Projektityön dokumentti

Father Long Legs

1. **Henkilötiedot**

   Father Long Legs; Jonathan Leinola, 477329, AIT, 20.4.2019.

2. **Yleiskuvaus**

   Father Long Legs on tasohyppelypeli, jossa pelaaja ohjaa hahmoa (Father Long Legs) kentän läpi hyppimällä erilaisten esteiden ja tasojen päälle. Hahmo on Isä hämähäkki, jonka tarkoitus on pelastaa perheensä suorittamalla kenttiä.

   Tasohyppelypeliin kuuluu graafinen käyttöliittymä ja toimiva törmäyksen tunnistus. Jos kentän pääsee kokonaan loppuun asti, voittaa silloin henkilö pelin. Häviö tapahtuu, jos henkilö ei onnistu ohjaamaan hahmoa kentän loppuun asti tietyn ajan kuluessa.

   Pelissä on kaksi kenttää, joidenka grafiikat koostuvat neliöistä. Hahmo on hämähäkki, jonka asento päivittyy sitä liikuttamalla tai hyppimällä. Pelin lisäksi kenttien suunnitteluun on erillinen kenttäeditori, sillä erilaisten kenttien tekeminen helpottuu huomattavasti.

   Lähtö oletuksena oli keskittyä keskivaikeaan tasoon, mutta kenttäeditori lisäsi hieman projektin vaikeustasoa.

3. **Käyttöohje**

   Pääohjelma:

   Pääohjelma käynnistetään ajamalla game.py tiedostoa. Jotta game.py tiedosto toimisi on käyttäjän asennettava pygame-kirjasto. Helpoin tapa asentaa pygame-kirjasto on käyttää pip-työkalua seuraavanlaisesti komentorivillä:

```
python3 -m pip install -U pygame --user
```

   Käynnistysvalikko:

   Käyttäjä voi valita nuolinäppäimillä joko start- tai quit- toiminnon. Peli lähtee käyntiin valitsemalla enter-näppäimellä start-toiminnon. Ohjelma suljetaan valitsemalla quit-toiminnon. Pelin voi aloittaa uudelleen painamalla ”r”-näppäintä.

   Hahmon toiminnot:

   Liikkuminen: Nuolinäppäimillä

   Hyppääminen: Välilyöntinäppäimellä

   Kenttäeditori:

   Kenttäeditori käynnistetään samalla tavalla kuin pääohjelma ja sekin tarvitsee toimiakseen pygame-kirjaston. Kenttäeditoria ajetaan leveleditor.py tiedostosta.

   Kenttäeditorin toiminnot:

   Tallenna kenttä: Enter-näppäin

Esteiden piirtäminen: valitsemalla hiiren vasemmalla näppäimellä pisteen ja pitämällä näppäintä pohjassa, kunnes este/taso on halutunlainen.

Esteiden/tasojen poistaminen: Edellisen esteen/tason poistaminen tapahtuu painamalla backspace-näppäintä.

4. **Ulkoiset kirjastot**

Ohjelmistossa käytetään seuraavia kirjastoja: random, os, sys, math ja re. Random kirjastoa tarvitaan satunnaisten lukujen arpomiseen. Moduulin os kautta voidaan hakea tietoja tiedostojärjestelmästä.

Sys-moduulia tarvitaan exit-metodin käyttöön ohjelman loppupuolella päättämään ohjelman suoritus. Math-kirjastoa taas tarvitaan vihollisen sijainnin laskemiseen. Moduulin re avulla voidaan käyttää säännöllisiä lausekkeita merkkijonoja käsittelyssä, jotka nousivat kenttien luomisessa tärkeään asemaan. Säännöllisten lausekkeiden avulla voidaan esimerkiksi etsiä tietynlaisia merkkijonoja.

Lisäksi ohjelmassa käytetään ulkoista Pygame-kirjastoa, joka on alustariippumaton Simple DirectMedia Layerin (SDL) päälle rakennettu moduuli. Pygame on tarkoitettu peliohjelmointiin ja se sisältää grafiikka- ja äänikirjastot.

5. **Ohjelman rakenne**

Ohjelman rakenne jakautuu seuraaviin pääluokkiin:

- Player
- Enemy
- Platform
- MovingPlatform
- Level
- Spritesheet

Player-luokassa määritellään mitä kohtaa ruutua pelaaja kontrolloi. Tässä tapauksessa pelaaja on ruudun alareunassa, ja kun liikkuminen tapahtuu, päivitetään näyttöä sen mukaan. Player luokassa keskeiset metodit ovat pelaajan liikkuminen (update), pelaajan hyppääminen (jump) ja gravitaation laskeminen pelaajalle (grav). Platform-luokassa määritetään taso, jolle pelaaja pystyy hyppäämään.

Enemy-luokassa määritellään viholliset. Tämän luokan metodit ovat update ja getposition. Update-metodin avulla päivitetään vihollista etenemään hahmon suuntaan.

Platform-luokassa asetetaan parametrit tasoille ja MovingPlatform-luokassa update-metodin avulla tasot saadaan liikkumaan.

Level-luokka tarvitaan eri kenttien määrittelyyn. Sen keskeisimpiä metodeja on päivittää kenttä (update), piirtää kaikki tarvittavat objektit (draw) ja näytön päivittäminen pelaajaluokan mukaisesti. Update-metodia tarvitaan siihen, että tarvittavat tiedot tulevat päivitetyksi ruudulle. Kun luodaan useita kenttiä, nämä kentät perivät Level-luokan seuraavanlaisesti:

```
Class level_1(level):
```

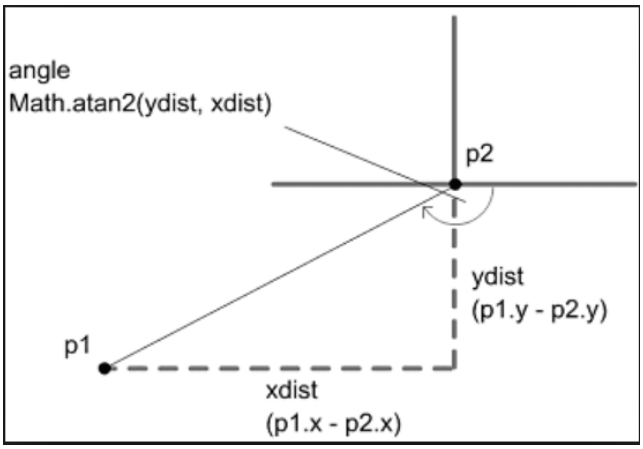
Spritesheet-luokan tarkoituksena on lukea kuvatiedostoja ja ottaa sieltä tarvittavat kuvat päivittääkseen hahmoa, vihollisia tai kenttää. Spritesheet-luokan tärkein metodi on hakea kuva (getimage), jossa kuva haetaan pelin kuvakansiosta ja iso kuva paloitellaan oikean kokoisiksi. Nämä kuvat asetetaan esimerkiksi Player-luokassa hahmon kuvaksi.

Perinnästä vielä, koska ohjelmassa käytetään pygame-kirjastoa, perivät luokat player, enemy ja platform luokan pygame.sprite, joka on yksinkertainen luokka peliobjektien luomiseen. Lisäksi liikkuva taso MovingPlatform-luokka perii Platform-luokan, joka taas saa tason liikkumaan.

6. Algoritmit

Vihollisen käyttäytymiseen olisi voitu käyttää ns. polunetsintä algoritmia. Polunetsinnän tarkoituksena on löytää mahdollisimman optimaalinen reitti pisteestä toiseen pisteeseen. Yksi näistä algoritmeista on nimeltään A*-algoritmi, joka on hahmon tekoälyssä käytetty algoritmi ratkaisun etsimiseen. Jotta kyseistä algoritmia olisi voitu käyttää, pelin kentät olisi pitänyt jakaa erillisiin neliöihin, jotta A*-algoritmi toimisi halutulla tavalla.

Sen sijaan pelissä käytetään vihollisen sijainnin laskentaan yksinkertaista geometrista matematiikkaa. Vihollisen ja pelaajan välinen kulma lasketaan seuraavanlaisesti:



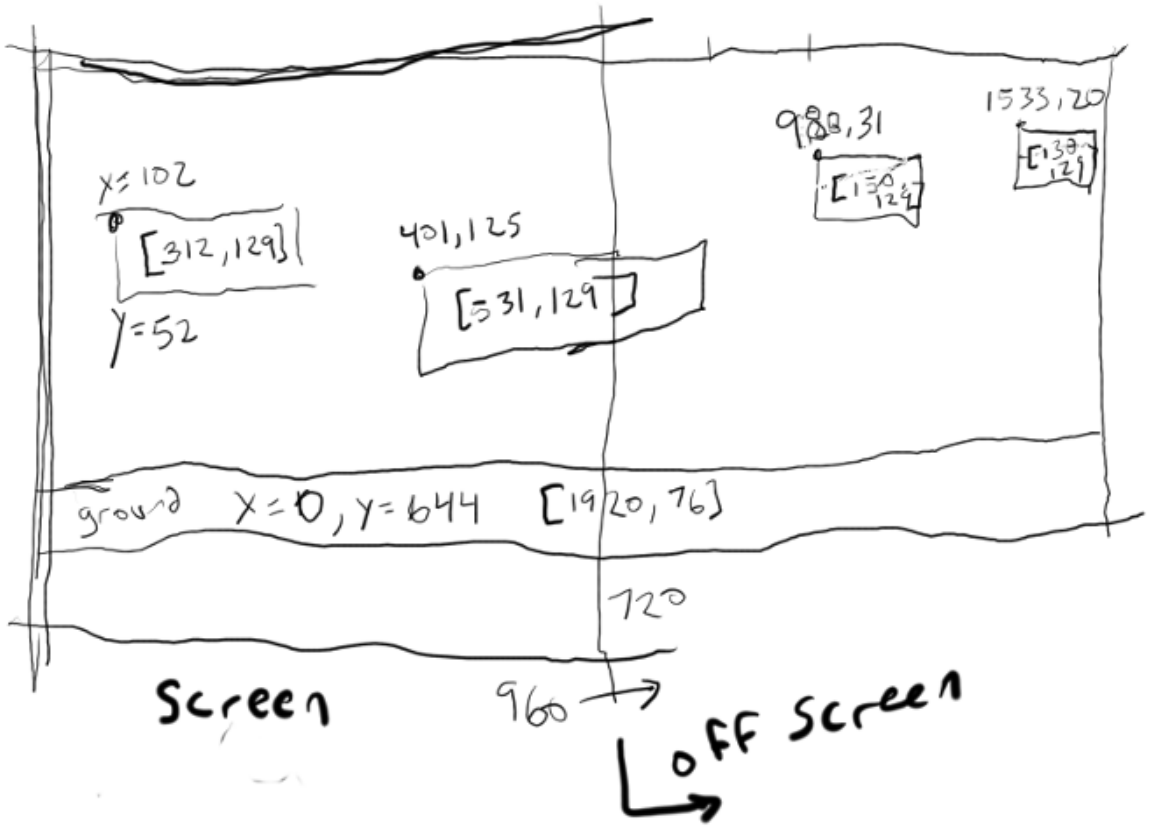
Kuva 1 Vihollisen ja pelaajan pisteiden välinen kulma

Kulman laskennan jälkeen vihollista liikutetaan asetetun kulman suuntaan asetetulla nopeudella. Tässä tapauksessa ei huomioida esteitä, joten vihollinen liikkuu niiden läpi.

7. Tietorakenteet

Ohjelmassa tarvitaan useita listoja ja yksi esimerkki niistä on matriisien esittämistä varten. Matriiseissa voi olla esimerkiksi kenttien luomiseen tarvittavat esteet, jotka luodaan seuraavanlaisesti:

level = [[50, 70, 500, 500], [x, x, x, x],jne.]



Kuva 2 Esimerkki kentästä ja sen objekteista

Listan ensimmäinen alkio kertoo esteen leveyden, korkeuden, sekä x- ja y-koordinaatit. Tasojen suunnittelu on tietysti tällä tavalla hieman raskauttavaa, siksi kenttäeditori osoittautui todella käteväksi. Sen lisäksi, esimerkiksi pygamessa monikulmioiden pygame.draw.polygon tai suorakulmioiden pygame.draw.rect piirtämisen annetaan parametrina lista pisteitä, jotka ilmaisevat x- ja y-koordinaatteina mihin kohtaan näyttöä monikulmio piirretään.

8. **Tiedostot**

Kenttäeditori tallentaa kentän level.txt tiedostoon, jossa esteet tai tasot ovat esitetty seuraavanlaisesti:

```
210 30 450 570
210 30 850 420
210 30 1000 520
200 30 1120 280
40 300 450 300
40 150 450 650
30 450 850 420
30 200 1000 520
100 750 1200 0
40 800 1400 0
40 50 1400 400
```

Kuva 3 level1.txt

Level.txt tiedostossa objektit ovat järjestely riveittäin, jossa ensimmäinen alkio kertoo esteen leveyden, korkeuden, sekä x- ja y-koordinaatit. Tästä tiedostosta pääohjelman on helppo lukea rivit eri objekteiksi ja siten piirtää ne kenttään seuraavalla tavalla:

level = [[210, 30, 450, 570], [210, 30, 850, 420], jne.]

Pygame-kirjaston avulla esimerkiksi suorakulmion piirtäminen tapahtuu kutsumalla pygame.rect() -oliota, joka tallentaa seuraavanlaisesti ylläolevat parametrit:

Rect(left, top, width, height)

9. **Testaus**

Projektisuunnitelmassa esitettiin ohjelman testaukselle seuraavanlaisia tapoja:

1. Ohjelmassa keskeisimpiä metodeja ovat gravitaation laskeminen, hahmon hyppääminen ja liikkuminen ja tietysti kentän piirtäminen. Näiden lisäksi hahmon liikkumisen rajaaminen ja törmäyksentarkistus on syytä tarkistaa toimivaksi.
2. Gravitaatiota laskiessa on syytä tarkastaa, jos hahmo on maassa niin sen pitäisi palauttaa arvo 0. Muussa tapauksessa sen pitäisi palauttaa arvo, joka on erisuuri kuin 0.
3. Hahmon liikkumisen rajauksen testaaminen testataan siten, että verrataan, onko hahmon neliö päivitetyn tason ulkopuolella. Mikäli hahmo on tason ulkopuolella, palautetaan error exception.
4. Törmäyksentarkistus tapahtuu siten, että verrataan pelaajan neliön nurkkaa kentän tasojen nurkkien pisteisiin. Jos pisteet ovat päällekkäin, silloin törmäyksentarkistus palauttaa error exceptionin.

Todellisuudessa erilaiset testitapaukset osoittautuivat hieman erilaisiksi.

1. Ohjelmoinnin aikana testaus tapahtui pitkälti yrityksen ja erehdyksen kautta, mikä ei välttämättä ollut aina optimaalisin tapa.
2. Erillistä testiä gravitaation laskemiseen ei ollut tarvetta tehdä ohjelmiston suunnittelun tai toteutuksen aikana.
3. Ohjelmassa ei ollut tarvetta testata hahmon liikkumisen rajausta, muuten kuin törmäyksentarkistuksella.
4. Pygame-kirjastossa on metodi pygame.sprite.spritecollide(). Sen avulla voidaan todeta, että osuuko hahmo tasoon ja tehdä tarvittavat muutokset hahmon liikuttamiseen ohjelmassa.

10. **Ohjelman tunnetut puutteet ja viat**

Ohjelman ongelmat liittyvät luokkien sisällä oleviin toimintoihin. Luokissa on yhdistelty pelin logiikkaan liittyviä toimintoja sekä pelin graafisia toimintoja. Näiden vuoksi ohjelmaa on huomattavasti vaikeampaa laajentaa, kuin jos logiikkaan liittyvät toiminnot ja graafiset toiminnot olisivat eri luokan vastuulla.

Pääohjelmassa on muutamia puutteita, joita en saanut korjatuksi. Pelissä huomataan, että hahmo saattaa liikkua ei haluttuun suuntaan, jos lähistöllä on useampia tasoja. Ongelmat johtuvat hahmon position päivityksestä, sillä kun sitä päivitetään, saatetaan asettaa hahmon positio väärään paikkaan.

11. **3 parasta ja 3 heikointa kohtaa**

Kokonaisuudessaan ohjelmassa parhaaksi ominaisuudeksi osoittautuu kenttäeditori. Kenttäeditorin avulla on helppo luoda uusia kenttiä, jos edelliset kaksi käyvät tylsiksi tai jopa mahdottomiksi. Pääohjelmassa on mahdollisuus luoda melko helposti liikkuvia

objekteja, jotka toteutetaan luokassa MovingPlatform(). Spritesheet-luokan avulla on taas kohtalaisen helppoa luoda sprite-grafiikkaa, jota hahmon toteutuksessa käytetään. Sen avulla muiden hahmojen, kuten vihollisten luominen peliin käy melko kätevästi.

Ohjelmassa on myös muutamia heikkoja kohtia edellisen kohdan 10 lisäksi, joita voisi parantaa. Kenttäeditorissa on suuria heikkouksia, sillä editorilla voidaan luoda vain yhdenlaisia tasoja. Sen lisäksi editori luo tiedot level.txt tiedostoon, josta tiedot joudutaan manuaalisesti siirtämään level1.txt tai level2.txt tiedostoon. Pääohjelmassa yksi heikkous on, että vihollinen ei tunnista ollenkaan tasoja, joten se liikkuu tasojen läpi.

12. **Poikkeamat suunnitelmasta**

Projektisuunnitelman aikataulu toteutui työvaiheittain melko hyvin, aloitusnäytön ohjelmointi siirtyi viikkokalenterissa myöhemmälle ja kenttäeditorin lisääminen toi suunnitelmaan yhden vaiheen lisää.

Eri vaiheiden mukaan työtunteja on yhteensä noin 108, joka tekee keskimääräinen n. 12 työtuntia per viikko, 9 viikon ajalle. Alkuperäisen suunnitelman mukaan työtunteja oli yhteensä noin 32-46h joten työmäärä vähintäänkin tuplaantui.

Toteutunut työjärjestys muuttui hieman alkuperäisestä, sillä mukaan tuli myös kenttäeditorin luominen.

13. **Toteutunut työjärjestys ja aikataulu**

Lopulta työ jakautui eri osiin, jotka etenivät seuraavien vaiheiden mukaisesti:

1. Vaihe 16h:

    Aloitusnäytön ohjelmointi ja implementointi.

2. Vaihe 24h:

    Player-luokan, platform-luokan ja level-luokan ohjelmointi:

3. Vaihe 8h:

    Kenttien luominen (max 2 kenttää)

4. Vaihe 12 h:

    Spritesheet luokan ohjelmointi

5. Vaihe 24h:

    Spritejen luominen

6. Viimeistelyä 14h

    Enemy-luokan luominen

    MovingPlatform-luokan luominen

    Peliäänien lisäys

    Yksikkötestausta

    Yleisilmeen parantelua

7. Vaihe 12h:

    Kenttäeditorin luominen

Viikko toteuma:

1. Viikko 9:

    Vaihe 2

2. Viikko 10

    Vaihe 3 ja 4

3. Viikko 11

    Vaihe 4 ja 5

4. Viikko 12

    Vaihe 6

5. Viikko 13:

    Vaihe 1

6. Viikko 14:

    Projektissa ei edistymistä

7. Viikko 15:

    Projektissa ei edistymistä

8. Viikko 16:

    Vaihe 7

8. Viikko 17:

    Vaihe 6

14. **Arvio lopputuloksesta**

    Ensimmäiseksi ohjelmistoprojektiksi työn laatu oli kohtalaisella tai hyvällä tasolla. Suurin ohjelman laatuun vaikuttava tekijä on se, että peliloogisia ja graafisia toimintoja on yhdistetty, jolloin laajennusten tekeminen on hieman hankalaa, varsinkin ohjelmoijalle, joka ei tunne kyseistä ohjelmaa hyvin. Tulevaisuudessa olisi siis ensimmäiseksi hyvä tehdä nämä erikseen.

    Ohjelmassa tietorakenteet ja luokkajaot onnistuivat mielestäni hyvin, sillä esimerkiksi Level()-luokan avulla on helppo tehdä uusia kenttiä ohjelmaan. En näe, että luokkajakoja olisi voinut tehdä paljon paremmin tässä valossa. Tietorakenteet olivat myös yksinkertaisia ja helposti laajennettavissa. Esimerkiksi suoraan tekstitiedostosta ladatut suorakulmaiset taso-objektit onnistuivat kätevästi.

    Ohjelmassa on potentiaalia viedä sitä seuraavalle tasolle, sillä yksinkertaiset pelilogiikat ja kentän luomiseen tarvittavat työkalut ovat aluillaan. Toisaalta koodi on melko helppo lukuista, joten laajennusten tekeminen ei ole mahdottomuus.

15. **Viitteet**

https://fi.wikipedia.org/wiki/Sprite-grafiikka

https://en.wikipedia.org/wiki/Game_testing

https://www.theseus.fi/bitstream/handle/10024/21239/kuisma_joentakanen.pdf?sequence=1

https://linkki.github.io/python2017/pygame.html

http://programarcadegames.com/index.php?chapter=example_code_platformer

https://www.pygame.org/news

https://www.cs.helsinki.fi/group/linkki/materiaali/peliohjelmointi/lista.html

https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.Sprite

https://opensource.com/article/18/7/put-platforms-python-game

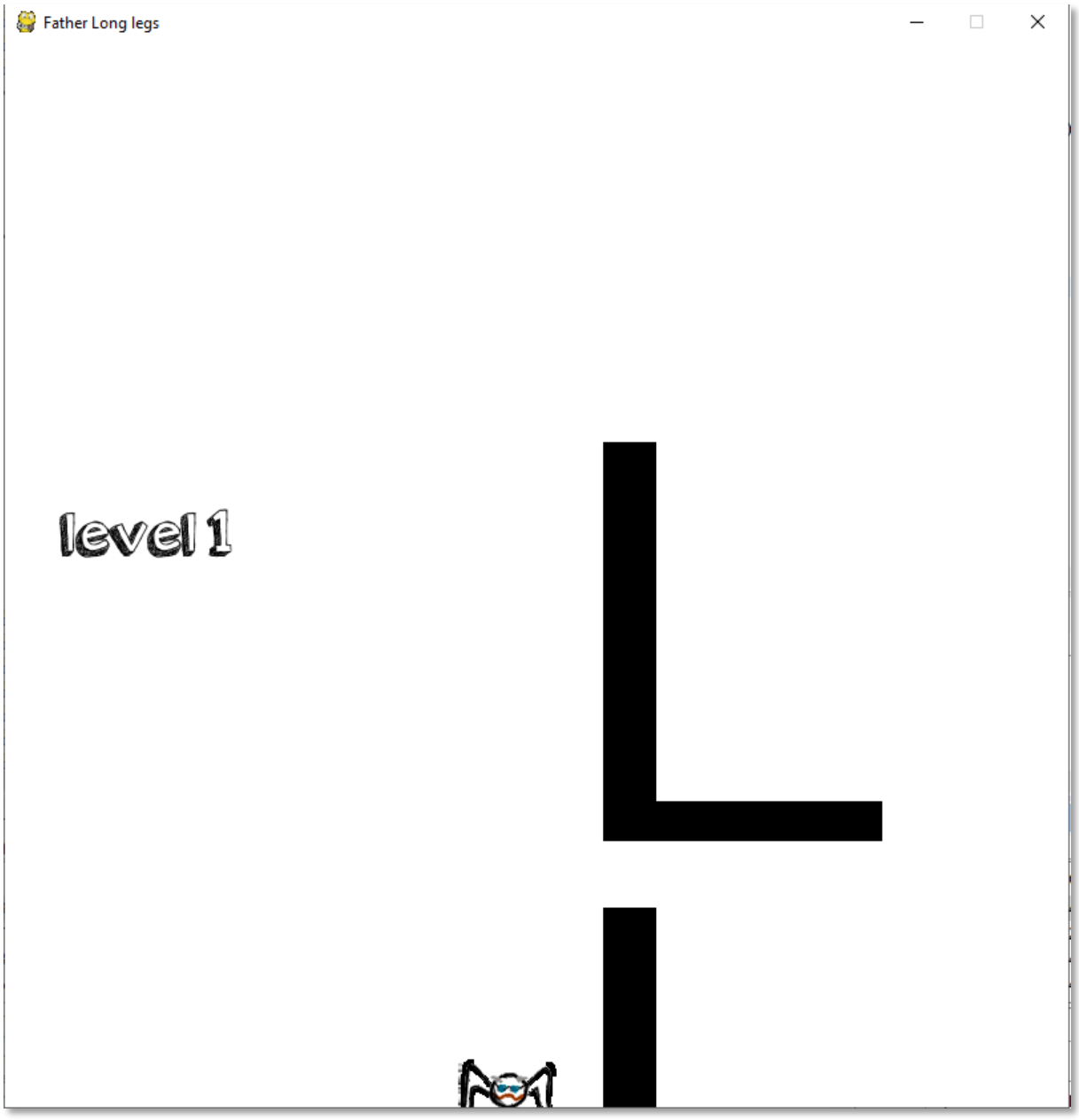
https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

https://fi.wikipedia.org/wiki/A*-algoritmi

https://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=python_09

16. **Liitteet**

Kuva 4 Aloitusvalikko



Kuva 5 Level1 pelaajahahmo pelissä