# PROJECT ALICE DOKUMENTOINTI

TKO_2111 Ohjelmoinnin harjoitustyö

Harjoitustyöohjeiden mukainen peli

Tekijät:

**Kurssilla:**

Botond Ortutay, Tekn. kand., tietotekniikka, Turun yliopisto, 523159, boti.ortutay@gmail.com

**Muut mukana olleet:**

Niklas Pirinen, Peliohjelmointi | insinööri (AMK) Kaakkois-Suomen ammattikorkeakoulu – Xamk, niklas.pirinen@gmail.com

Patrik Weck, Tekn. kand., tietotekniikka, Aalto-yliopisto, 786146, weck.patrik@gmail.com

# 1. Tehtävän kuvaus & Analysointi

Ohjelmoinnin harjoitustyö-kurssilla tehtävänä oli kehittää joko peli, kalenterisovellus, tai joku muu omavalintainen projekti. Meidän ryhmämme teki pelin nimeltä Project Alice. Vaatimukset harjoitustyön ohjeiden mukaiselle pelille olivat seuraavat: ohjelmakoodin lauseita tuli olla 100 jokaista ryhmän jäsentä kohden, eli meidän ryhmässä 3*100 = 300. Ohjelman piti käyttää listoja ja hajautustauluja, sisältää jonkinlaisen käyttöliittymän, sisältää vähintään kolme eri metodia, lukea ja kirjoittaa tiedostoja, sekä lukea aloitusdata ulkoisesta tiedostosta.  Jokaisen pelityyppisen harjoitustyön piti myös sisältää pelin tilan tallennusjärjestelmä.

Meidän projektissamme nämä vaatimukset täyttyvät. Lauseita meidän projektissamme on yhteensä 305 kappaletta. Lauseet on laskettu tekemällä ohjelmakoodista kopion, josta on poistettu kaikki tyhjät rivit ja kommentit ja tiivistetty hajautustaulujen määrittely yhdelle riville, jonka jälkeen käytimme bashin `wc -l` komentoa. Kommentit, tyhjät rivit ja hajautustaulujen määrittely mukaan laskettuna ohjelmassa on 384 riviä. Metodeja taas ohjelmassa on 12. Ohjelma käyttää myös listoja, esimerkkinä `main()` metodista löytyvä `pInventory` , sekä hajautustauluja, esimerkiksi `/resources/data/objects.json` tiedostosta tuotu `objects` .

Suurin osa ohjelman aloitusdatasta, kuten esimerkiksi kaikki pelin dialogi, renderöintiohjeet, kuvatiedostot joita peli käyttää renderöintiin ja esineisiin liittyvät tiedot tuodaan ulkopuolisista tiedostoista. Pelissä on kuitenkin muutamia yksittäisiä muuttujia, joille on asetettu aloitusarvo pelin alussa. Nämä ovat muuttujat `previousRoomID` , `currentRoomID` , `previousChatID` , `currentChatID` , `canChat` , `currentPuzzleSolved` , sekä `running` , jotka on pakko asettaa alkuun, koska pelin pääsilmukka tarkistaa ne joka suorituskerralla ja niiden virheelliset tai olemattomat arvot aiheuttaisivat pelille ongelmia. Peli on toteutettu graafisesti ja sisältää täten myös käyttöliittymän. Kuten jo mainittu, peli lukee aloitusdatansa ulkoisista tiedostoista ja tukee täten tiedosto-operaatioita. Peli kykenee myös kirjoittamaan tiedostoon vaatimusten mukaisen tallennusjärjestelmän käytön yhteydessä.

Peli on point-and-click tyylinen peli, joka saa pelaajalta syötteen mikäli pelikentälle renderöityjä objekteja klikataan tämä hoituu `pygame` kirjaston `pygame.event.get()` ja `pygame.Rect.collidepoint()` metodien avulla. Tulostaminen, eli käytännössä grafiikoiden renderöinti hoituu käymällä läpi hajautustaulua `objects[currentRoomID]` , joka sisältää tiedon kaikista ruudulla sillä hetkellä näkyvistä objekteista, niitä vastaavista kuvatiedostoista, sekä niiden koordinaateista. Kaikki hajautustaulun objektit piirretään ruudulle `pygame.Surface.blit()` metodin avulla, jonka jälkeen päivitetään näyttö käyttäen `pygame.display.flip()` metodia. Tämä aiheuttaa käytännössä sellaisen rajoituksen ohjelmalle, että renderöinti tapahtuu `objects[currentRoomID]` :n järjestyksen mukaan, joka tarkoittaa että ruudulle piirrettävät objektit voivat renderöityä toistensa päälle. Tämä rajoitus johti käytännön vaikeuksiin kehityksen aikana, koska

a) objektit katosivat toistensa taakse väärän järjestyksen takia

   ja

b) Peli rekisteröi syötteen sekä päällimmäisestä että alimmaisesta objektista jota klikattiin ja tämä johti ongelmiin.

Peliä myös suunniteltiin paljon laajemmaksi kuin mitä se tällä hetkellä on. Alun perin pelin oli tarkoitus sisältää neljä ”pakene huoneesta” tyyppistä pulmaa, mutta loppupeli sisältää tasan yhden. Pelistä myös leikattiin kesken kehtiyksen ominaisuuksia pois, kuten musiikki ja ajastetusti renderöityvät viestit. Olemme harkinneet jatkavamme tämän pelin kehitystä ja lisäävämme nämä ominaisuudet projektin palauttamisen jälkeenkin, mutta saa nähdä miten käy.

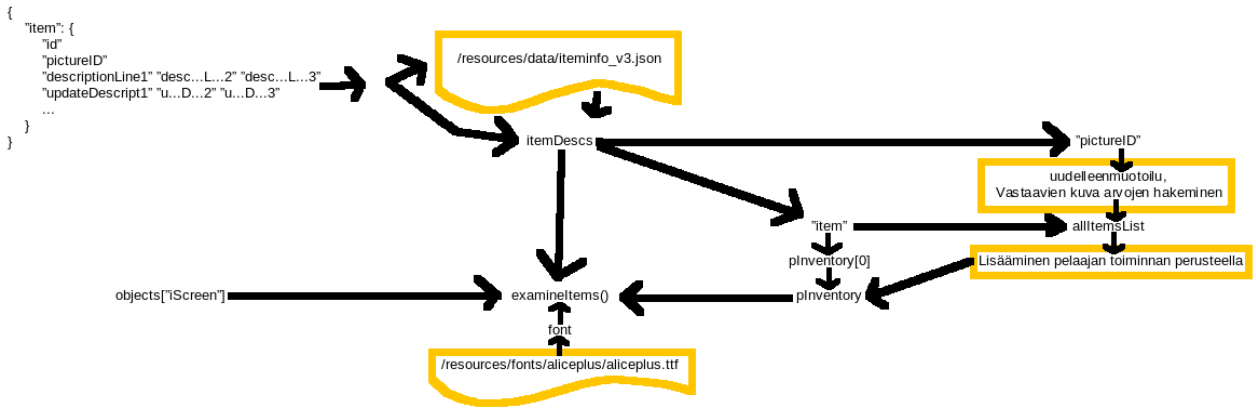
# 2. Ratkaisuperiaate

Kuten jo mainittu, peli saa syötteen, kun pelikentälle renderöityjä objekteja klikataan. Käyttämällä `pygame.Rect.collidepoint()` metodia, peli saa tietää mitä objektia klikataan. Tämä tieto, yhdessä muuttujan `currentRoomID` ja muutaman muun parametrin kanssa syötetään `action` metodille. `currentRoomID` sisältää tietysti tiedon siitä, missä pelin huoneista pelaaja on sillä hetkellä. Tämä tieto tarvitaan koska samoja objekteja voi olla useassa huoneessa. Esimerkiksi joka huoneessa on sitä huonetta vastaava taustakuva ( `background` ), joilla voisi olla eri huoneissa eri toiminnallisuus, jonka takia on tärkeää tietää missä huoneessa ollaan. `action` metodi palauttaa merkkijonon joka tallennetaan `currentRoomID` muuttujaan. Tämän jälkeen ohjelma tarkistaa, mikäli tämä uusi `currentRoomID` sisältää edelleen tiedon jostain huoneesta. Mikäli näin ei ole, on action ista palautettu jokin toimintaohje merkkijonona (esim. merkkijono `”SAVEGAME”` palautuu, kun pelaaja klikkaa tallennusikonia huoneessa `00n` tai `10w` ). Tällöin toteutetaan toimintaohjeen toiminta ja asetetaan `currentRoomID` arvoksi `previousRoomID` , jonka arvo on se, mikä `currentRoomID` arvona oli ennen action metodia.

Toinen tärkeä käyttö muuttujalle `currentRoomID` koskee objektien renderöintiä. Hajautustaulu `objects` sisältää tiedon kaikista pelin objekteista kategorisoituna hajautustauluihin huoneittain. Esimerkiksi hajautustaulu `objects[”00n”]` sisältää kaikki huoneeseen `00n` renderöitävät objektit. Koska `currentRoomID` sisältää tiedon siitä missä huoneessa ollaan milloinkin, kun peli renderöi pelikentän, se käyttää hajautustaulua `objects` avaimella `currentRoomID` saadakseen tiedon kaikista objekteista jotka juuri sillä hetkellä kuuluu renderöidä.

Pelin tarina käsittelee tilannetta, jossa tekoäly on murtautunut yrityksen tietojärjestelmiin ja esiintyy muina työntekijöinä chatissä. Pelaajan pitää kiertää yrityksen tyhjässä toimistossa etsien vihjeitä ja sitten puhua niistä chätissä muiden työntekijöiden kanssa. Pelissä on siis kaksi todella erityyppistä pelityyliä: vihjeiden etsiminen, sekä chattailu. Nämä molemmat pelityylit toimivat sen verran eri lailla, että olemme kehittäneet niitä varten kaksi eri metodia. Metodi `gameMode` on vastuussa chättiosuuksista. Se renderöi ja siirtää chattiviestit ja käsittelee kaiken niihin liittyvän tiedon. Tämä kuitenkin tarkoittaa, että pelin ”pääsilmukka”, joka hoitaa kaikki muut pelin toiminnot, keskeytyy `gameMode` n suorittamisen ajan, joka johtaa esimerkiksi siihen, että peliä ei voi sulkea `gameMode` n ajamisen aikana.

Chatin lisäksi toinen tärkeä pelin ominaisuus on vihjeiden keräily. Kun pelaaja klikkaa jotakin objektia pelikentällä niin hän voi saada vihjeenä toimivan esineen, josta hän sitten voi aloittaa keskustelun chatistä keräten lisää tietoa omasta tilanteestaan. Nämä tiedot tallennetaan pelaajan inventaarioon (lista `pInventory` ). Pelaaja voi itse nähdä oman inventaarionsa klikkaamalla reppu-ikonia huoneissa `00n` ja `10w` . Graafisen inventaarion renderöinnin hoitaa metodi `examineItems` , joka ottaa parametrinään hajautustaulun `itemDescs` , joka sisältää kaikki tiedot kaikista esineen peleistä (luettu tiedostosta `/resources/data/iteminfo_v3.json` ), listan `pInventory` , renderöintiohjeet hajautustaulusta `objects[”iScreen”]` sekä tekstirenderöintiä varten käytetyn fontin `font` (luettu tiedostosta `/resources/fonts/aliceplus/aliceplus.ttf` ). Alla graafinen havainnekuva `examineItems` metodiin syötettävistä arvoista:




Toimiakseen peli vaatii kirjastot `pygame` , `os` , `json` , `time` , sekä `re` . Mikäli näitä kirjastoja ei ole asennettuna, peli ei toimi. `pygame` on hyvin keskeinen kirjasto pelin ajamisen suhteen, sillä siinä olevilla metodeilla hoidetaan syötteen saaminen, sekä objektien renderöinti. `os` kirjastoa tarvitaan, jotta pelin tiedostorakenteessa olevat tiedostoja, kuten kaikki kuvatiedostot, tekstirenderöintiin tarvittava fontti, sekä ulkopuolisista tiedostoista tuleva data, voitaisiin lukea. Myös `json` kirjasto tarvitaan datan lukemista varten, sillä se mahdollistaa `.json` tyyppisten tiedostojen lukemisen. `time` kirjasto tarvitaan aikaa käsittelevien komentojen käyttöön ja `re` kirjasto tarvitaan monimutkaisempaan tekstinkäsittelyyn.

# 3. Metodien kuvaukset

## 3.1 `objects` -hajautustaulua muuttavat metodit

`objects` -hajautustaulu sisältää renderöintiohjeet kaikille koko pelissä näkyville esineille. Kuitenkin tietyt metodit, kuten `gameMode` ja `examineItems` lisäävät renderöitävien esineiden määrää täten muokaten `objects` -hajautustaulua. Seuraavien metodeiden tarkoitus on poistaa muiden metodeiden tekemiä muutoksia `objects` -hajautustauluun.

### 3.1.1 `emptyInventory`

Parametrit:

1. `ibg` , tyyppiä `pygame.surface` (eli käytännössä kuvadataa), graafisen inventaarion taustakuva
2. `x`, tyyppiä `pygame.surface`, ikoni jota käytetään valikoiden sulkemiseen

Palauttaa: hajautustaulun, joka sisältää tyhjän inventaarion renderöintiohjeet, tallennetaan `objects[”iScreen”]` iin

Ei ota parametriksi tietorakenteita.

### 3.1.2 `emptyChat`

Parametrit:

1. `screen_bg` , tyyppiä `pygame.surface` , chattiruudun taustakuva
2. `button` , tyyppiä `pygame.surface` , ”näkymätön napin paikka”

Palauttaa: hajautustaulun, joka sisältää tyhjän chattiruudun renderöintiohjeet, tallennetaan `objects[”monitor”]` iin

## 3.2 `updateItemDescription`

Tarkoitus: Muuttaa `itemDescs` hajautustaulussa olevat esineiden kuvaukset päivitetyiksi. Käytetään kun pelissä halutaan päivittää esineiden kuvaukset inventaariossa.

Parametrit:

1. `itemDescs` , hajautustaulu , sisältää tiedot kaikista esineistä
2. `item` , merkkijono , sisältää jonkin esineen ID:n, tarkoitus toimia avaimena hajautustaulussa `itemDescs` .

Ei varsinaista palautusta

Muutokset tietorakenteissa: `itemDescs` in indeksillä `item` : muuttaa `itemDescs[item]` alaisesta hajautustaulusta arvoja `"descriptionLine1"` , `"descriptionLine2"` ja `"descriptionLine3"` , sekä poistaa arvot `"updateDescript1"` , `"updateDescript2"` ja `"updateDescript3"` .

## 3.3 `saveGame`

Tarkoitus: Pakkaa tallennettavat tiedot oikeaan muotoon ja tallentaa pelin. Harjoitustyöohjeiden perusvaatimus.

Parametrit:

1. `pInventory` , lista , sisältää tiedot pelaajan hallussaan pitämistä esineistä
2. `itemDescs` , hajautustaulu , sisältää tiedot kaikista esineistä
3. `rID` , merkkijono , sisältää tiedon siitä, missä huoneessa pelaaja on parhaillaan (otetaan tässä muuttujasta `previousRoomID` , `currentRoomID` ei toimi, koska kun peliä tallennetaan siinä on merkkijono `"SAVEGAME"` )

Ei varsinaista palautusta, mutta kirjoittaa tiedostoon `/resources/data/savefile.json` .

Ei muokkaa sille parametreiksi annettuja tietorakenteita mitenkään.

## 3.4 Listan sisäisten listojen etsimiseen tarkoitetut metodit

Pelissä on useita listatyyppisiä muuttujia, jotka ovat listoja listoista, joiden ensimmäinen jäsen on merkkijonomuotoinen ID. Esimerkiksi pelaajan esineet sisältävä `pInventory` rakentuu seuraavalla tavalla:

```
    pInventory = [[esine1ID, esine1skin1, esine1skin2],[esine2ID, esine2skin1, esine2skin2]]
```

Seuraavien metodeiden tarkoitus on palauttaa tiettyjä tietoja näin rakennetuista listoista jonkun esineID :n avulla.

### 3.4.1 `returnListIncludingSearchable`

Parametrit:

1. `listOfLists` , mikä tahansa lista listoista joka rakentuu kohdassa 3.4 määritellyllä tavalla
2. `searchable` , merkkijono , 3.4 kohdassa määritellyn listoista koostuvaan listaan kuuluvan listan ID

Palauttaa: Ensimmäisen sellaisen listan, jonka ID on `searchable` .

Ei muokkaa sille parametreiksi annettuja tietorakenteita mitenkään.

### 3.4.2 `firstInAnyList`

Parametrit:

1. `listOfLists` , mikä tahansa lista listoista joka rakentuu kohdassa 3.4 määritellyllä tavalla
2. `searchable` , merkkijono , 3.4 kohdassa määritellyn listoista koostuvaan listaan kuuluvan listan ID

Palauttaa: Mikäli `listOfLists` sisältää sellaisen listan, jonka ID on `searchable` , palauttaa totuusarvon `True` (tosi), muuten palauttaa totuusarvon `False` (epätosi).

Ei muokkaa sille parametreiksi annettuja tietorakenteita mitenkään.

## 3.5 `examineItems`

Tarkoitus: Luo graafisen esityksen pelaajan hallussa olevista esineistä ja mahdollistaa niiden valitsemisen, jotta pelaaja pystyisi näkemään miten hän vuorovaikuttaa ympäristönsä kanssa.

Parametrit:

1. `invItems` , hajautustaulu , sisältää graafisen inventaarioruudun päivitysohjeet (otetaan hajautustaulusta `objects[”iScreen”]` ).
2. `pInventory` , lista , sisältää tiedot pelaajan hallussaan pitämistä esineistä
3. `itemDescs` , hajautustaulu , sisältää tiedot kaikista esineistä

4. `font` , tyyppiä `pygame.font.Font` (eli fontti) , tähän on ladattu aliceplus fontti tiedostosta `/resources/fonts/aliceplus/aliceplus.ttf` ) , käytetään tekstin renderöintiin.

Ei varsinaista palautusta

Muutokset tietorakenteissa: Lisää `invItems` iin renderöitäviä objekteja, sekä renderöitäviä tekstiobjekteja

## 3.6 `dialogue`

Tarkoitus: Ladata ID:n mukainen keskustelu tiedostosta `/resources/data/chat.json` . Käytetään pelin chattiosioissa.

Parametrit:

1. `id` , merkkijono , tunnus, jonka perusteella metodi tietää minkä keskustelun ladata

Palauttaa: lataamansa keskustelun (hajautustaulu)

Ei ota parametriksi tietorakenteita.

## 3.7 `action`

Tarkoitus: joka kerta kun jotakin objektia klikataan pelialueella niin `action` palauttaa joko uuden roomID:n jonne siirtyä, tai merkkijonon, joka kertoo ohjelmalle kuinka toimia. Mahdollistaa pelialueella liikkumisen ja siihn vaikuttamisen.

Parametrit:

1. `inp` , merkkijono , sen objektin ID, jota klikattiin
2. `rID` , merkkijono , sisältää tiedon siitä, missä huoneessa pelaaja on parhaillaan (otetaan tässä muuttujasta `currentRoomID` )
3. `prevRID` , merkkijono , sisältää tiedon siitä, missä huoneessa pelaaja oli ennen nykyiseen huoneeseen tuloa (otetaan muuttujasta `previousRoomID` )
4. `pInventory` , lista , sisältää tiedot pelaajan hallussaan pitämistä esineistä

5. `currentPuzzleSolved` , totuusarvo , sisältää tiedon siitä, onko pelin juuri meneillään oleva pulma ratkaistu vai ei

Palauttaa: merkkijonon, joka on joko seuraavan huoneen roomID, tai joku toimintaohje pelille, esimerkiksi `"SAVEGAME"`

Ei muokkaa sille parametreiksi annettuja tietorakenteita mitenkään.

## 3.8 `gameMode`

Tarkoitus: Siirtää, sekä renderöidä kaikki chatin viestit, vastaa pelin chattiosiosta.

Parametrit:

1. `chatID` , merkkijono , tunnus, jonka perusteella metodi tietää minkä keskustelun ladata
2. `font` , tyyppiä `pygame.font.Font` (eli fontti) , tähän on ladattu aliceplus fontti tiedostosta `/resources/fonts/aliceplus/aliceplus.ttf` ) , käytetään tekstin renderöintiin
3. `monitorObjects` , hajautustaulu , sisältää chättiruudun grafiikoiden päivitysohjeet (otetaan hajautustaulusta `objects["monitor"]` ).
4. `screen` , tyyppiä `pygame.Surface` , itse näyttö, eli se pinta jolle kaikki muut objektit renderöidään.
5. `button` , tyyppiä `pygame.surface` , ”näkymätön napin paikka”
6. `x`, tyyppiä `pygame.surface`, ikoni jota käytetään valikoiden sulkemiseen

Ei palauta mitään

Muutokset tietorakenteissa: Lisää `monitorObjects` iin renderöitäviä tekstiobjekteja, sekä muokkaa siinä olevia objekteja.

## 3.9 `renderObjects`

Tarkoitus: Luoda lista kaikista objekteista, ja samalla renderöidä ne, jotta pelin grafiikat päivittyisivät seuraavalla ruudun päivityksellä

Parametrit:

1. `screen` , tyyppiä `pygame.Surface` , itse näyttö, eli se pinta jolle kaikki muut objektit renderöidään.
2. `roomID` , hajautustaulu , sen hetken huoneeseen renderöitävät objektit (otetaan hajautustaulusta `objects` avaimella `currentRoomID` )

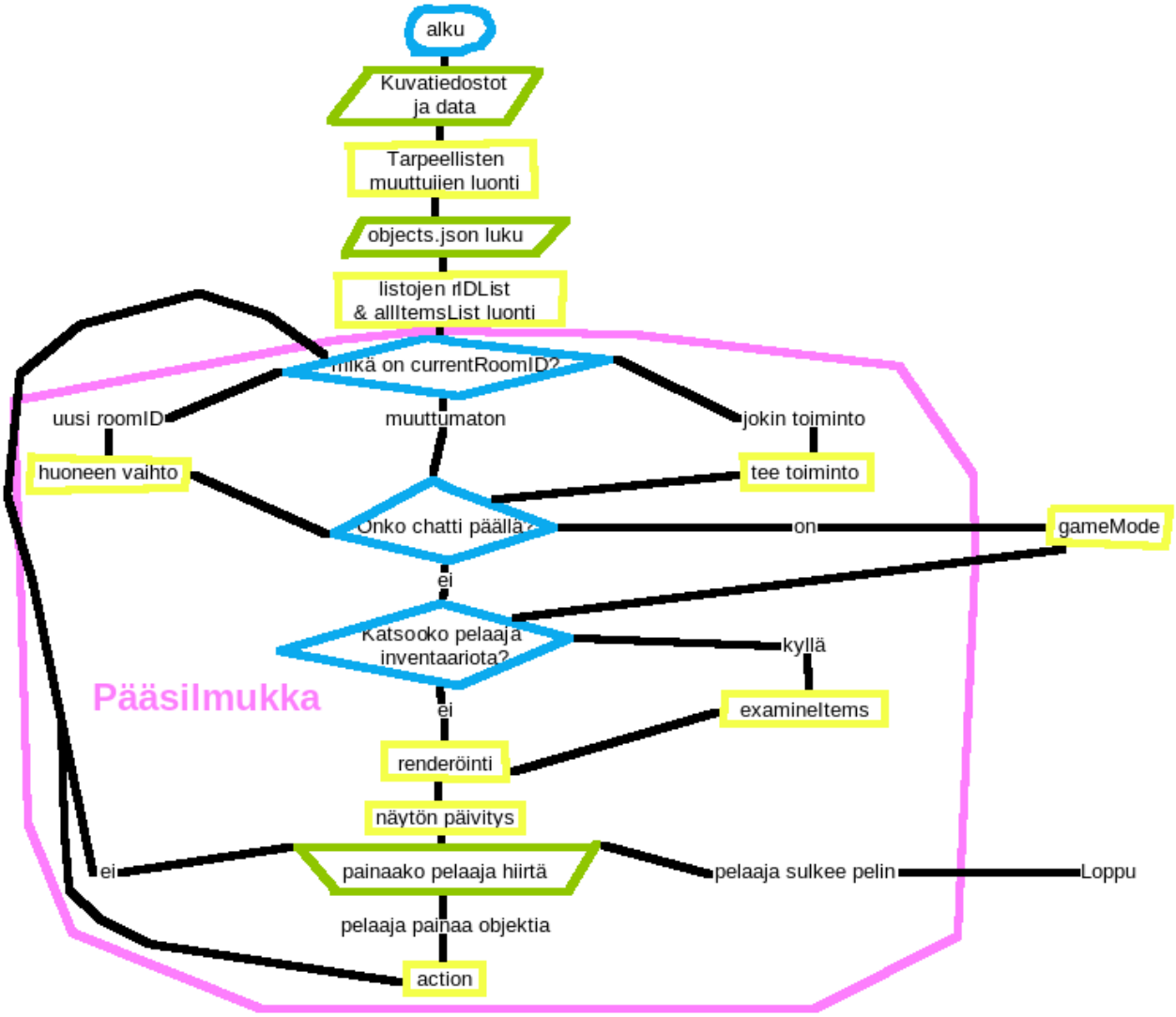
Palauttaa: luomansa listan

Ei muokkaa sille parametreiksi annettuja tietorakenteita mitenkään.

## 3.10 `main`

Tämä metodi on erilainen kuin kaikki muut metodit, sillä tämä on pelin päämetodi. Tämä tarkoittaa, että kun pelin ajaa tämä metodi suoritetaan. Tämä metodi toimi seuraavalla tavalla: ensin se lataa suurimman osan pelin kuvatiedostoista ja datasta, luo ne muuttujat joita tarvitaan pääsilmukan käynnistämiseen, lataa pelin käyttöön `objects` hajautustaulun, sekä luo annetun datan avulla listat `rIDList` ja `allItemsList` . Tämän jälkeen käynnistetään pelin pääsilmukka. Jokaisen silmukan alussa tarkistetaan muuttuja `currentRoomID` ja sen statuksen perusteella joko ei tapahdu mitään, vaihdetaan huonetta tai tehdään joku siihen kirjattu toiminta. Seuraavaksi tarkistetaan mikäli chattimuoto on päällä (eli `canChat` on tosi ja `currentRoomID` on `"monitor"`). Mikäli chattimuoto on päällä, keskeytetään silmukka, suoritetaan `gameMode` ja jatketaan pääsilmukan ajoa. Mikäli chattimuoto ei ole päällä, tätä ei tehdä. Sitten tarkistetaan katsooko pelaaja juuri inventaariotaan ja mikäli katsoo ajetaan metodi `examineItems` . Seuraavaksi renderöidään, päivitetään näyttö, tarkistetaan mitä pelaaja tekee hiirellä ja tarvittaessa ajetaan metodi `action`. Kun tämä kaikki on tehty silmukkaa toistetaan kunnes pelaaja lähtee pelistä.

Alla graafinen esitys `main` silmukan toiminnasta:




# 4. Testausjärjestelyt

Tämän pelin osalta testausstrategia koostui pääosin siitä, että peliä tallennettiin ja ajettiin todella usein ja mikäli havaittiin jokin virhe, sitä alettiin välittömästi korjaamaan. Lisäksi pelin kehittäminen perustui hyvin pitkälle eri versioiden luomiseen. Kun peliin lisättiin jokin suuri ja pelin logiikkaa olennaisesti muuttava ominaisuus, tehtiin kopio pelin lähdekoodista, päivitettiin versionumero ja siirrettiin vanha versio varmuuskopioksi. Lisäksi joitain yksinkertaisempia pelin metodeja kehitettiin käyttämällä Python 3 komentoriviä. Alla esimerkiksi ruudunkaappaus siitä kun saimme vihdoin metodin `firstInAnyList` toimimaan komentorivillä:

```
>>> listOfLists = [["a", "b", "c"], ["d", "e", "f"], ["g", "h", "i"]]
>>> d = "d"
>>> for lInList in listOfLists:
...     if lInList[0] == d:
...             print(True)
...
True
>>> d = "Pohjois-Korea"
>>> for lInList in listOfLists:
...     if lInList[0] == d:
...             print(True)
...
>>>
```

Varsinaisen pelin käyttämä metodi on suurimmalta osin suoraan kopioitu komentorivikokeiluista.

# 5. Liitteet

Dokumentin liitteenä alkuperäinen tehtävänanto.