(self, other):
    return self.age + other.age
```

KOODI 6. Yhteenlaskuoperaattorin ylimäärittely

Funktioiden ja luokkien avulla voidaan koota ohjelmakoodia asiakokonaisuuksiksi. Vastaavasti funktioita voidaan koota moduuleihin, mitkä ovat käytännössä .py-päätteisiä tiedostoja. Moduuleita voidaan kirjoittaa myös muilla kielillä, kuten C-kielellä, jolla Pythonin ydin on kirjoitettu. Käytännössä moduulin tarkoitus on tarjota jokin tietty palvelu tai kokonaisuus, kuten Pythonin standardikirjastossa oleva math-moduuli tarjoaa runsaasti funktioita matemaattiseen laskentaan. Moduuli tuodaan pääohjelmaan käyttämällä

import-käskyä (KOODI 7). Import-käskyllä on useita eri muotoja. (Summerfield 2010, 196.)

```
import math
import math, random
import math as m
from math import *
```

KOODI 7. Import-syntaksi

2.3 Moduuleja peliohjelmointiin

Tässä alaluvussa käydään läpi muutama peliohjelmoinnin kannalta keskeinen moduuli.

Time-moduuli tarjoaa muutaman hyödyllisen työkalun, kuten ajastimen, unix-aikaleiman ja funktioita ajan tulostamiseen. Clock-funktio on hyödyllinen työkalu, kun halutaan optimoida koodinpätkää, funktiota tai silmukkaa, sillä se palauttaa ohjelman käynnistämisestä kuluneen ajan sekunneissa. (Python Software Foundation c 2011.)

Satunnaislukujen generointi on usein keskeisessä asemassa lähes missä tahansa tietokonepelissä. Pythonin standardikirjastossa on random-moduuli, joka tarjoaa satunnaislukujen generoinnin (KOODI 8.). Useimmat moduulin funktioista perustuvat random-funktioon, joka palauttaa liukuluvun väliltä [0.0, 1.0]. (Python Software Foundation a 2011.)

```
>>> random.random()
0.80802979158039956
>>> random.randint(1,10)
10
```

KOODI 8. Satunnaislukujen generointi

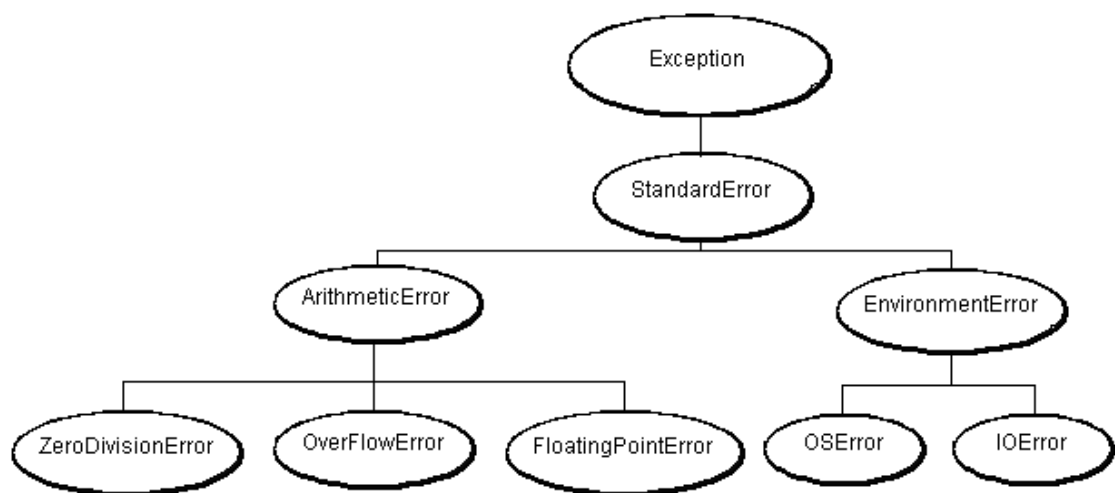
Os-moduuli tarjoaa rajapinnan käyttöjärjestelmän tiedostojärjestelmään. Peliohjelmoinnin kannalta yksi os-moduulin käytännöllisimmistä funktioista on getcwd(), joka palauttaa hakemistopolun, mistä ohjelmaa ajetaan. Pelin

datatiedostot voidaan siis löytää dynaamisesti, vaikka sen hakemistopolkua muokattaisiin. (Python Software Foundation b 2011.)

2.4 Virhekäsittely

Virhekäsittelyllä pyritään välttämään ohjelman kaatumista viimeiseen saakka nappaamalla poikkeuksia. Kaikki fataalit virheet tulee pyrkiä käsittelemään. Suurimpia ongelmia ohjelmoinnissa ovat käyttäjälähtöiset poikkeukset, olivat ne sitten tahallisia tai tahattomia. Aina kun sallii käyttäjän mukauttamaan ohjelmaa, tulee miettiä, mitä kaikkia poikkeuksia se saattaa aiheuttaa.

Pythonissa virheiden käsittelyyn on olemassa hierarkkinen luokkajärjestelmä (KUVIO 1). Virhetilanteessa Python nostattaa virheen, joka voidaan napata käyttämällä try... catch-käskyä.



KUVIO 1. Virhekäsittelyn luokkahierarkkia.

Tyypillisimpiä tapauksia virhekäsittelystä on tiedoston luku tai tiedostoon kirjoittaminen (KODI 9). Ei voida olettaa, että tiedosto on olemassa tai että luku- tai kirjoitusoikeudet ovat olemassa, joten on käytettävä virhekäsittelyä. (Summerfield 2010, 164.)

```
try:  
    f = open(filename, 'r')  
    data = f.read()  
    f.close()  
except IOError, error: print error
```

KOODI 9. Virhekäsittely.

3 PYOPENGL

OpenGL valjastaa näytönohjaimen resurssit ohjelmoijan käyttöön. Näytönohjaimessa on omat muistipiirinsä, jotka nopeuttavat 3D-piirtoa huomattavasti. OpenGL sisältää käskyjä ja pitkän listan eri tiloja, jotka määrittelevät sen, millä tavalla 3-ulotteisia objekteja ja kuvia muunnetaan pikseleiksi näyttöruudulle. Yhden tilan muuttaminen ei vaikuta toisiin tiloihin, vaan kaikkiin käskyihin, joilla on jokin yhteys kyseiseen tilaan. (Rost 2006.)

OpenGL:n kehitys alkoi 80-luvulta, kun todettiin, että grafiikkapiireille ei ollut olemassa standardia, vaan ohjelmoijat joutuivat kirjoittamaan rajapintoja uudelleen jokaista grafiikkapiiriä varten. OpenGL:n avulla haluttiin standardisoida grafiikkapiirien käyttö ja siten helpottaa ohjelmoijien työtaakkaa. OpenGL:n takana on Silicon Graphics. Nykyään OpenGL:n kehitystä kontrolloi OpenGL Architecture Review Board (ARB), johon kuuluu yrityksiä, kuten SGI, Intel, NVIDIA, ATI, IBM, Apple, 3Dlabs ja Sun Microsystems. (Rost 2006.)

3.1 Matriisit

3D-ohjelmoinnin kulmakivi on matriisi, erityisesti 4×4 -matriisi. Matriisit yksinkertaistavat 3-ulotteisen illuusion luomisen 2-ulotteiselle näyttöpinnalle. 4×4 -matriiseilla voidaan tehdä skaalaus- ja pyörähdysoperaatiot objekteille tai kameraprojektiolle. Jotta 3-ulotteinen avaruus voidaan näyttää 2-ulotteisella pinnalla, tarvitaan koordinaatistomuunnos. OpenGL:n runkoon kuuluvat GL_MODELVIEW ja GL_PROJECTION -pinomatriisit, joissa GL_MODELVIEW sisältää 3-ulotteisen objektiavaruuden ja GL_PROJECTION 2-ulotteisen muunnoksen.

Muunnosmatriisin avulla voidaan siirtää objekteja 3-ulotteisessa avaruudessa. Ilman muunnoksia kaikki objektit sijaitsevat origossa. Objektin paikkavektori summataan GL_MODELVIEW-matriisin 4. rivin komponenttien kanssa. OpenGL-käsky muunnokselle on `glTranslatef(x, y, z)`. Kaikki piirtotapahtumat kyseisen operaation jälkeen on muunnettu. Käskyjä `glPushMatrix()` ja `glPopMatrix()`

käyttämällä, muunnoksen voi rajoittaa haluttuihin objekteihin. (McGugan 2007, 185.)

Objektia voidaan pyöräyttää määritetyn akselin ympäri käyttämällä `glRotate`-käskyä. Y-akselin ympäri pyöräytys 180 astetta voidaan tehdä käskyllä `glRotatef(180, 0, 1, 0)`. Jos jotakin objektia halutaan sekä siirtää että pyöräyttää, matriisimuunnokset voidaan tehdä joko erikseen tai matriisit voidaan kertoa keskenään ja tehdä vain yksi muunnos. Lopputulos on sama, kunhan kertominen tehdään oikeassa järjestyksessä.

Projektiomatriisi toimii kamerana. 3-ulotteisen maailman näyttäminen näytöruudulla vaatii tarkastelupisteen ja kulman. Kameraprojektio on 4 x 4 -matriisi, missä ensimmäiset kolme riviä sisältävät x-, y- ja z-komponentit, eli katselukulman. Neljäs rivi sisältää kamerasijainnin. (McGugan 2007, 181–186.)

3.2 Tekstuurit ja valaistus

Kun 3-ulotteinen objekti on mallinnettu, voi se olla joko täytetty tai läpinäkyvä. Tekstuurit on tekniikka, jolla voidaan ladata kuva objektin pinnalle, jotta se näyttäisi realistisemmalta. Tekstuurin alustamiseksi asetetaan sille identiteetti käskyllä `glGenTextures`, jonka jälkeen näytönohjaimelle voidaan syöttää kuvadata ja tekstuurin ominaisuudet identiteettikohtaisesti käyttämällä `glBindTexture`-käskyä. Kuvadatan syöttö tehdään `glTexImage2D`-käskyllä, jonka parametreissa syötetään mm. tekstuurin korkeus ja leveys, data ja sen formaatti. Datan syödon jälkeen tekstuurille voidaan asettaa erityisominaisuuksia ja -parametreja `glTexParameter`-käskyllä. (McGugan 2007, 235–248.)

Valaistuksen käyttö pelissä on olennainen osa oikeanlaisen tunnelman luomisessa. OpenGL hyödyntää piirrettävän objektin normaaleja sen valaisemiseen. Kun puhutaan valaistuksesta, käytännössä se tarkoittaa värien prosessointia objektin pinnalla. Valaistus otetaan käyttöön käskyllä `glEnable` (`GL_LIGHTING`). Valon ominaisuuksia määritellään `glLight`-käskyllä, jonka

parametreja voivat olla mm. valon paikka, intensiteetti ja diffuusio. (McGugan 2007, 263–266.)

4 PYGAME

Pygame on Simple DirectMedia Layer -kirjastoon perustuva Python-paketti, joka tarjoaa pelimoottorin perustarpeet, kuten tapahtumien hallinnan, pelisilmukan ajoituksen, näytön alustuksen ja näytölle piirtämisen. Paketti sisältää joukon moduuleita (TAULUKKO 1), jotka joko tarjoavat työkaluja tai toimivat rajapintoina laitteistolle.

TAULUKKO 1. Pygame-moduulit. (McGugan. 2007)

Moduuli	Kuvaus
pygame.display	Näyttörajapinta
pygame.draw	Piirtotyökalut
pygame.event	Tapahtumat
pygame.font	Tekstipiirto
pygame.image	Kuvapiirto
pygame.key	Näppäimistörajapinta
pygame.mixer / pygame.music	Äänirajapinta
pygame.mouse	Hiirirajapinta
pygame.sprite	Liikkuvien peliobjektien hallintatyökalut
pygame.surface	Kuvien ja näytön välinen rajapinta
pygame.time	Ajan ja prosessointinopeuden hallinta

4.1 Näyttörajapinta

Display-moduuli sisältää työkalut näytön ohjaamiseen. set_mode-funktiossa ajetaan alustustoimenpiteet ja palautetaan surface-olio. Set_mode-funktiolla palautettu pinta on koko näytön kattava kuva, jota päivitetään pelisilmukan lopussa. Ohjelmassa voi olla vain yksi set_mode-funktiolla palautettu pinta. Muut surface-tyyppiset oliot kopioidaan blit-funktion avulla näyttöpintaan.

Näyttöpinta voidaan alustaa useissa eri moodeissa. Set_mode-funktion argumenteissa määritellään moodi lippujen avulla. Tuplapuskurointia varten on olemassa DOUBLEBUF- ja PyOpenGL-piirtoa varten OPENG- lippu (KOODI 10).

```
self.screen = pygame.display.set_mode(SCREEN_SIZE,  
HWSURFACE|OPENG-|OPENG-|DOUBLEBUF)
```

KOODI 10. Näyttöpinnan alustus

Pelisilmukan lopussa näytön päivittämiseen on olemassa sekä flip- että update-funktiot. Flip-funktiolla voidaan päivittää koko näyttö ja update-funktiolla määritelty osa siitä. Update-funktio soveltuu flip-funktiota paremmin piirtoon, jossa näytöllä liikkuvia osia on vähän. Flip-funktiota käytetään PyOpenGL-piirtoon. (Pygame Documentation a. 2011.)

4.2 Tapahtumien käsittely

Syötteiden ja tulosten hallinta on keskeisimpiä asioita peliohjelmoinnissa. Pygamessa syötteiden hallintaa hoitaa pääosin event-moduuli. Jos hiirellä tai näppäimistöllä tehdään jotain, tapahtumat siirtyvät tapahtumajonoon. Joitakin mahdollisia tapahtumia ovat hiiren liike, hiiren näppäinten painaminen ja näppäimistön painaminen. Ohjelmoija voi myös itse määritellä haluamansa tapahtumatyyppin käyttämällä event-luokkaa.

Tapahtumaolioilla on attribuutteja, jotka kertovat tapahtumien ominaisuuksista tarkemmin. Tapahtumaolion pos-attribuutti sisältää x- ja y-koordinaatin. Pelisilmukan jokaisella kierroksella tarkistetaan, onko tapahtumajonoon kertynyt tapahtumia, event.get-funktiota käyttäen. Get-funktio palauttaa listan tapahtumista ja tyhjentää jonon. (Pygame Documentation b. 2011.)

Äänillä voidaan tehostaa pelikokemusta. Äänten ulostulo voidaan toteuttaa mixer-moduulilla. Toiminta perustuu kanaviin, Channel-olioihin, joissa toistetaan Sound-olioita. Sound-olioihin ladataan äänitiedosto (KOODI 11). Mixer-moduuli alustetaan toistamaan tiettytyyppisiä äänitiedostoja. Käyttämällä play- ja stop-

metodeita, äänentoisto voidaan aloittaa ja pysäyttää. (Pygame Documentation c. 2011.)

```
self.mixer = pygame.mixer  
self.mixer.init(44100,16,1,2048)  
self.ballrotatesound = self.mixer.Sound('ballroll.ogg')  
self.channel = pygame.mixer.Channel(1)
```

KOODI 11. Äänimoduulin alustus

4.3 Prosessointinopeuden rajoitus

Prosessointinopeutta tulee rajoittaa, jotta peli toimii samalla nopeudella eri alustoilla. Prosessointinopeutta voidaan rajoittaa asettamalla pelisilmukalle päivitysrajoitus. Tämä lähestymistapa ei ota huomioon eri alustojen prosessointinopeutta. Toinen tapa toteuttaa prosessointinopeuden rajoitus on suhteuttaa liike aikaan. Asetetaan kaikille liikkuville osille nopeus (nopeusyksikköä/sekunti). Nopeus kerrottuna ajanmuutoksella tekee liikkeestä yhtä nopeaa kuvanpäivitysnopeudesta riippumatta.

Time-moduuli tarjoaa työkalut prosessointinopeuden rajoittamiseen. Clock-luokan tick-metodille voidaan syöttää argumenttina haluttu kuvanpäivitysnopeus, ja se palauttaa millisekunneissa ajan, joka päivitykseen käytetään. (Pygame Documentation. 2011 d; McGugan. 2007, 43–61.)

5 PROTOBALL

Protoball on puzzle-tyyppisen pallopelin prototyyppi, jonka tuottamiseksi käsitellään peliohjelmoinnin perusasioita ja edellisissä luvuissa mainittuja asioita.

5.1 Vaatimukset

Sovellus ei ole kaupallinen, eikä sitä ole tarkoitettu levitettäväksi. Sovelluksen tulee toimia ainakin Windows XP- ja 7 –käyttöjärjestelmissä, ja käyttäjän näytönohjaimessa on oltava grafiikkakiihdytin. Sovelluksen on oltava luotettava ja resurssitehokas. Latausaikojen tulee olla mahdollisimman lyhyitä, eikä sovelluksessa saa olla muistiylivuotoja. Mahdolliset virheet ja poikkeukset tulee käsitellä sovelluksen sisällä.

Peli koostuu tasoista joita pelaajan tulee läpäistä. Tasot koostuvat 20 x 20 -ruudukosta, missä yksi ruutu voi olla joko tyhjä tai sisältää objektin. Objekteja ovat pallon aloituspiste, suunnanmuuttaja, pysäyttäjä, pyöräyttäjä ja maali. Tason läpäisemiseksi pelaajan tulee liikuttaa pallo alkupisteestä loppupisteeseen.

Sovelluksen tulee olla puzzle-tyyppinen ongelmanratkomispeli, jossa muutamalla objektilla, jotka toimivat yksinkertaisin säännöin, luodaan monimutkaisia ja vaikeasti ratkottavia yhdistelmiä. Peli-idean toteuttamiseen käytetään 3D-grafiikkaa ylhäältä päin kuvattuna.

5.2 Peliobjektit

Aloituspiste määrittelee pallon alkupisteen 20 x 20 -ruudukossa. Aloituspiste-objekteja voi olla vain yksi, joten niiden määrä tulee virhekäsitellä. Pallon nopeus on vakio, ilman kiihtyvyyssvaihetta. Nopeuden suunta voi olla yksi neljästä, x- tai y-akselien suuntaisesti. Näytöllä origo on vasemmassa

yläkulmassa. Kun suunta on valittu, pallo jatkaa matkaansa törmäillen suunnanmuuttajiin, kunnes se kohtaa pysäyttäjän tai lopetuspisteen.

Suunnanmuuttaja on suorakulmainen kolmio, joka kääntää pallon suuntaa 90 astetta, tai 180 astetta, jos törmäys on kohtisuora.

Pysäyttäjä pysäyttää pallon ja antaa pelaajalle mahdollisuuden valita pallolle uusi suunta. Pysäyttäjä on pelin tärkein objekti, sillä se lisää kombinaatioiden määrää.

Pyöräyttäjä pyöräyttää kaikkia suunnanmuuttajia 90astetta. Toteutus on sopivan yksinkertainen tasodatan tallennuksen kannalta. Toisessa toteutusmallissa pyöräyttäjällä voisi pyöräyttää 180 ja 270astetta vain tiettyjä, pyöräyttäjään linkitettyjä suunnanmuuttajia.

5.3 Graafiset tehosteet

Pelissä graafiset tehosteet ovat usein tärkeässä asemassa oikeanlaisen tunnelman luomisessa. Peliä suunnitellessa ohjelmoijalla voi olla paljon hyviä ideoita, mutta ei täyttä varmuutta niiden toteutuksesta, jolloin vaaditaan esitutkimusta. Tässä alaluvussa käsitellään joitakin pelissä käytettyjä tehosteita ja algoritmeja.

5.3.1 Salamaefekti

Salamaefekti voidaan luoda yksinkertaisella algoritmilla, kun salaman alku- ja loppupiste tunnetaan. Algoritmi perustuu suoran pirstalointiin (KODI 12). Ensimmäinen suora lisätään listaan. Listassa olevat suorat jaetaan kahteen yhtä pitkään suoraan, joita yhdistää piste, joka on alkuperäisen suoran keskipisteestä kohtisuorasti siirretty satunnaiseen suuntaan (KUVIO 2).

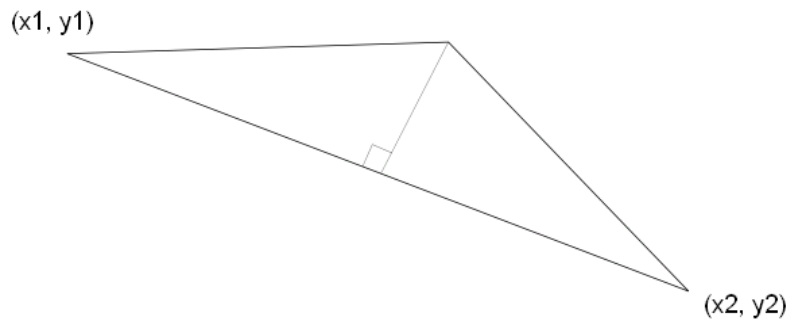
```
#Pseudo-koodi  
for i in range(iterations)
```

```

for segment in segments:           #Käydään läpi segmentit
    segments.remove(segment)        #Pirstaloitava segmentti poistetaan
    calculate segment mid_point     #Lasketaan segmentin keskipiste
    randomize offset amount         #Satunnainen keskipisteen siirtymä
    offset mid_point perpendicularly #Siirretään keskipistettä kohtisuorasti
    s1 = Segment(segment.start, mid_point) #Luodaan uudet segmentti-oliot
    s2 = Segment(mid_point, segment.end)
    segments.append(s1)             #lisätään uudet segmentit listaan
    segments.append(s2)
    if i<2:
        calculate branch_end        #Lasketaan haaran loppupiste
        b = Segment(segment.start, branch_end) #Lisätään haara-segmentti listaan
    max_offset *= 0.5               #Siirtymän maksimin puolitus

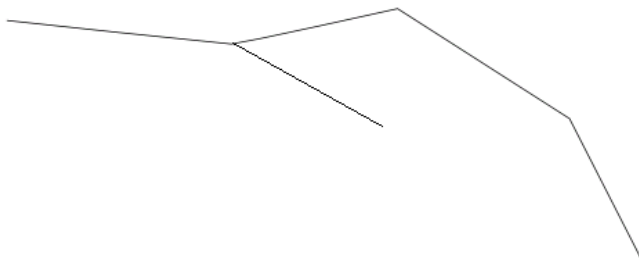
```

KOODI 12. Salama-algoritmin pseudokoodi



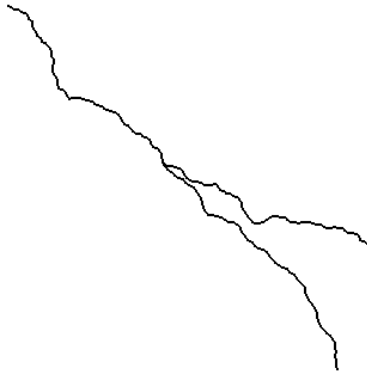
KUVIO 2. Suoran paloittelu

Toisella kierroksella jaettavia suoria on kaksi (KUVIO 3). Kierroksia jatketaan, jolloin salaman muoto alkaa hahmottua. Ensimmäisillä kierroksilla lisätään yksi tai useampia haaroja, joiden alkupiste on siirretty suoran keskipiste ja loppupiste on suoran pituisen matkan päässä alkuperäisen suoran suuntaan.



KUVIO 3. Haaran lisäys

Lopuksi generoidut suorat piirretään (KUVIO 4).



KUVIO 4. Mahdollinen lopputuotos

Algoritmi on tarpeeksi kustannustehokas, sillä yksinkertaisen salaman generointiin kuluu alle millisekunti. Jos haaroja ja iterointeja on paljon, saman kuvanpäivityksen aikana ei kannata generoida montaa salamaa. (Drillian's House of Game Development 2009.)

5.3.2 Pallon törmäystarkistus ja pyöritys

Pallon törmäystarkistukseen käytetään yksinkertaista hitbox-menetelmää. Suunnanmuuttajille määritellään kaksi neliötä, ulompi ja sisempi (KUVIO 5), jotka erittelevät kohtisuorat ja 90asteen törmäykset. Kun pallo leikkaa ulomman neliön reunan, tarkistetaan pallon suunta ja suunnanmuuttajan asento. Jos törmäys ei ole kohtisuora, pallon annetaan jatkaa matkaa, kunnes se leikkaa sisemmän neliön reunan (KODI 13).

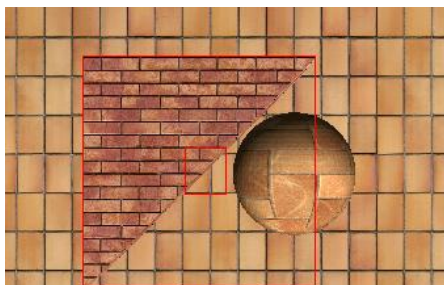
```
#Pseudo-koodi
for obj in rotators:
    if obj is not previously_collided_object:
        if sphere intersects outer square:
            if collision is orthogonal:
                ball.v *= -1
                previously_collided_object = obj
```

```

else if sphere intersects inner square:
    modify ball.v based on obj alignment
    previously_collided_object = obj

```

KOODI 13. Pallon törmäystarkistuksen pseudokoodi



KUVIO 5. Pallon törmäystarkistus

Kun pallo liikkuu pinnalla, sen tulee pyöriä. Pyörimisen grafiikka joudutaan toteuttamaan erikseen. Tässä tapauksessa peli-idea on sen verran yksinkertainen, että pallon pyörittämiseen vaadittava matematiikka ja logiikka eivät ole liian vaativia. Jokaisen kuvanpäivityksen aikana palloa pyöritetään x- ja z-akseleidensa ympäri. Kulman suuruus voidaan laskea pallon paikkavektorista. Kun pallo on matkustanut yhtä pitkän matkan kuin pallon piirin pituus, kulma on 360 astetta. Kun kulmat on laskettu, käytetään `glRotatef`-käskyä (KOODI 14). On huomioitava, että OpenGL:ää käytettäessä kaikki objektit sijaitsevat origossa, ennen kuin ne siirretään muunnosmatriisilla. (McGugan 2007, 186.)

```

x_angle += v.x/(r*2*pi / 360) *td
z_angle += v.z/(r*2*pi / 360) *td
glRotatef(self.x_angle, 0.0, 0.0, 1.0) #Z-akselin ympäri
glRotatef(self.z_angle, 1.0, 0.0, 0.0) #X-akselin ympäri

```

KOODI 14. Pallon pyöritys

5.4 Suunnittelu

Suunnitteluun käytetään Unified Modelling Language (UML) luokka-, sekvenssi- ja tilakaavioita.

5.4.1 Tasodatan formaatti

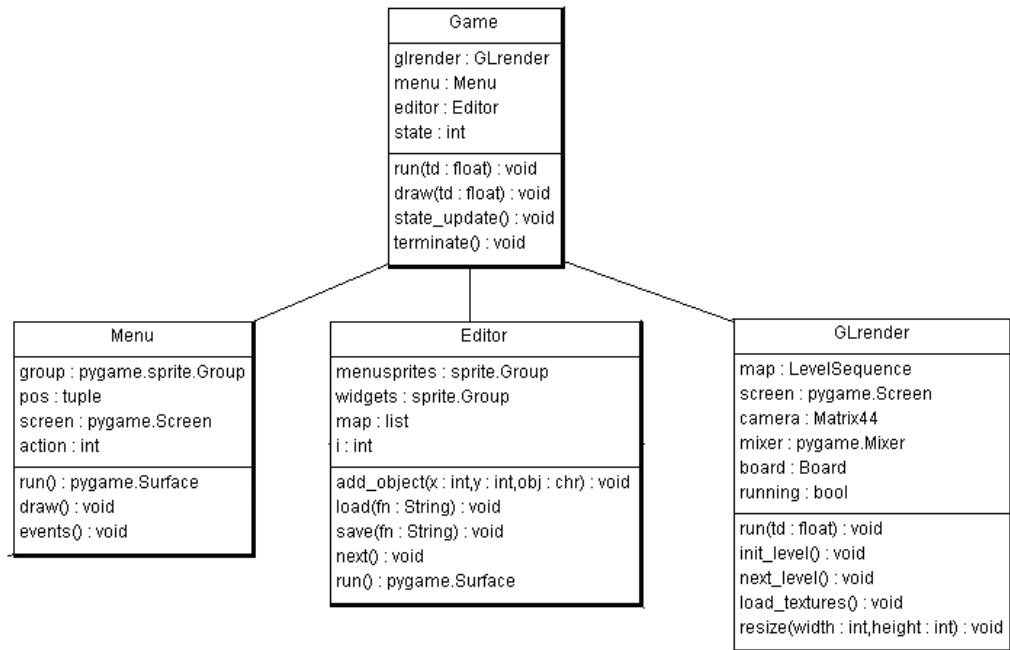
Tasot tallennetaan tekstitiedostoihin, joissa tietyt merkit edustavat tiettyjä peliobjekteja (TAULUKKO 2). Ensimmäinen taso alkaa tiedoston ensimmäiseltä riviltä, jonka jälkeen seuraavat 20 riviä, 20 merkkiä per rivi kuuluvat kyseiseen tasoon. Tasot on eritelty tyhjällä rivillä.

TAULUKKO 2. Tasodatan formaatti

Merkki	Objekti
0	Tyhjä ruutu
1, 2, 3 ja 4	Suunnanmuuttaja ja sen eri asennot
s	Alku
e	Loppu
q	Pysäyttävä
r	Pyöräyttävä

5.4.2 Luokkarakenne

Game-luokka on pelin pääluokka (KUVIO 6). Se sisältää pygamen ja muiden luokkien alustukset, pelin tilan ja näytön päivityksen. Game-luokan tila määrittelee, missä tilassa peli on. Eri tiloja on kolme, menu-tila, play-tila ja editor-tila. Jos peli on menu-tilassa, niin menu-luokan koodia ajetaan ja se palauttaa näyttöpinnan game-luokalle piirrettäväksi. Vastaavasti jos peli on play-tilassa, PyOpenGL-renderöintiluokan koodia ajetaan ja palautetaan näyttöpinta game-luokan piirrettäväksi, jolloin myös näyttörajaus on alustettu OpenGL:ää varten.



KUVIO 6. Luokkakaavio 1

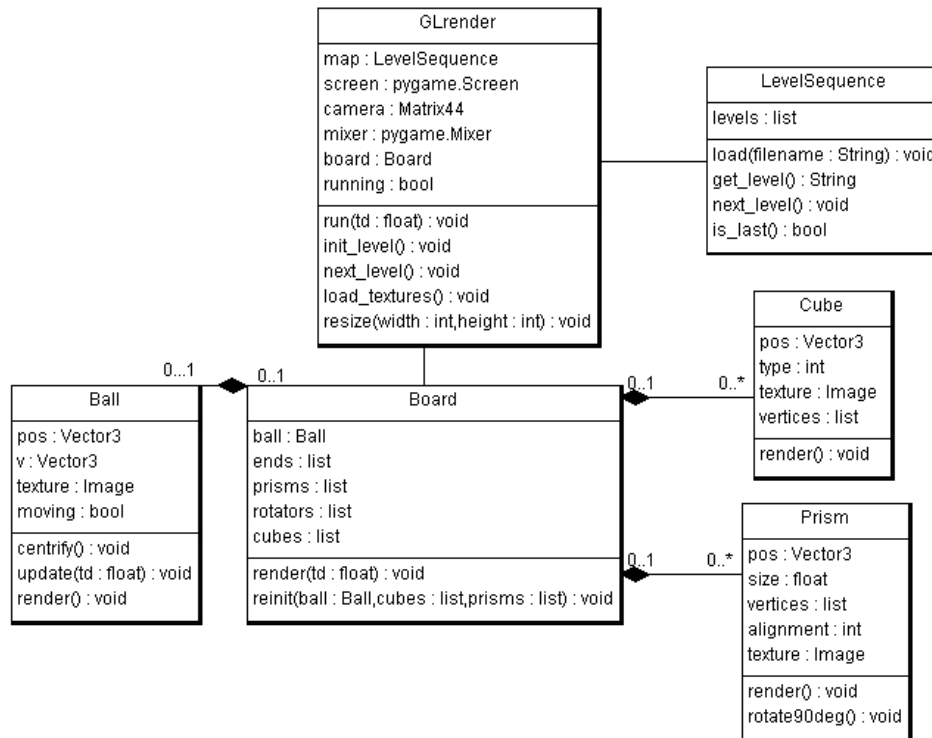
Glrender-luokka vastaa play-tilan koodista, joka koostuu enimmäkseen OpenGL-käskyistä. Glrender-luokka sisältää board-luokan, joka taasen sisältää listat kaikista peliobjekteista, joita ovat cube-, prism- ja ball-luokka.

Cube-luokka on näytölle piirrettävä kuutio, joka vastaa yhtä ruutua. Kuutiolla voi olla kolmea eri tyyppiä, tyhjä ruutu, pysäyttävä tai pyöräyttävä. Eri kuutiotyypeille ladataan eri tekstuurit, jolloin ne erottuvat toisistaan. Kuutio voidaan piirtää kutsumalla cube-luokan render-metodia, joka sisältää pääosin OpenGL-käskyjä.

Suunnanmuuttajia varten on prism-luokka. Se sisältää render-funktion lisäksi rotate-metodin, jossa muutetaan suunnanmuuttajan asentoa. Kyseistä metodia kutsutaan, kun pallon törmäys pyöräyttäjään huomataan.

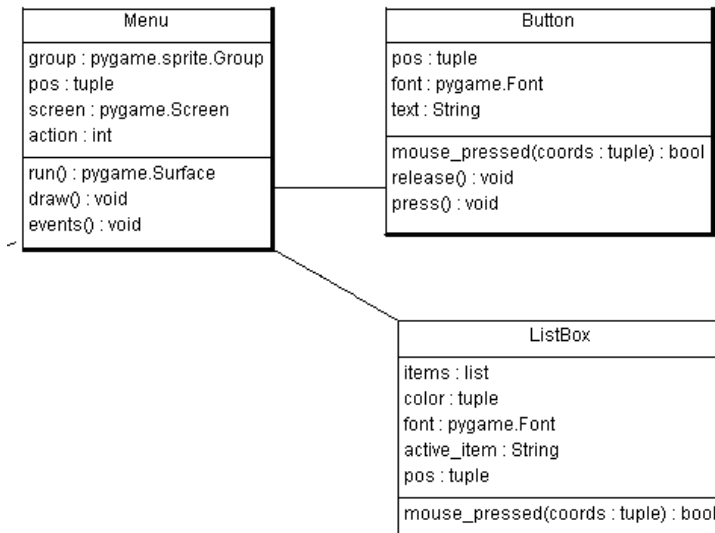
Ball-luokka on ohjattavaa palloa varten. Se sisältää paikka- ja nopeusvektorit.

LevelSequence-luokka hoitaa tasodatan luvun. Load-metodilla voidaan ladata tekstitiedosto, joka sisältää kokoelman eri tasoja. Get_level-metodi palauttaa tason string-tyyppisenä GLrender-luokan käyttöön (KUVIO 7).



KUVIO 7. Luokkakaavio 2

Menu-luokka on valikkoja varten. Se sisältää pari käyttöliittymäkomponenttia, kuten napin ja listan (KUVIO 8). Käyttöliittymäkomponentit toteutetaan `pygame.sprite`-luokkaa hyödyntäen.



KUVIO 8. Luokkakaavio 3.

Editor-luokka on tasojen muokkausta varten. Sen toteutukseen ei käytetä OpenGL-piirtoa, vaan pygamen tarjoamia metodeita 2-ulotteiseen piirtoon. Kun editori käynnistetään, käyttäjä voi alkaa muokata tyhjää tasokokoelmaa tai ladata jo olemassa olevan.

Editor-luokassa on map-muuttuja, 3-ulotteinen lista, mihin ladataan peliobjekteja esittävät merkit. Map-muuttujan ensimmäinen järjestysluku kertoo tason ja kaksi jälkimmäistä koordinaatit. Luokassa on järjestyslukumuuttuja, joka kertoo, mitä tasoa muokataan. Next- ja prev-metodeilla voidaan liikkua tasosta toiseen, jolloin myös järjestyslukumuuttujaa muokataan.

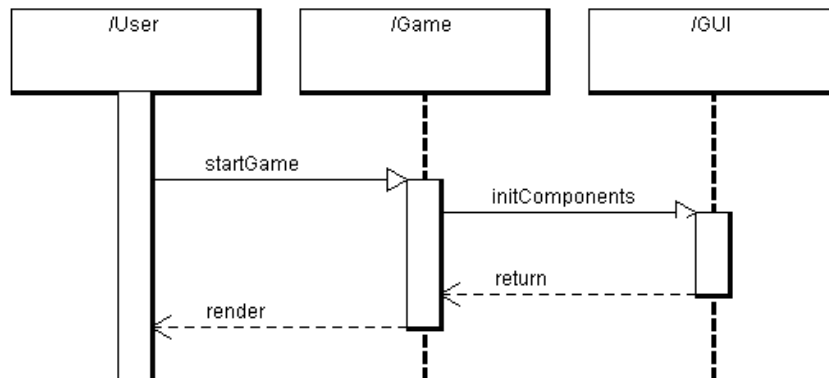
Add_object-metodilla voi lisätä peliobjekteja tasoon. Metodi ottaa argumentteina x- ja y-koordinaatit ja lisättävän peliobjektin. Kenttämuokkauksen loppuksi käyttäjä voi tallentaa muutokset, jolloin save-metodia kutsutaan.

5.4.3 Pelielementtien vuorovaikutus

Peli koostuu keskenään kommunikoivista palasista. Suunnittelussa kaikkia palasia voidaan kuvailla olioilla, käyttäjät mukaan lukien. Sekvenssikaavioilla

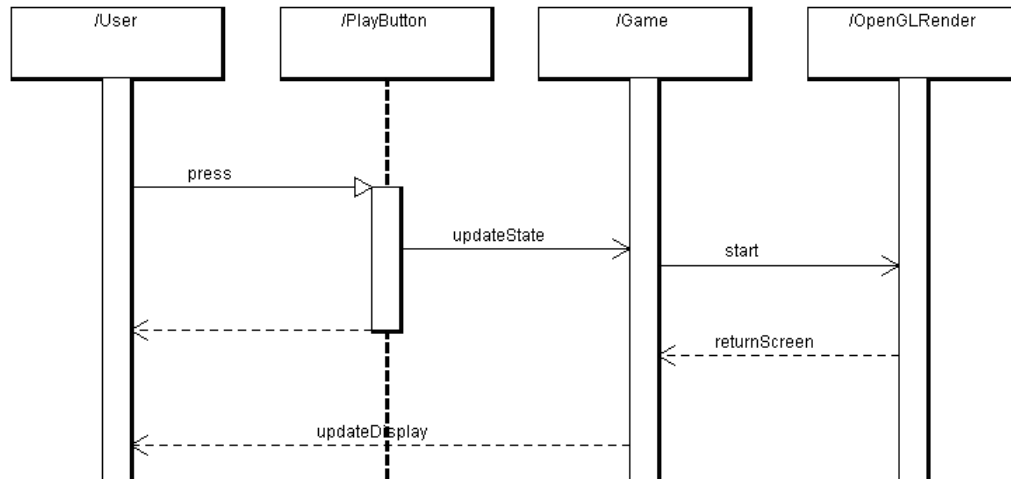
voidaan kuvailla olioiden välisiä suhteita ja vuorovaikutuksia ja sitä kautta löytää tehokkain toteutusmalli. (Ambler 2010.)

Sekvenssikaavio kuvailee pelinavaamistilannetta (KUVIO 9). Käyttäjä aloittaa pelin. Peli alustaa ja piirtää käyttöliittymän.



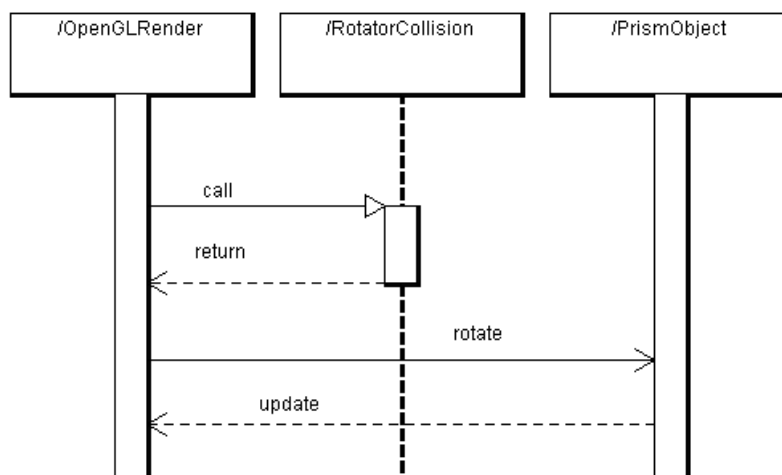
KUVIO 9. Sekvenssikaavio 1

Kaavio kuvailee tilannetta, missä käyttäjä painaa pelaa-painiketta (KUVIO 10). Jos pelaa-painiketta on painettu, mikä tapahtuu käytännössä tapahtumajonoa kuuntelemalla, pelin tila päivitetään. Kun peli on play-tilassa, OpenGLRender-olion koodia suoritetaan.



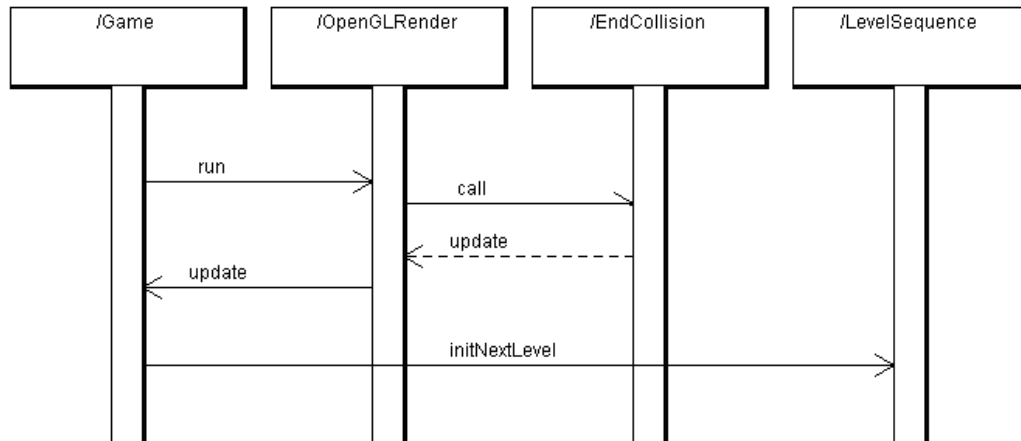
KUVIO 10. Sekvenssikaavio 2

Kaavio kuvailee tilannetta, missä törmäys on tapahtunut. Tormäysfunktio on palauttanut True-arvon, jolloin ohjelma haarautuu kohtaan, missä suunnanmuuttajat pyöräytetään (KUVIO 11). Suunnanmuuttajien asentoattribuuttia muutetaan ja OpenGL-käskyt hoitavat graafisen puolen piirto-metodissa.



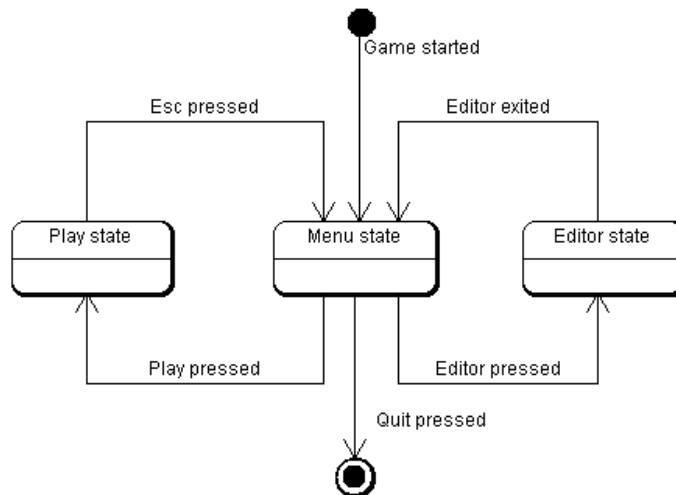
KUVIO 11. Sekvenssikaavio 3

Kun törmäysfunktio palauttaa True-arvon, pallo- ja maaliobjektien kohdalla, niin peli-olion tehtävänä on alustaa seuraava kenttä. (KUVIO 12). Peli-olio alustaa seuraavan tason.



KUVIO 12. Sekvenssikaavio 4

Pelillä on kolme tilaa, play-, menu- ja editor-tila. Tiloja voisi olla useampiakin, kuten load- tai story-tila. Vastaavasti tiloilla voisi olla alitiloja, kuten play-tilalla pause-tila. Rajaus on kuitenkin tehty kolmeen päätilaan. Tilakaaviolla voidaan tarkastella tiloja tarkemmin eli sitä, mitkä tapahtumat on sallittu muuttamaan pelin tilaa. (KUVIO 13).



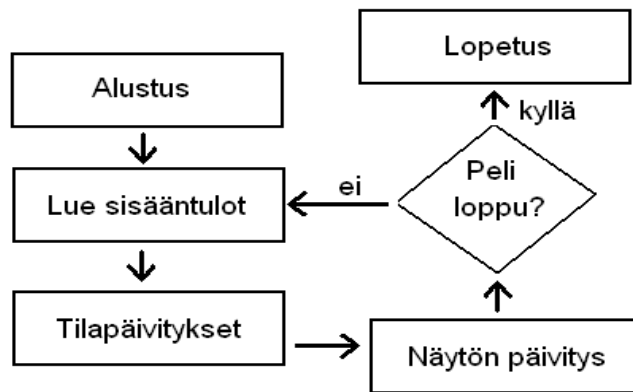
KUVIO 13. Tilakaavio

Tilakaaviota tehtäessä voidaan tehdä sellaisia havaintoja, jotka saattaisivat jäädä huomaamatta; esimerkiksi pääseekö kaikista tiloista pois sekä hiiren että näppäimistön avulla ja toimivatko globaalit, jo standardeiksi muodostuneet näppäinyhdistelmät, kuten Alt+F4 ja Alt+Tab. Käyttöliittymään jää vähemmän vajeita hyvällä suunnittelulla.

5.5 Toteutus

5.5.1 Pelisilmukka

Peliohjelmoinnin aloittamiseksi on ymmärrettävä pelisilmukan ja animaation toiminta (KUVIO 14). Pelisilmukka on loputon silmukka, jota ajetaan tarpeeksi monta kertaa sekunnissa, että saadaan luotua illuusio liikkuvasta kuvasta. Jokaisella kierroksella tehdään tarvittavia päivityksiä käyttäjän tuottamien sisääntulojen mukaan.



KUVIO 14. Pelisilmukan toimintamalli (Dawson 2004.)

Sisääntulot eli syötteet ovat keskeinen asia, sillä ne erottavat pelit animaatioista ja tekevät niistä dynaamisia. Käyttäjän tuottamat syötteet luetaan pelisilmukan alussa, jotta voidaan tehdä tarvittavat tilapäivitykset. Tilapäivityksiin sisältyy mm. törmäystarkistukset, peliobjektien sijaintien, orientaatioiden ja nopeuksien päivitykset, tekoälyn prosessointi ja äänen ulostulo. Näytön päivitys on pelisilmukan viimeisimpiä vaiheita. Pelin lopetuskäsky voi olla useassa paikassa, kuten näppäimistöpainallusten lukuvaiheessa tai jonkin tapahtuman tapahtuessa. (Dawson 2004.)

5.5.2 Alustukset

Gamemain on pelin päämoduuli, joka sisältää pääfunktion, pelin tilan ja piirtotoimenpiteet. Gamemain-moduuli sisältää Game-, GLRender-, Sphere- ja Cube-luokat. Ensimmäiseksi moduuliin tuodaan joitakin välttämättömiä kirjastoja. Ohjelman ajo alkaa main-funktiosta, jossa tehdään ensimmäiset alustukset (KOODI 15). Clock-olion tick-metodi palauttaa edellisestä kuvapäivityksestä kuluneen ajan millisekunneissa. FPS-vakio on määritelty constants.py-tiedostossa arvoksi 60, eli jos tietokone on tarpeeksi tehokas, niin td-muuttujan arvo on n. 16 ms, joka tulee olemaan myös yhdelle kuvapäivitykselle varattu aika.

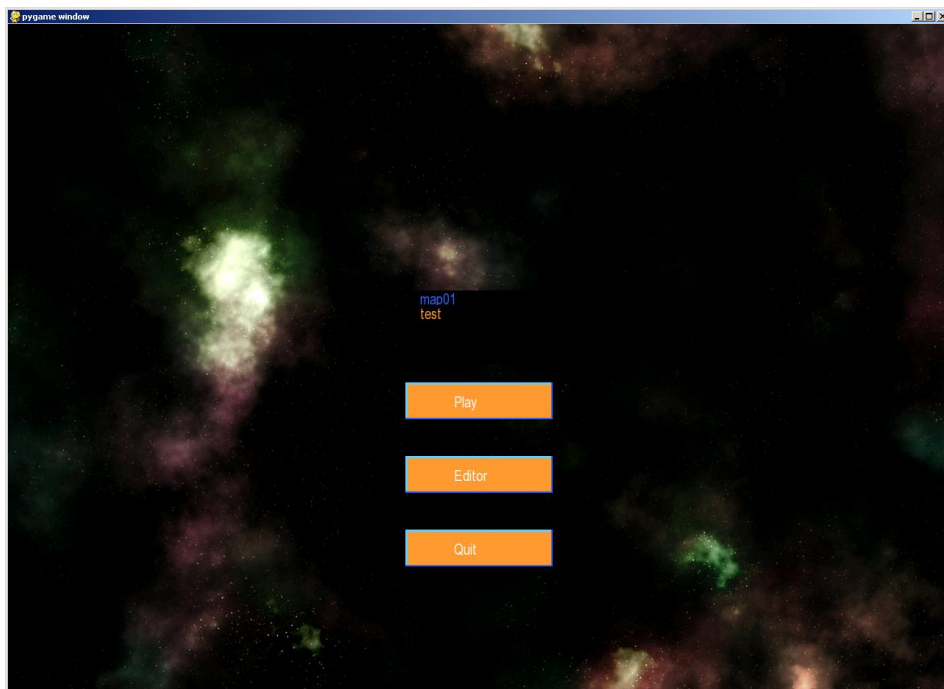
```
def main():
    pygame.init()
    clock = pygame.time.Clock()
    while(1):
        td = clock.tick(FPS)
        ...
```

KOODI 15. Pygamen alustus

Kun menu-tilaan on siirrytty, game-luokassa on tehty muiden komponenttien alustukset. Gamemenu-moduuli sisältää menu-luokan ja käyttöliittymäkomponenttiluokat. Valikko on pelin ensimmäinen näytöllä näkyvä komponentti (KUVIO 15). Menu-luokassa alustetaan käyttöliittymäkomponentit (KOODI 16).

```
class Menu:
    def __init__(self):
        self.screen = pygame.Surface(SCREEN_SIZE)
        self.group.add(Button((x-x_size) / 2, (y-y_size)/2, 20, "Play", ...))
        items = []
        for fn in os.listdir(os.getcwd()+"\\'+MAPS_FOLDER):
            if ".txt" in fn:
                items.append(fn)
        self.group.add(ListBox(... , items, ...))
```

KOODI 16. Menu-tilan alustus



KUVIO 15. Menu-tila

Play-tilaan siirryttäessä ladataan ja alustetaan käyttäjän valitsema kenttä sekä peliobjektien tekstuurit. Game-luokan td-muuttuja kuljetetaan renderöinti-luokkaan, jonka run-metodissa päivitetään peliobjektien tiloja (KOODI 17).

```
class Game:
    def __init__(self):
        pygame.display.init()
        self.render = GLRender()
        self.state = 0 #0 = menu, 1 = game 2 = editor
        ...
    def draw(self, td):
        if self.render.running:
            self.render.run(td)
class GLRender:
    ...
    def run(self, td):
        ...
        self.board.render(td)
```

KOODI 17. Play-tilaan siirtyminen

5.5.3 Syötteet ja tilapäivitykset

Tapahtumien ja syötteiden hallinta on toteutettu jokaiselle tilalle erikseen käyttämällä pygamen tarjoamaa event-moduulia. Tapahtumajono käydään läpi jokaisen kuvanpäivityksen aikana. Kaikissa pelin tiloissa kuunnellaan sekä näppäimistöllä että hiirellä tuotettuja syötteitä. Jokaisen tapahtumatyyppin kohdalla tarkistetaan, mitä painiketta on painettu ja missä kohdassa.

Valikkotilassa napattavia tapahtumia ovat MOUSEBUTTONUP, MOUSEBUTTONDOWN ja KEYDOWN. Jos hiiren vasemmanpuoleisinta painiketta on painettu ja tapahtuman sijainti on kohdassa, missä sijaitsee jokin käyttöliittymäkomponentti, tehdään vaadittavia tilapäivityksiä (KOODI 18). Tällöin käyttäjä saa jonkinlaisen viestin, että jotain tapahtuu, kuten pelin siirtyminen toiseen tilaan. Näppäimistöltä kuunneltavat painikkeet ovat K_ESCAPE ja K_RETURN, joilla voidaan poistua pelistä tai siirtyä play-tilaan.

```
def events(self):
    for event in pygame.event.get():
        if(event.type == MOUSEBUTTONUP):
            if(event.button == 1):
                for sprite in self.group:
                    if isinstance(sprite, Button):
                        if sprite.mouse_pressed(event.pos):
                            if sprite.text == "Play":
                                self.action = 1
```

KOODI 18. Play-nappulan toiminta

Play-tilassa (KUVIO 16) tapahtumia käydään läpi samalla periaatteella. Käytäviä tapahtumia ovat MOUSEBUTTONDOWN, MOUSEMOTION ja KEYDOWN. Hiiritapahtumilla voidaan asettaa kameran sijainti. Jos hiiren vasemmanpuoleisinta painiketta on painettu, kursori menee piiloon ja hiiri siirtyy kameransiirtotilaan, missä liikkeellä voi siirtää kameraa x-, ja z-akselilla ja rullalla y-akselilla. Kun painiketta painetaan uudestaan, kursori tulee näkyviin ja kameransiirtotilasta poistutaan. Prosessoitavat näppäimistötapaukset ovat K_ESCAPE ja nuolinäppäinten painallukset. Nuolinäppäinten painalluksilla käsketään pallo liikkeelle (KOODI 19). Pallon liikutus on mahdollista vain silloin, kun pallo ei ole liikkeessä.

```

if self.board.ball.moving == False:
    if pressed[K_LEFT]:
        self.board.ball.v.x = -BALLSPEED
        self.board.ball.moving = True

```

KOODI 19. Pallon liikuttaminen

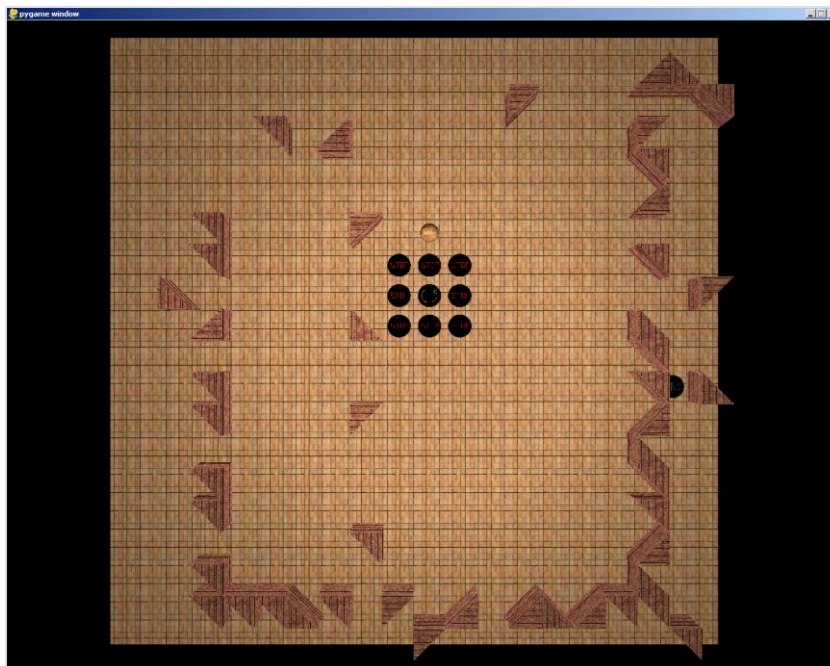
Pallon liikkeestä tehdään td-muuttujan avulla aikaan perustuvaa, jolloin kuvanpäivitysnopeudesta riippumatta liike on yhtä nopeaa. Pallon paikkavektoriin summataan pallon nopeusvektori kerrottuna td:llä (KOODI 20).

```

class Ball:
    ...
    def update(self, td):
        self.pos += self.v * td

```

KOODI 20. Pallon paikan laskeminen



KUVIO 16. Play-tila

Cube-oliot ovat staattisia kuutioita, joilla on tyyppiattribuutti, joka määrittelee, onko ruutu tyhjä vai onko siinä pysäyttävä tai pyöräyttävä. Kun pallon törmäys pyöräyttäjään on huomattu, suunnanmuuttajien asento-attribuuttia muutetaan.

5.5.4 Piirto

Valikkotilan piirto on toteutettu pygamen tarjoamilla työkaluilla. Jokaisen kuvanpäivityksen aikana piirretään ensimmäisenä taustakuva ja sen jälkeen käyttöliittymäkomponentit.

Play-tilaan siirryttäessä näyttö alustetaan tilaan, joka tukee OpenGL:ä ja grafiikkakiihdytystä. Jokaisella pelisilmukan kierroksella piirrettäviä objekteja ovat sekä pallo että cube- ja prism-oliot. Pallon data on generoitu OpenGL.GLU-kirjaston gluSphere-käskyllä. Cube- ja prism-olioitten data on määritelty manuaalisesti, piste kerrallaan (KOODI 21).

```
vertices = [(0.0, 0.0, 2.0),  
            (2.0, 0.0, 2.0),  
            (2.0, 2.0, 2.0),  
            (0.0, 2.0, 2.0),  
            (0.0, 0.0, 0.0),  
            (2.0, 0.0, 0.0),  
            (2.0, 2.0, 0.0),  
            (0.0, 2.0, 0.0)]
```

KOODI 21. Cube-olion pisteet

Cube- ja prism-olioiden piirron tehostamiseksi käytetään OpenGL:n glNewList-käskyä, jonka avulla voidaan nopeuttaa staattisten, paikallaan pysyvien objektien piirtoa. Kun lista luodaan, sen jälkeen kaikki OpenGL käskyt vaikuttavat listaan, kunnes glEndList-käskyä kutsutaan. Sen jälkeen listaa voidaan kutsua glCallList-käskyllä. Listan kutsuminen piirtää saman lopputuloksen kuin käskyt, joita listan luontiin syötettiin. Kaikki cube- ja prism-oliot on lisätty listoihin play-tilaan siirryttäessä. (Ahn 2005.)

5.5.5 Kenttäeditori

Gameeditor-moduuli sisältää kaiken, mitä editor-tila vaatii. Moduuli sisältää Editor-luokan, jonka metodit hoitavat suurimman osan kaikesta moduulin toiminnasta. Add_object-metodi lisää listaan sprite-tyyppisiä olioita, jotka

edustavat peliobjekteja (KOODI 22). Metodia kutsutaan, kun hiiren painiketta on painettu editointi-alueella ja jokin objekti on valittu lisättäväksi.

```
def add_object(self, x, y, sc):
    for sprite in self.sprites.sprites():
        if sprite.pos == (x, y): self.sprites.remove(sprite)
        self.map[self.i][x][y] = '0'
    if sc == 'r':
        self.sprites.add(Rotator((x, y), 'rotator.png', BLACK))
    ...
    self.map[self.i][x][y] = sc
```

KOODI 22. Editori: objektin lisäys

Next- ja prev-metodeilla (KOODI 23) voidaan siirtyä tasosta toiseen tasokokoelman sisällä. Tason data on tallennettu valmiiksi map-listaan, jolloin ei tarvitse tehdä muuta kuin tyhjentää sprite-lista ja lisätä uudet tilalle.

```
def next(self):
    self.clear()
    self.i += 1
    print self.i, len(self.map), self.map
    if self.i == len(self.map):
        self.map.append(['0' for row in range(MAPSIZE)] for col in range(MAPSIZE))
    self.populate_map()

def prev(self):
    self.clear()
    self.i -= 1
    if self.i == -1: self.i = len(self.map)-1
    self.populate_map()
```

KOODI 23. Editori: kentän vaihto

Load- ja save-metodeilla (KOODI 24) voidaan ladata ja tallentaa tasokokoelmia. Metodeita kutsutaan, jos tiettyä käyttöliittymäkomponenttia on painettu (KUVIO 17).

```
def save(self, fn):
    try:
        f = open(MAPS_FOLDER+'\\'+fn+'.txt', 'w')
        for i in range(len(self.map)):
            for row in range(MAPSIZE):
                for col in range(MAPSIZE):
                    f.write(self.map[i][col][row])
```

```

        f.write(chr(10))
        f.write(chr(10))
        f.close()
    except IOError, error: print error

```

KOODI 24. Editori: kenttäkokoelman tallennus

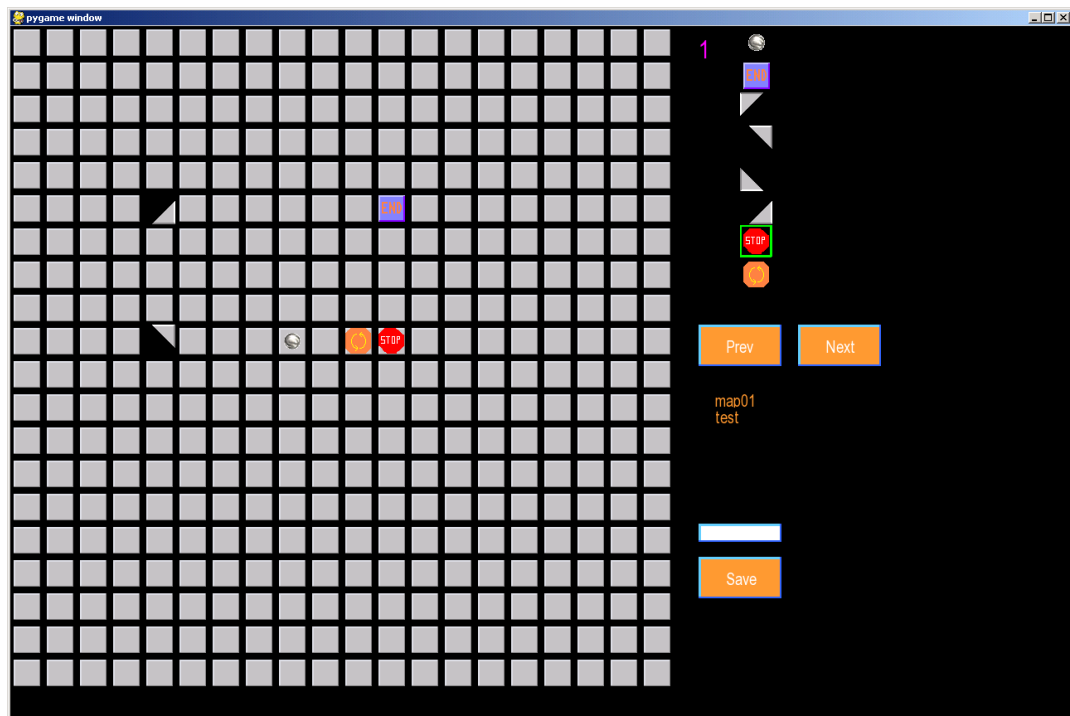
Editor-luokan events-metodissa (KOODI 25) käsitellään hiiren ja näppäimistön painallukset. Hiiren painalluksen koordinaatit otetaan talteen ja tarkistetaan, ovatko ne editointi-alueella.

```

def events(self):
    for event in pygame.event.get():
        if event.type == MOUSEBUTTONDOWN:
            if event.button == 1:
                x, y = event.pos
                x /= TILESIZE
                y /= TILESIZE
                if x < 20 and y < 20:
                    if self.selected_object != None:
                        self.add_object(x, y, self.selected_object.sc)

```

KOODI 25. Editori: tapahtumien käsittely



KUVIO 17. Editor-tila

5.6 Tehokkuustestaus

Testaus suoritetaan koristelijafunktiolla (KOODI 26), joka voidaan ujuttaa minkä tahansa metodin tai funktion eteen. Koristelijafunktio tulostaa metodin tai funktion käyttämän ajan. Testattavia metodeita ovat eri tilojen piirtoajat ja niiden alimetodit.

```
def benchmark(func):
    def wrapper(*args, **kwargs):
        t = time.clock()
        res = func(*args, **kwargs)
        tt = time.clock()-t
        print func.__name__,tt
        return res
    return wrapper
```

KOODI 26. Testauksen apufunktio.

Play-tilan kaikkeen prosessointiin kului 4,2 ms (TAULUKKO 3.), joka on yllättävän lyhyt aika ottaen huomioon prosessoinnin määrän. Se viestii siitä, että näytönohjaimen resursseja käytettäessä pelilogiikkaan ja algoritmeihin jäi huomattavasti enemmän aikaa. Yhdelle kuvanpäivitykselle on annettu prosessointiaikaa n.16 ms, joten siitä kulutettiin vain reilut yksi neljäsosaa. Jos kuvanpäivityksen aikana pyöräytettiin 50 suunnanmuuttajaa, aikaa kului yhden kuvanpäivityksen ajan 19 ms (TAULUKKO 3), jonka jälkeen kulutus palautui normaalille tasolle. On siis liian hidasta pyöräyttää monta objektia kerralla. Parempi toteutusmalli olisi tallentaa suunnanmuuttajan neljä eri asentoa välimuistiin. Pallon piirtoon kului 0,5 ms, joka on suhteellisen pitkä aika. Pallo on ainoa objekti, mikä ei ole osana staattista listaa, koska se on suurimman osan ajasta liikkeessä. Pallossa on myös suuri määrä pisteitä, mikä osittain selittää piirtoon kulutettua aikaa.

Valikko- ja editointitilojen prosessointiin kului n.1 ms, mikä ei yllätä, koska liikkuvia objekteja ei ole. Toisaalta jos editor-tilassa kentän täytti objekteilla, prosessointiaika nousi selvästi, jopa liian korkeaksi. Kun editor-tilassa siirtyi kentästä toiseen, aikaa kului 60 objektin kenttään 15 ms ja 120 objektin kenttään n. 30 ms (TAULUKKO 3). Tekstuurien alustuksiin kului 90 ms, mitkä

suoritetaan play-tilaan siirryttäessä. 90 ms on sen verran pitkä aika, että alustuksen voisi siirtää pelin käynnistykseen.

TAULUKKO 3. Prosessointiaikojen testaus

Testattava	Kulutettu aika(ms)
Play-tilan silmukka	4,2
Suunnanmuuttajien pyöräytys	19
Pallon piirto	0,5
Menu-tilan silmukka	1
Editor-tilan silmukka	1,2–10
Editor-tila: kentän vaihto	0,5–100
Tekstuurien alustus	90

6 YHTEENVETO

Jos ohjelmointikielestä voi koskaan käyttää sanaa mukava, Python on ainakin hyvin lähellä sitä. Lähes kaikissa tilanteissa ohjelman kirjoittaminen Pythonilla tuntuu helpolta, järkevältä ja elegantilta. Suurimpia ongelmia Pythonin kanssa aiheuttaa .exe-tiedostojen luominen ja niiden alustariippumattomuus. Jos joku todella haluaa tehdä Pythonilla pelin, se vaatii alemman tason koodia, kuten C-koodia.

Ongelmat .exe-tiedostojen luomisessa johtivat muutamiin kysymyksiin. Mihin Python oikein on loppujen lopuksi tarkoitettu? Mikä on sen asema muiden ohjelmointikielten joukossa? Jos C/CPP-kieltä vaaditaan ”kunnollisen” Python-ohjelman luomiseen, miksi ei kannattaisi pysyttäytyä kokonaan C/CPP-kielessä? Pythonia voi käyttää myös verkkosivujen tekoon, esimerkiksi djangoa käyttämällä. Toisaalta sillä saralla PHP on ainakin toistaiseksi käytetyin kieli.

Pygame on aloittelijalle sopiva ja hyvin selkeä paketti oppia peliohjelmoinnin perusniksit, ja sopii se kokeneemmallekin. PyOpenGL toimii pygamen kanssa mutkattomasti.

Pelin luominen oli mielenkiintoinen kokemus, ja siinä varmasti karttui uusia taitoja, tosin parannettavaakin jäi runsaasti. Parannettavaa jäi tapahtumien käsittelyn suunnittelussa, OpenGL-käskyissä ja pelin viimeistelyssä.

Python sopii erittäin hyvin opiskeluun tai pelien prototyypittämiseen. Mutta jos ohjelmoija haluaa jakaa työnsä muille tai tehdä jopa kaupallisen tuotteen, kannattaa ohjelma ainakin osittain tehdä C-kielellä.

LÄHTEET

A Byte of Python, v1.92 (for Python 3.0). 2009. Www-dokumentti. Saatavissa: <http://www.ibiblio.org/swaroopch/byteofpython/read/>

Ahn S. 2005. Www-dokumentti. Saatavissa: http://www.songho.ca/opengl/gl_displaylist.html

Ambler S. 2010. Www-dokumentti. Saatavissa: <http://www.agilemodeling.com/artifacts/sequenceDiagram.htm>

Dawson M. 2004. Beginning C++ Game Programming. Premier Press.

Drillian's House of Game Development. 2009. Www-dokumentti. Saatavissa: <http://drilian.com/2009/02/25/lightning-bolts/>

Hall T. Python 3 for Absolute Beginners. 2009. Apress.

McGugan W. 2007. Beginning Game Development with Python and Pygame - From Novice to Professional. Berkeley: Apress.

Payne J. Beginning Python - Using Python 2.6 and Python 3.1. 2010. Indianapolis: Wiley Publishing, Inc.

Pygame Documentation. 2011 a. Www-dokumentti. Saatavissa: <http://www.pygame.org/docs/ref/display.html>

Pygame Documentation. 2011 b. Www-dokumentti. Saatavissa: <http://www.pygame.org/docs/ref/event.html>

Pygame Documentation. 2011 c. Www-dokumentti. Saatavissa: <http://www.pygame.org/docs/ref/mixer.html>

Pygame Documentation. 2011 d. Www-dokumentti. Saatavissa: <http://www.pygame.org/docs/ref/time.html>

Python Community. 2011. Www-dokumentti. Saatavissa: <http://www.Python.org/about/>

Python Software Foundation. 2011 a. Www-dokumentti. Saatavissa: <http://docs.Python.org/library/random.html>

Python Software Foundation. 2011 b. Www-dokumentti. Saatavissa: <http://docs.Python.org/library/os.html>

Python Software Foundation. 2011 c. Www-dokumentti. Saatavissa:
<http://docs.Python.org/library/time.html>

Rost R. 2006. OpenGL Shading Language. Addison Wesley Professional.

Summerfield M. 2010. Programming in Python 3 - A Complete Introduction to the Python Language. Second Edition. Boston: Addison-Wesley.

LIITTEET

```
"""gamemain.py"""
```

```
import sys, pygame, time, gamemenu, gameeditor
from gamemap import LevelSequence
from pygame.locals import *
from constants import *
from Image import *
from math import *
#from numpy import *
from OpenGL.GL import *
from OpenGL.GLU import *
from gameobjects.matrix44 import *
from gameobjects.vector3 import *
#from ctypes import util
```

```
try:
    from OpenGL.platform import win32
except AttributeError:
    pass
```

```
#-----
```

```
def collision_orthogonal(sphere, prism):
    if sphere.v.x < 0.0 and (prism.align == 2 or prism.align == 4):
        return True
    elif sphere.v.x > 0.0 and (prism.align == 1 or prism.align == 3):
        return True
    elif sphere.v.z < 0.0 and (prism.align == 3 or prism.align == 4):
        return True
    elif sphere.v.z > 0.0 and (prism.align == 1 or prism.align == 2):
        return True
    else: return False
```

```
#-----
```

```
def collision_detect(sphere, object_list):
    x1, y1, z1 = sphere.pos
    r = sphere.r
    for obj in object_list:
        x2, y2, z2 = obj.pos
        #if prism, do special checks
        if isinstance(obj, Prism):
            #if intersecting a tile that contains a prism
            if x1 > x2-r and x1 < x2 + TSIZE+r and z1 > z2-r and z1 < z2 + TSIZE+r:
```

```
                #make a small box to smoothen visuals of the collision
                boxsize = TSIZE * 0.3
```

```
                #is the prism collision orthogonal
                if collision_orthogonal(sphere, obj):
                    return obj
```

```
                #else wait for the reaction until the middle of the tile
                elif x1 > x2 + boxsize and x1 < x2 + TSIZE - boxsize and z1 > z2 + boxsize and z1 <
z2 + TSIZE - boxsize:
                    return obj
                #else do a basic check
                elif x1 > x2 and x1 < x2 + TSIZE and z1 > z2 and z1 < z2 + TSIZE:
                    return obj
```

```

#-----
class Prism:
    def __init__(self, position, texture, align = 1):
        x, z = position
        self.y = 0.0
        self.x = x
        self.z = z
        self.mapcoords = (x/TSIZE, z/TSIZE)
        self.pos = Vector3(self.x, self.y, self.z)
        self.texture = texture
        self.align = align
        self.size = TSIZE
        self.vertices = [ (0.0, self.size*2, 0.0),
                          (self.size, self.size*2, 0.0),
                          (0.0, self.size*2, self.size),
                          (self.size, self.size, 0.0),
                          (0.0, self.size, 0.0),
                          (0.0, self.size, self.size) ]

    rect_faces = 3
    tri_faces = 1

    #normals of the faces
    normals = [ (0.0, 0.0, -1.0), # back
                (-1.0, 0.0, 0.0), # left
                (1.0, 0.0, 1.0), # right-front
                (0.0, 1.0, 0.0) ] # top

    vertex_indices = [ (0, 1, 3, 4), # back
                       (0, 2, 5, 4), # left
                       (2, 1, 3, 5), # right-front
                       (0, 1, 2) ] # top

#-----
    def rotate(self):
        print self.pos
        if self.align == 2:
            angle = 90
        elif self.align == 3:
            angle = 180
        elif self.align == 4:
            angle = 270

#-----
    def render(self):
        # Adjust all the vertices so that the cube is at self.position
        vertices = []
        for v in self.vertices:
            vertices.append(tuple(Vector3(v)))

        glPushMatrix()
        glNormal3dv(self.normals[3])
        if self.align == 2:
            glTranslatef(self.pos.x+self.size, self.pos.y, self.pos.z)
            glRotatef(270, 0.0, 1.0, 0.0)
        elif self.align == 3:
            glTranslatef(self.pos.x, self.pos.y, self.pos.z+self.size)
            glRotatef(90, 0.0, 1.0, 0.0)
        elif self.align == 4:
            glTranslatef(self.pos.x+self.size, self.pos.y, self.pos.z+self.size)

```

```

        glRotatef(180, 0.0, 1.0, 0.0)
    else:
        glTranslatef(self.pos.x, self.pos.y, self.pos.z)
        glBindTexture(GL_TEXTURE_2D, self.texture)

        glBegin(GL_QUADS)
        for face_no in xrange(self.rect_faces):
            #glNormal3dv( self.normals[face_no] )
            v1, v2, v3, v4 = self.vertex_indices[face_no]
            glTexCoord2f(0.0, 0.0)
            glVertex(vertices[v1])
            glTexCoord2f(1.0, 0.0)
            glVertex(vertices[v2])
            glTexCoord2f(1.0, 1.0)
            glVertex(vertices[v3])
            glTexCoord2f(0.0, 1.0)
            glVertex(vertices[v4])
        glEnd()

        #draw top side
        glBegin(GL_TRIANGLES)
        glTexCoord2f(0.0, 0.0)
        glVertex(self.vertices[0])
        glTexCoord2f(1.0, 0.0)
        glVertex(self.vertices[1])
        glTexCoord2f(0.0, 1.0)
        glVertex(self.vertices[2])
        glEnd()
        glPopMatrix()

```

```

#-----
class Cube(object):
    def __init__(self, position, texture, cubetype):
        x, z = position
        self.pos = x, 0.0, z
        self.mapcoords = (x/TSIZE, z/TSIZE)
        self.texture = texture
        self.type = cubetype
    # Cube information
    num_faces = 6
    vertices = [ (0.0, 0.0, TSIZE),
                  (TSIZE, 0.0, TSIZE),
                  (TSIZE, TSIZE, TSIZE),
                  (0.0, TSIZE, TSIZE),
                  (0.0, 0.0, 0.0),
                  (TSIZE, 0.0, 0.0),
                  (TSIZE, TSIZE, 0.0),
                  (0.0, TSIZE, 0.0) ]
    normals = [ (0.0, 0.0, +1.0), # front
                (0.0, 0.0, -1.0), # back
                (+1.0, 0.0, 0.0), # right
                (-1.0, 0.0, 0.0), # left
                (0.0, +1.0, 0.0), # top
                (0.0, -1.0, 0.0) ] # bottom
    vertex_indices = [ (0, 1, 2, 3), # front
                       (4, 5, 6, 7), # back
                       (1, 5, 6, 2), # right
                       (0, 4, 7, 3), # left
                       (3, 2, 6, 7), # top

```

(0, 1, 5, 4)] # bottom

```
#-----
def render(self):
    # Adjust all the vertices so that the cube is at self.position
    vertices = []
    for v in self.vertices:
        vertices.append( tuple(Vector3(v)+ self.pos) )
    # Draw all 6 faces of the cube
    glBindTexture(GL_TEXTURE_2D, self.texture)

    glBegin(GL_QUADS)
    #glNormal3dv(self.normals[4])
    for face_no in xrange(self.num_faces):
        glNormal3dv( self.normals[face_no] )
        v1, v2, v3, v4 = self.vertex_indices[face_no]
        if face_no == 4:
            glTexCoord2f(0.0, 0.0)
            glVertex( vertices[v1] )
            glTexCoord2f(1.0, 0.0)
            glVertex( vertices[v2] )
            glTexCoord2f(1.0, 1.0)
            glVertex( vertices[v3] )
            glTexCoord2f(0.0, 1.0)
            glVertex( vertices[v4] )
        else:
            glVertex( vertices[v1] )
            glVertex( vertices[v2] )
            glVertex( vertices[v3] )
            glVertex( vertices[v4] )
    glEnd()
```

```
#-----
class GameBoard(object):
    def __init__(self):
        w, h = (MSIZE, MSIZE)
        self.display_list = None
        self.ends = []
        self.rotators = []
        self.stoppers = []
        self.cubes = []

    #-----
    def reinit(self, ball, cubes, triangles):
        self.ball = ball
        self.ends = []
        self.rotators = []
        self.stoppers = []
        self.cubes = []
        for cube in cubes:
            if cube.type == 1:
                self.ends.append(cube)
            elif cube.type == 2:
                self.stoppers.append(cube)
            elif cube.type == 3:
                self.rotators.append(cube)
            else:
                self.cubes.append(cube)
        self.triangles = triangles
        self.static_display_list = None
        self.prism_display_list = None
```

```

#-----
def render(self, td):
    if self.static_display_list is None:
        self.static_display_list = glGenLists(1)
        glNewList(self.static_display_list, GL_COMPILE)
        for cube in self.cubes:
            cube.render()
        for stopper in self.stoppers:
            stopper.render()
        for rotator in self.rotators:
            rotator.render()
        for end in self.ends:
            end.render()
        glEndList()
    else:
        glCallList(self.static_display_list)

    if self.prism_display_list is None:
        self.prism_display_list = glGenLists(1)
        glNewList(self.prism_display_list, GL_COMPILE)
        for triangle in self.triangles:
            triangle.render()
        glEndList()
    else:
        glCallList(self.prism_display_list)
    self.ball.update(td)

#-----
class Ball:
    def __init__(self, pos, texture):
        self.texture = texture
        self.r = 0.6
        x, z = pos
        self.pos = Vector3(x, TSIZE, z)
        self.collpos = (-1, -1)
        self.startpos = self.pos.copy()
        self.v = Vector3()
        self.x_angle = 0.1
        self.z_angle = 0.1
        self.moving = False
        self.centrifing = False
        self.Q = gluNewQuadric()

        glTranslatef(self.pos.x, self.pos.y, self.pos.z)
        self.render()

#-----
    def __str__(self):
        return "Position x: {0}"

#-----
    def centrifry(self, frames = 1):
        x, y, z = self.pos
        x = int(x/TSIZE)
        z = int(z/TSIZE)
        if frames == 1:
            self.pos.x = x*TSIZE + TSIZE/2
            self.pos.z = z*TSIZE + TSIZE/2
        else:
            self.cdx = (self.pos.x - (x*TSIZE + TSIZE/2)) / frames
            self.cdz = (self.pos.z - (z*TSIZE + TSIZE/2)) / frames

```



```

        self.frames = frames
        self.centriying = True
#-----
def center_distance(self):
    x, y, z = self.pos
    x = int(x/TSIZE)
    z = int(z/TSIZE)
    return (self.pos.x - (x*TSIZE + TSIZE/2), self.pos.z - (z*TSIZE + TSIZE/2))

#-----
def stop(self):
    self.v.x = 0.0
    self.v.z = 0.0
    self.moving = False

#-----
def reset_angles(self):
    self.z_angle = 0.1
    self.x_angle = 0.1

#-----
def update(self, td):
    if self.centriying:
        self.stop()
        if self.frames > 1:
            self.pos.x -= self.cdx
            self.pos.z -= self.cdz
            self.frames -= 1
        else:
            self.centriying = False
    else:
        self.pos += self.v * td
    bounds = MSIZE*TSIZE+self.r
    if self.pos.x < -self.r or self.pos.x > bounds or self.pos.z < -self.r or self.pos.z > bounds:
        self.stop()
        self.pos = self.startpos.copy()
        self.collpos = (-1, -1)
        self.centrify()
    self.render()

#-----
def render(self):
    glTranslatef(self.pos.x, self.pos.y+self.r, self.pos.z)
    self.x_angle += self.v.x/(self.r*2*3.14 / 360)
    self.z_angle += self.v.z/(self.r*2*3.14 / 360)

    if self.centriying:
        self.x_angle += self.cdx/(self.r*2*3.14 / 360)
        self.z_angle += self.cdz/(self.r*2*3.14 / 360)
    glRotatef(self.x_angle*(1./FPS)*-1.0, 0.0, 0.0, self.z_angle%360)
    glRotatef(self.z_angle*(1./FPS), self.x_angle%360, 0.0, 0.0)
    gluQuadricDrawStyle(self.Q, GLU_FILL)
    gluQuadricTexture(self.Q, GL_TRUE)
    gluQuadricNormals(self.Q, GLU_FLAT)
    glBindTexture(GL_TEXTURE_2D, self.texture)
    gluSphere(self.Q, self.r, 64, 32)

#-----
class GLRender:
    def __init__(self):

```

```

        self.running = False
#-----
def start(self, map_):
    if ISFULLSCREEN:
        self.screen = pygame.display.set_mode(SCREEN_SIZE,
HWSURFACE|FULLSCREEN|OPENGL|OPENGLBLIT|DOUBLEBUF)
    else:
        self.screen = pygame.display.set_mode(SCREEN_SIZE,
HWSURFACE|OPENGL|OPENGLBLIT|DOUBLEBUF)
        self.resize(*SCREEN_SIZE)
        self.load_textures()
        glEnable(GL_DEPTH_TEST) #enable depth testing and depth buffer
        glShadeModel(GL_FLAT) #set flat shading
        glClearColor(0.0, 0.0, 0.0, 0.0) #clear color buffer? RGBA
        glEnable(GL_COLOR_MATERIAL) #have one or more material parameters track the
current color
        glEnable(GL_LIGHTING) #use the current lighting parameters to compute vertex color or
index
        glEnable(GL_LIGHT0) #include light i in lighting equation
        glEnable(GL_TEXTURE_2D)
        glEnable(GL_NORMALIZE)

    self.map = map_

    self.board = GameBoard()
    self.init_level()
    self.cam_active = False
    self.cam_x = 20
    self.cam_y = 40
    self.cam_z = 20
    self.camera_matrix = Matrix44()
    self.camera_matrix.translate = (10.0, .6, 10.0)
    #Initial camera position
    self.camera_matrix.x_axis = [1, 0, 0, 0]
    self.camera_matrix.y_axis = [0, 0, -1, 0]
    self.camera_matrix.z_axis = [0, 1, 0, 0]
    self.camera_matrix.translate = [20, 40, 20, 1]
    self.running = True

#sounds
self.mixer = pygame.mixer
self.mixer.init(44100,16,1,2048)
self.sounds = []
self.sounds.append(self.mixer.Sound('ballroll.ogg'))
self.sounds.append(self.mixer.Sound('boom.ogg'))
self.sounds.append(self.mixer.Sound('rotate.ogg'))
self.roll_channel = pygame.mixer.Channel(1)
self.collision_channel = pygame.mixer.Channel(2)
#-----
def init_level(self):
    """level string interpreter"""
    i = 0
    cubes = list()
    triangles = list()
    for char in self.map.get_level():
        if char != 'q' and char != 'e' and char != 'r':
            position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
            cube = Cube(position, self.textures[0], 0)
            cubes.append(cube)

```

```

    if char == '1' or char == '2' or char == '3' or char == '4':
        align = int(char)
        position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
        triangles.append(Prism(position, self.textures[5], align))

    if char == 'q':
        position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
        cube = Cube(position, self.textures[2], 2)
        cubes.append(cube)

    elif char == 's':
        position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
        ball = Ball(position, self.textures[1])
        ball.centrfify()

    elif char == 'e':
        position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
        cube = Cube(position, self.textures[3], 1)
        cubes.append(cube)

    elif char == 'r':
        position = (i%MSIZE*TSIZE, i/MSIZE*TSIZE)
        cube = Cube(position, self.textures[4], 3)
        cubes.append(cube)

    i+=1
    self.board.reinit(ball, cubes, triangles)
#-----
def next_level(self):
    if not self.map.is_last():
        self.map.next()
        self.init_level()
#-----
def load_textures(self):
    #global texture
    image = open("text2.jpg")

    ix = image.size[0]
    iy = image.size[1]
    image = image.tostring("raw", "RGBX", 0, -1)

    image2 = open("floor.jpg")

    ix2 = image2.size[0]
    iy2 = image2.size[1]
    image2 = image2.tostring("raw", "RGBX", 0, -1)

    image3 = open("stop.jpg")

    ix3 = image3.size[0]
    iy3 = image3.size[1]
    image3 = image3.tostring("raw", "RGBX", 0, -1)

    image4 = open("end.jpg")

    ix4 = image4.size[0]
    iy4 = image4.size[1]
    image4 = image4.tostring("raw", "RGBX", 0, -1)

```

```

image5 = open("rotator.jpg")

ix5 = image5.size[0]
iy5 = image5.size[1]
image5 = image5.tostring("raw", "RGBX", 0, -1)

image6 = open("prism.jpg")

ix6 = image6.size[0]
iy6 = image6.size[1]
image6 = image6.tostring("raw", "RGBX", 0, -1)

#cube
self.textures = glGenTextures(6)
glBindTexture(GL_TEXTURE_2D, self.textures[0]) # 2d texture (x and y siz
#glPixelStorei(GL_UNPACK_ALIGNMENT,1)
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix, iy, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
#sphere
glBindTexture(GL_TEXTURE_2D, self.textures[1])
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix2, iy2, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image2)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL)
#stopper
glBindTexture(GL_TEXTURE_2D, self.textures[2]) # 2d texture (x and y siz
#glPixelStorei(GL_UNPACK_ALIGNMENT,1)
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix3, iy3, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image3)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
#end
glBindTexture(GL_TEXTURE_2D, self.textures[3]) # 2d texture (x and y siz
#glPixelStorei(GL_UNPACK_ALIGNMENT,1)
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix4, iy4, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image4)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
#rotator
glBindTexture(GL_TEXTURE_2D, self.textures[4]) # 2d texture (x and y siz
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix5, iy5, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image5)

```

```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
#glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
#triangle
glBindTexture(GL_TEXTURE_2D, self.textures[5]) # 2d texture (x and y siz
glTexImage2D(GL_TEXTURE_2D, 0, 3, ix6, iy6, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image6)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)

#-----
def resize(self, width, height):
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60.0, float(width)/height, .1, 1000.)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

#-----
def run(self, td):

    for event in pygame.event.get():
        if event.type == QUIT:
            return
        if event.type == MOUSEMOTION:
            if self.cam_active:
                x, z = event.rel
                self.cam_x += x/10.
                self.cam_z += z/10.
        if event.type == MOUSEBUTTONDOWN:
            if event.button == 1:
                if self.cam_active:
                    pygame.mouse.set_visible(True)
                    self.cam_active = False
                else:
                    pygame.mouse.set_visible(False)
                    self.cam_active = True
            if self.cam_active:
                if event.button == 4:
                    self.cam_y -= 2.0
                if event.button == 5:
                    self.cam_y += 2.0
        if event.type == KEYUP:
            if event.key == K_ESCAPE:
                pygame.mouse.set_visible(True)
                self.running = False
            elif event.key == K_p:
                self.board.ball.stop()
                #self.next_level()
                #self.ball.r += 1
                #if self.ball.r > 10:

```

```

        #self.ball.r = 1

# Clear the screen, and z-buffer
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
td = td / 1000.
pressed = pygame.key.get_pressed()

if self.board.ball.moving == False:
    self.roll_channel.stop()
    if pressed[K_LEFT]:
        self.board.ball.v.x = -BALLSPEED
        self.board.ball.moving = True
        self.roll_channel.play(self.sounds[0],-1)
    elif pressed[K_DOWN]:
        self.board.ball.v.z = BALLSPEED
        self.board.ball.moving = True
        self.roll_channel.play(self.sounds[0],-1)
    elif pressed[K_RIGHT]:
        self.board.ball.v.x = BALLSPEED
        self.board.ball.moving = True
        self.roll_channel.play(self.sounds[0],-1)
    elif pressed[K_UP]:
        self.board.ball.v.z = -BALLSPEED
        self.board.ball.moving = True
        self.roll_channel.play(self.sounds[0],-1)

#set camera projection
#self.camera_matrix.x_axis = [1, 0, 0, 0]
#self.camera_matrix.y_axis = [0, 0, -1, 0]
#self.camera_matrix.z_axis = [0, 1, 0, 0]
self.camera_matrix.x_axis = [1, 0, 0, 0]
self.camera_matrix.y_axis = [0, 0, -1, 0]
self.camera_matrix.z_axis = [0, 1, 0, 0]
#set camera position
self.camera_matrix.translate = [self.cam_x, self.cam_y, self.cam_z, 1]
glLoadMatrixd(self.camera_matrix.get_inverse().to_opengl())
"""

#different camera version that follows ball
bx,by,bz = self.ball.pos
self.camera_matrix.x_axis = [1, 0, 0, 0]
self.camera_matrix.y_axis = [0, 0, -1, 0]
self.camera_matrix.z_axis = [0, 1, 0, 0]
self.camera_matrix.translate = [bx, 15, bz, 1]
glLoadMatrixd(self.camera_matrix.get_inverse().to_opengl())"""

#glLight(GL_LIGHT0, GL_SPECULAR, (0.1,0,0.0,1.0))
#glLight(GL_LIGHT0, GL_SPOT_DIRECTION, (10.0, -11, 0.0))
glLight(GL_LIGHT0, GL_POSITION, (20, 15, 20, 1.0))
#glLight(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1)
if collision_detect(self.board.ball, self.board.stoppers):
    obj = collision_detect(self.board.ball, self.board.stoppers)
    if self.board.ball.collpos != obj.mapcoords:
        self.board.ball.centrfy(10)
        self.board.ball.collpos = obj.mapcoords
        self.board.ball.reset_angles()

elif collision_detect(self.board.ball, self.board.rotators):
    obj = collision_detect(self.board.ball, self.board.rotators)
    if self.board.ball.collpos != obj.mapcoords:

```

```

self.board.ball.collpos = obj.mapcoords
for triangle in self.board.triangles:
    if triangle.align == 1:
        triangle.align = 3
    elif triangle.align == 2:
        triangle.align = 1
    elif triangle.align == 3:
        triangle.align = 4
    elif triangle.align == 4:
        triangle.align = 2
    self.board.prism_display_list = None
    self.collision_channel.play(self.sounds[2])

elif collision_detect(self.board.ball, self.board.ends):
    self.next_level()

elif collision_detect(self.board.ball, self.board.triangles):
    obj = collision_detect(self.board.ball, self.board.triangles)
    if self.board.ball.collpos != obj.mapcoords:
        ball = self.board.ball
        TOPLEFT = 1
        TOPRIGHT = 2
        BOTTOMLEFT = 3
        BOTTOMRIGHT = 4
        align = obj.align
        if ball.v.x > 0.0:
            if(align == TOPLEFT or align == BOTTOMLEFT):
                self.board.ball.v *= -1
            elif(align == TOPRIGHT):
                self.board.ball.v.x = 0
                self.board.ball.v.z = BALLSPEED
            elif(align == BOTTOMRIGHT):
                self.board.ball.v.x = 0
                self.board.ball.v.z = -BALLSPEED

        elif ball.v.z > 0.0:
            if(align == TOPLEFT or align == TOPRIGHT):
                self.board.ball.v *= -1
            elif(align == BOTTOMLEFT):
                self.board.ball.v.z = 0
                self.board.ball.v.x = BALLSPEED
            elif(align == BOTTOMRIGHT):
                self.board.ball.v.z = 0
                self.board.ball.v.x = -BALLSPEED

        elif ball.v.x < 0.0:
            if align == TOPRIGHT or align == BOTTOMRIGHT:
                self.board.ball.v *= -1
            elif(align == TOPLEFT):
                self.board.ball.v.x = 0
                self.board.ball.v.z = BALLSPEED
            elif(align == BOTTOMLEFT):
                self.board.ball.v.x = 0
                self.board.ball.v.z = -BALLSPEED

        elif ball.v.z < 0.0:
            if align == BOTTOMLEFT or align == BOTTOMRIGHT:
                self.board.ball.v *= -1
            elif(align == TOPLEFT):
                self.board.ball.v.z = 0

```

```

        self.board.ball.v.x = BALLSPEED
    elif(aligned == TOPRIGHT):
        self.board.ball.v.z = 0
        self.board.ball.v.x = -BALLSPEED
    ball.collpos = obj.mapcoords
    ball.reset_angles()
    self.collision_channel.play(self.sounds[1])

self.board.render(td)
#print self.board.ball.center_distance()

#-----
class Game:
    def __init__(self):
        pygame.display.init()
        if ISFULLSCREEN:
            self.screen = pygame.display.set_mode(SCREEN_SIZE, FULLSCREEN)
        else:
            self.screen = pygame.display.set_mode(SCREEN_SIZE, RESIZABLE)

        self.menu = gamemenu.Menu()
        self.editor = gameeditor.Editor()
        self.sequence = LevelSequence()
        self.render = GLRender()
        self.state = 0 #0 = menu, 1 = game 2 = editor

#-----
    def run(self, td):
        if self.state == 0:
            self.returned_screen = self.menu.run()
        elif self.state == 2 and self.editor.running:
            self.returned_screen = self.editor.run()
        self.draw(td)
        self.state_changes()

#-----
    def draw(self, td):
        if self.state == 0 or self.state == 2:
            self.screen.blit(self.returned_screen,(0, 0))
        if self.render.running:
            self.render.run(td)
        pygame.display.flip()

#-----
    def state_changes(self):
        #if in menu
        if self.state == 0:
            #start game
            if self.menu.action == 1:
                self.state = 1
                self.menu.action = 0
                for widget in self.menu.group:
                    if isinstance(widget, gamemenu.ListBox):
                        self.sequence.load_sequence(widget.active_item)
                self.render.start(self.sequence)
            #start editor
            elif self.menu.action == 2:

```



```

        self.state = 2
        self.editor.running = True
        self.editor.clear()
        self.menu.action = 0
    #quit
    if self.menu.action == 3:
        self.terminate()
        self.menu.action = 0
    #if in game
    elif self.state == 1:
        #if game was aborted
        if self.render.running == False:
            if ISFULLSCREEN:
                pygame.display.set_mode(SCREEN_SIZE, FULLSCREEN)
            else:
                pygame.display.set_mode(SCREEN_SIZE, RESIZABLE)
            #go back to menu
            self.state = 0
            self.render.mixer.quit()
            self.menu.bg = self.returned_screen
    #if in editor
    elif self.state == 2:
        if self.editor.running == False:
            if ISFULLSCREEN:
                pygame.display.set_mode(SCREEN_SIZE, FULLSCREEN)
            else:
                pygame.display.set_mode(SCREEN_SIZE, RESIZABLE)
            self.state = 0

#-----
def terminate(self):
    pygame.quit()

#-----
def main():
    pygame.mixer.pre_init(44100,16,1,2048)
    pygame.init()
    clock = pygame.time.Clock()
    game = Game()

    while(1):
        td = clock.tick(FPS)
        game.run(td)
if __name__ == "__main__":
    main()

```

```

"""gameeditor.py"""
import pygame
import os
from constants import *
from pygame.locals import *
from gamemenu import Button, ListBox, TextBox
from gamemap import LevelSequence

def get_image(imgsrc):
    if os.access('images/', os.F_OK):
        return pygame.image.load('images/'+imgsrc)
    else:
        print "images folder doesn't exist"

#-----
class GameSprite(pygame.sprite.Sprite):
    def __init__(self, drawpos, imgsrc, colorkey):
        pygame.sprite.Sprite.__init__(self)
        self.imgsrc = imgsrc
        self.src_image = get_image(imgsrc)
        self.image = self.src_image
        self.colorkey = colorkey
        if colorkey != None:
            self.image = self.src_image
            self.image.set_colorkey(colorkey)
        else:
            self.image = self.src_image.convert_alpha()
        self.rect = drawpos

    def highlight(self, borderw = 3, c = GREEN):
        pygame.draw.line(self.image, c, (0, 0), (TILESIZE-borderw, 0), borderw)
        pygame.draw.line(self.image, c, (0, 0), (0, TILESIZE-borderw), borderw)
        pygame.draw.line(self.image, c, (TILESIZE-borderw, TILESIZE-borderw), (TILESIZE-
borderw, 0), borderw)
        pygame.draw.line(self.image, c, (TILESIZE-borderw, TILESIZE-borderw), (0, TILESIZE-
borderw), borderw)

    def highlight_remove(self):
        self.image = get_image(self.imgsrc)
#-----
class GameObject(GameSprite):
    def __init__(self, pos, imgsrc, colorkey = None, key = None):
        self.key = key
        self.pos = pos
        x, y = pos
        x, y = x*TILESIZE, y*TILESIZE
        GameSprite.__init__(self, (x, y), imgsrc, colorkey)

#-----
def set_pos(self, pos):
    x, y = pos
    draw_x, draw_y = self.rect
    self.pos = pos
    if isinstance(self, Ball):
        self.rect = (x * TILESIZE + 10, y * TILESIZE + 10)
    else: self.rect = (x*TILESIZE, y*TILESIZE)

#-----
class Triangle(GameObject):
    def __init__(self, pos, alignment, imgsrc, colorkey = None, key = None):

```

```

GameObject.__init__(self, pos, imgsrc, colorkey, key)
self.align = alignment
self.orig_align = alignment
self.pos = pos
self.sc = str(alignment)
self.image = self.transform(self.src_image, self.align)
surf = pygame.Surface((40, 40))
self.imgsrc = imgsrc

if self.align == 1:
    surf.blit(self.image, (0, 0))
elif self.align == 3:
    surf.blit(self.image, (0, 12))
elif self.align == 4:
    surf.blit(self.image, (11, 11))
elif self.align == 2:
    surf.blit(self.image, (11, 0))
self.image = surf
self.image.set_colorkey(colorkey)
self.orig_image = self.image

def highlight(self, borderw = 3, c = GREEN):
    w, h = self.rect
    pygame.draw.line(self.image, c, (0, 0), (TILESIZE-borderw, 0), borderw)
    pygame.draw.line(self.image, c, (0, 0), (0, TILESIZE-borderw), borderw)
    pygame.draw.line(self.image, c, (TILESIZE-borderw, TILESIZE-borderw), (TILESIZE-
borderw, 0), borderw)
    pygame.draw.line(self.image, c, (TILESIZE-borderw, TILESIZE-borderw), (0, TILESIZE-
borderw), borderw)

def highlight_remove(self):
    self.image = get_image(self.imgsrc)
    self.image = self.transform(self.image, self.align)
    surf = pygame.Surface((40, 40))
    if self.align == 1:
        surf.blit(self.image, (0, 0))
    elif self.align == 3:
        surf.blit(self.image, (0, 11))
    elif self.align == 4:
        surf.blit(self.image, (11, 11))
    elif self.align == 2:
        surf.blit(self.image, (11, 0))
    self.image = surf

#-----
def rotate(self, move=1):
    """rotate triangle by 1=90, 2=180 or 3=270 degrees
    (2 + 1) % 4 = 3, (3 + 1) % 4 = 0 and so on"""
    self.align = (self.align + move) % 4
    if move == 1:
        self.image = pygame.transform.rotate(self.image, 90)
#-----
def transform(self, img, align):

    if align == 1:
        surf = pygame.transform.rotate(img, 90)
    elif align == 2:
        surf = img
    elif align == 3:

```

```

        surf = pygame.transform.rotate(img, 180)
    elif align == 4:
        surf = pygame.transform.rotate(img, 270)

    return surf

#-----
def restore(self):
    """restore to original alignment"""
    self.align = self.orig_align
    self.image = self.orig_image

#-----
#def update(self):
#    self.image = self.transform(self.orig_image, self.align)
#-----
class End(GameObject):
    """The finish object"""
    def __init__(self, pos, imgsrc, colorkey, key = None):
        GameObject.__init__(self, pos, imgsrc, colorkey, key)
        self.sc = 'e'

#-----
class Ball(GameObject):
    """The player-controllable ball object"""
    def __init__(self, pos, imgsrc, colorkey, key = None):
        GameObject.__init__(self, pos, imgsrc, colorkey, key)
        self.pos = pos
        x, y = pos
        #draw to center of tile
        self.rect = (x * TILESIZe + 10, y * TILESIZe + 10)
        self.sc = 's'

#-----
class Stop(GameObject):
    """The stopper object"""
    def __init__(self, pos, imgsrc, colorkey, key = None):
        GameObject.__init__(self, pos, imgsrc, colorkey, key)
        self.sc = 'q'

#-----
class Rotator(GameObject):
    """The rotator object"""
    def __init__(self, pos, imgsrc, colorkey, key = None):
        GameObject.__init__(self, pos, imgsrc, colorkey, key)
        self.sc = 'r'

#-----
class Tile(GameObject):
    def __init__(self, pos, imgsrc):
        GameObject.__init__(self, pos, imgsrc,)

#-----
class Editor:
    def __init__(self):
        self.screen = pygame.Surface(SCREEN_SIZE)
        self.selected_object = None
        self.selected_tile = None

        self.font = pygame.font.Font(FONT, 30)

```

```

#sprites drawn for selection
gap = 1
x = 22
self.menusprites = pygame.sprite.Group()
self.menusprites.add(Ball((x, 0), 'ball.png', BLACK, K_q))
self.menusprites.add(End((x, gap), 'end.png', BLACK, K_e))
self.menusprites.add(Triangle((x, gap * 2), 1, 'triangle.png', BLACK, K_1))
self.menusprites.add(Triangle((x, gap * 3), 2, 'triangle.png', BLACK, K_2))
self.menusprites.add(Triangle((x, gap * 4), 3, 'triangle.png', BLACK, K_3))
self.menusprites.add(Triangle((x, gap * 5), 4, 'triangle.png', BLACK, K_4))
self.menusprites.add(Stop((x, gap * 6), 'stop.png', BLACK, K_w))
self.menusprites.add(Rotator((x, gap * 7), 'rotator.png', BLACK, K_r))

#sprites drawn on the game area
tile = pygame.image.load('images/tile.png')
self.bg = pygame.Surface((MAPSIZE*TILESIZE, MAPSIZE*TILESIZE))
for row in range(MAPSIZE):
    for col in range(MAPSIZE):
        self.bg.blit(tile, (row * TILESIZE, col * TILESIZE))
self.sprites = pygame.sprite.Group()

self.widgets = pygame.sprite.Group()
h_b = 50
w_b = 100
x, y = SCREEN_SIZE
self.widgets.add(Button(MAPSIZE*TILESIZE+30, gap*9*TILESIZE, 20, "Prev", (w_b,
h_b)))
self.widgets.add(Button(MAPSIZE*TILESIZE+150, gap*9*TILESIZE, 20, "Next", (w_b,
h_b)))
self.widgets.add(Button(MAPSIZE*TILESIZE+30, gap*16*TILESIZE, 20, "Save", (w_b,
h_b)))

#populate maps
self.widgets.add(ListBox(MAPSIZE*TILESIZE+30, gap*11*TILESIZE, w_b, 100,
(125,221,163), self.populate_maps()))
self.widgets.add(TextBox(MAPSIZE*TILESIZE+30, gap*15*TILESIZE))
#map
self.sequence = LevelSequence()
self.map = [['0' for row in range(MAPSIZE)] for col in range(MAPSIZE)]
#map index for sequence
self.i = 0
self.running = False
#-----
def populate_maps(self):
    items = []
    for fn in os.listdir(os.getcwd()+'\\'+MAPS_FOLDER):
        if ".txt" in fn:
            items.append(fn)
    return items
#-----
def add_object(self, x, y, sc):
    for sprite in self.sprites.sprites():
        if sprite.pos == (x, y): self.sprites.remove(sprite)
        self.map[self.i][x][y] = '0'
    if sc == 's':
        if self.start_exists():
            self.move_start(x, y)
        for sprite in self.sprites:
            if isinstance(sprite, Ball):
                sprite.set_pos((x, y))

```

```

        else:
            self.sprites.add(Ball((x, y), 'ball.png', BLACK))
    elif sc == 'r':
        self.sprites.add(Rotator((x, y), 'rotator.png', BLACK))
    elif sc == 'q':
        self.sprites.add(Stop((x, y), 'stop.png', BLACK))
    elif sc == 'e':
        self.sprites.add(End((x, y), 'end.png', BLACK))
    else:
        self.sprites.add(Triangle((x, y), int(sc), 'triangle.png'))
    self.map[self.i][x][y] = sc
#-----
def start_exists(self):
    for row in range(MAPSIZE):
        for col in range(MAPSIZE):
            if self.map[self.i][row][col] == 's':
                return True
    return False
#-----
def move_start(self, x, y):
    for row in range(MAPSIZE):
        for col in range(MAPSIZE):
            if self.map[self.i][row][col] == 's':
                self.map[self.i][row][col] = '0'
    self.map[self.i][x][y] = 's'
#-----
def next(self):
    self.clear()
    self.i += 1
    print self.i, len(self.map), self.map
    if self.i == len(self.map):
        self.map.append(['0' for row in range(MAPSIZE)] for col in range(MAPSIZE))
    self.populate_map()
#-----
def prev(self):
    self.clear()
    self.i -= 1
    if self.i == -1: self.i = len(self.map)-1
    self.populate_map()
#-----
def prints(self):
    for row in range(MAPSIZE):
        for col in range(MAPSIZE):
            if self.map[self.i][row][col] != '0':
                print row, col, self.map[self.i][row][col]
    print self.sprites.sprites()
#-----
def clear(self):
    self.sprites.empty()
#-----
def load(self, fn):
    self.sequence.load_sequence(fn)
    self.map = list()
    self.clear()
    j = 0
    for lev in self.sequence.levels:
        mapp = ['0' for row in range(MAPSIZE)] for col in range(MAPSIZE)]
        for char in lev:
            mapp[j%MAPSIZE][j/MAPSIZE] = char

```

```

        j += 1
    j = 0
    self.map.append(mapp)
    self.i = 0
    self.populate_map()

#-----
def populate_map(self):
    for row in range(MAPSIZE):
        for col in range(MAPSIZE):
            if self.map[self.i][row][col] != '0':
                self.add_object(row, col, self.map[self.i][row][col])
#-----
def save(self, fn):
    try:
        f = open(MAPS_FOLDER+'\\'+fn+'.txt', 'w')
        for i in range(len(self.map)):
            for row in range(MAPSIZE):
                for col in range(MAPSIZE):
                    f.write(self.map[i][col][row])
                    f.write(chr(10))
                f.write(chr(10))
        f.close()
    except IOError, error: print error

#-----
def run(self):
    self.events()
    self.updates()
    self.draw()
    return self.screen
#-----
def events(self):
    for event in pygame.event.get():
        if(event.type == MOUSEBUTTONUP):
            for sprite in self.widgets:
                if isinstance(sprite, Button):
                    sprite.release_button()
                if sprite.mouse_pressed(event.pos):
                    if sprite.text == "Prev":
                        self.prev()
                    elif sprite.text == "Next":
                        self.next()
                    elif sprite.text == "Save":
                        for w in self.widgets:
                            if isinstance(w, TextBox):
                                self.save(w.text)
                            for w in self.widgets:
                                if isinstance(w, ListBox):
                                    w.items = self.populate_maps()
        if event.type == MOUSEBUTTONDOWN:
            if event.button == 1:
                x, y = event.pos
                x /= TILESIZE
                y /= TILESIZE
                if x < 20 and y < 20:
                    if self.selected_object != None:
                        self.add_object(x, y, self.selected_object.sc)
                for sprite in self.widgets:
                    if isinstance(sprite, Button):

```

```

        if sprite.mouse_pressed(event.pos):
            sprite.press_down()
    if isinstance(sprite, ListBox):
        if sprite.mouse_pressed(event.pos):
            x, y = event.pos
            bx, by = sprite.rect
            try:
                sprite.active_item = sprite.items[(y-by)/sprite.margin]
                self.load(sprite.active_item)
            except IndexError: return
    if isinstance(sprite, TextBox):
        if sprite.mouse_pressed(event.pos):
            sprite.active = True
        else: sprite.active = False
    sprite.update()
if event.button == 3:
    x, y = event.pos
    x /= TILESIZE
    y /= TILESIZE
    if x < 20 and y < 20:
        for sprite in self.sprites.sprites():
            if sprite.pos == (x, y):
                self.sprites.remove(sprite)
        self.map[self.i][x][y] = '0'

if event.type == KEYUP:
    if event.key == K_ESCAPE:
        self.running = False
if(event.type == KEYDOWN):
    active = False
    for sprite in self.widgets:
        if isinstance(sprite, TextBox):
            if sprite.active:
                sprite.update(event.key)
                active = sprite.active
    if not active:
        for sprite in self.menusprites.sprites():
            if event.key == sprite.key:
                temp = self.selected_object
                if temp != None:
                    temp.highlight_remove()
                self.selected_object = sprite
                sprite.highlight()

#-----
def updates(self):
    self.textsurf = self.font.render(str(self.i+1), True, (255, 0, 255), (0, 0, 0))
    if self.selected_object != None: pass
    for widget in self.widgets:
        if isinstance(widget, ListBox):
            widget.update()

#-----
def draw(self):
    self.screen.blit(self.bg, (0, 0))
    self.menusprites.draw(self.screen)
    self.sprites.draw(self.screen)
    self.widgets.draw(self.screen)
    self.screen.blit(self.textsurf, (21*TILESIZE-10, 10))

```



```

"""gamemap.py"""
from constants import *

#-----
class LevelSequence:
    def __init__(self):
        self.levels = list()
        self.i = 0

#-----
    def load_sequence(self, filename):
        self.levels = list()
        self.i = 0
        try:
            f = open(MAPS_FOLDER+'\\'+filename, 'r')
            map_str = f.read()
            f.close()
            str_list = map_str.splitlines()
            map_str = ""
            for row in str_list:
                map_str += row
                if(row == ""):
                    self.levels.append(map_str)
                    map_str = ""
            self.levels.append(map_str)
        except IOError, error: print error

#-----
    def get_level(self, listed = False):
        if listed:
            mapp = [['0' for row in range(MAPSIZE)] for col in range(MAPSIZE)]
            if not self.is_next_empty():
                j = 0
                for char in self.levels[self.i]:
                    mapp[j%MAPSIZE][j/MAPSIZE] = char
                    j += 1
            return mapp
        else: return self.levels[self.i]

#-----
    def next(self):
        self.i += 1

#-----
    def prev(self):
        if self.i > 0:
            self.i -= 1

#-----
    def is_last(self):
        if(len(self.levels)-1 == self.i):
            return True
        else:
            return False

    def is_next_empty(self):
        if(len(self.levels) == self.i):
            return True
        else:
            return False

```



```

        sprite.release_button()
    if sprite.mouse_pressed(event.pos):
        if sprite.text == "Play":
            self.action = 1
        elif sprite.text == "Editor":
            self.action = 2
        elif sprite.text == "Quit":
            self.action = 3
    if(event.type == MOUSEBUTTONDOWN):
        if(event.button == 1):
            for sprite in self.group:
                if isinstance(sprite, Button):
                    if sprite.mouse_pressed(event.pos):
                        sprite.press_down()
                if isinstance(sprite, ListBox):
                    if sprite.mouse_pressed(event.pos):
                        x, y = event.pos
                        bx, by = sprite.rect
                        try:
                            sprite.active_item = sprite.items[(y-by)/sprite.margin]
                        except IndexError: return
    if(event.type == KEYDOWN):
        if event.key == K_ESCAPE: pygame.display.quit()
        elif event.key == K_RETURN: self.action = 1

#-----
class ListBox(pygame.sprite.Sprite):
    def __init__(self, x, y, w, h, c, items, first_item = False):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((w, h))
        self.rect = (x, y)
        self.color = c
        self.items = items
        self.font = pygame.font.Font(FONT, 20)
        self.margin = 20

        if first_item: self.active_item = self.items[0]
        else: self.active_item = None

#-----
    def mouse_pressed(self, coords):
        x, y = self.rect
        ex, ey = coords
        w = self.image.get_width()
        h = self.image.get_height()
        if x < ex and ex < x + w and y < ey and ey < y + h:

            return True
        else:
            return False

#-----
    def update(self):
        i = 0
        self.image.fill((0, 0, 0))
        for item in self.items:
            if item == self.active_item:
                fontcolor = BLUE
            else: fontcolor = ORANGE
            #item[:-4] removes '.txt' extension

```

```

self.textsurf = self.font.render(item[:-4], True, fontcolor, BLACK)
self.image.blit(self.textsurf, (20, self.margin*i))
i+=1

```

```

#-----
class Button(pygame.sprite.Sprite):
    def __init__(self, x, y, fontsize = 20, text = None, size = (100, 50), bgcolor = ORANGE):
        pygame.sprite.Sprite.__init__(self)
        self.color = bgcolor
        self.rect = (x, y)
        self.image = pygame.Surface(size)
        self.image.fill(bgcolor)
        self.size = size
        self.text = text
        w, h = size
        pygame.draw.line(self.image, TEAL, (0, 0), (w, 0), 2)
        pygame.draw.line(self.image, TEAL, (0, 0), (0, h), 2)
        pygame.draw.line(self.image, BLUE, (w, h), (w, 0), 2)
        pygame.draw.line(self.image, BLUE, (w, h), (0, h), 2)

        self.font = pygame.font.Font(FONT, fontsize)
        self.textsurf = self.font.render(text, True, (255,255,255), bgcolor)
        #center_x = (w - (len(text) * fontsize)) / 2
        center_y = (h - fontsize) / 2
        self.image.blit(self.textsurf, (w/3, center_y))

        #Borders
        self.bw = 5
        self.release_button()

        self.pressed = False
        self.hovered = False

#-----
def mouse_pressed(self, coords):
    x, y = self.rect
    ex, ey = coords
    w = self.image.get_width()
    h = self.image.get_height()
    if x < ex and ex < x + w and y < ey and ey < y + h:
        return True
    else:
        return False

#-----
def release_button(self):
    w, h = self.size
    pygame.draw.line(self.image, TEAL, (0, 0), (w, 0), self.bw)
    pygame.draw.line(self.image, TEAL, (0, 0), (0, h), self.bw)
    pygame.draw.line(self.image, BLUE, (w, h), (w, 0), self.bw)
    pygame.draw.line(self.image, BLUE, (w, h), (0, h), self.bw)

#-----
def press_down(self):
    w, h = self.size
    pygame.draw.line(self.image, BLUE, (0, 0), (w, 0), self.bw)
    pygame.draw.line(self.image, BLUE, (0, 0), (0, h), self.bw)
    pygame.draw.line(self.image, TEAL, (w, h), (w, 0), self.bw)
    pygame.draw.line(self.image, TEAL, (w, h), (0, h), self.bw)

```

```

#-----
class TextBox(pygame.sprite.Sprite):
    def __init__(self, x, y, fontsize = 14, text = "", size = (100, 22)):
        pygame.sprite.Sprite.__init__(self)
        self.rect = (x, y)
        self.image = pygame.Surface(size)
        self.image.fill((255, 255, 255))
        self.size = size
        self.text = text
        w, h = size

        self.font = pygame.font.Font(FONT, fontsize)
        self.textsurf = self.font.render(text, True, (0,0,0), (255,255,255))
        #center_x = (w - (len(text) * fontsize)) / 2
        self.center_y = (h - fontsize) / 2 - 1
        self.image.blit(self.textsurf, (w/3, self.center_y))

        #Borders
        self.bw = 5
        self.draw_border()

        self.active = False

#-----
def update(self, char = ""):
    w, h = self.size
    #backspacing
    if char != "":
        try:
            if char == 8: self.text = self.text[:-1]
            else:
                self.text += chr(char)
        except ValueError: return

    self.textsurf = self.font.render(self.text + self.cursor(), True, (0,0,0), (255, 255, 255))
    self.image.fill((255, 255, 255))
    self.draw_border()
    self.image.blit(self.textsurf, (5, self.center_y))

#-----
def mouse_pressed(self, coords):
    x, y = self.rect
    ex, ey = coords
    w = self.image.get_width()
    h = self.image.get_height()
    if x < ex and ex < x + w and y < ey and ey < y + h:
        return True
    else:
        return False

#-----
def cursor(self):
    if self.active: return '|'
    else: return " "

#-----
def draw_border(self):
    w, h = self.size
    pygame.draw.line(self.image, TEAL, (0, 0), (w, 0), self.bw)
    pygame.draw.line(self.image, TEAL, (0, 0), (0, h), self.bw)
    pygame.draw.line(self.image, BLUE, (w, h), (w, 0), self.bw)
    pygame.draw.line(self.image, BLUE, (w, h), (0, h), self.bw)

```

```

"""constants.py"""
import pygame.font

images = \
{'ball'   : 'ball.png',
 'end'    : 'end.png',
 'rotator': 'rotator.png',
 'stop'   : 'stop.png',
 'triangle': 'triangle.png'}
#options = {'resolution': '640,480', 'fullscreen': 0}
#try:
#    f = file('config.txt')
#    f
MAPS_FOLDER = "maps"
BALLSPEED = 1

MSIZE = 20
TSIZE = 2.0
SCREEN_SIZE = (1280, 1024)
#SCREEN_SIZE = (1920, 1080)
ISFULLSCREEN = 0
GAMEWIDTH = 800
FPS = 60
Y_BASE = 1.0

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

MAPSIZE = 20
TILESIZE = 40

FONT = pygame.font.match_font('arial')

```