CS-A1121 Ohjelmoinnin peruskurssi Y2

Projektin dokumentaatio, Tornipuolustus

Juuso Pulkkinen, 431284

ENG, KJR, -14

15.4.2019

Sisällysluettelo

1. Yleiskuvaus

Tämän projektin tavoitteena oli tehdä klassinen tornipuolustuspeli, jossa tarkoituksena on estää vihollisen etenevien joukkojen pääsyä maaliin. Joukot etenevät pitkin ennalta määrättyä reittiä. Pelaajan tehtävä on rakentaa reitin varrelle torneja jotka ampuvat hyökkääviä joukkoja. Tuhotuista vihollisista pelaaja saa pisteitä sekä kultaa jonka avulla voi rakentaa lisää torneja. Peliä ei varsinaisesti voi päästä “läpi”, vaan pelaajan pitää kerätä mahdollisimman paljon pisteitä yhden kierroksen aikana. Pelin vaikeustaso nousee asteittain ajan kuluessa.

Peli on toteutettu Keskivaikea - vaikeusaste mielessä, mutta täyttää joitakin Vaativan - vaikeusasteen vaatimuksia (uusien kenttien, vihollistyyppien ja tornityyppien lisäys pidetty mielessä).

2. Käyttöohje

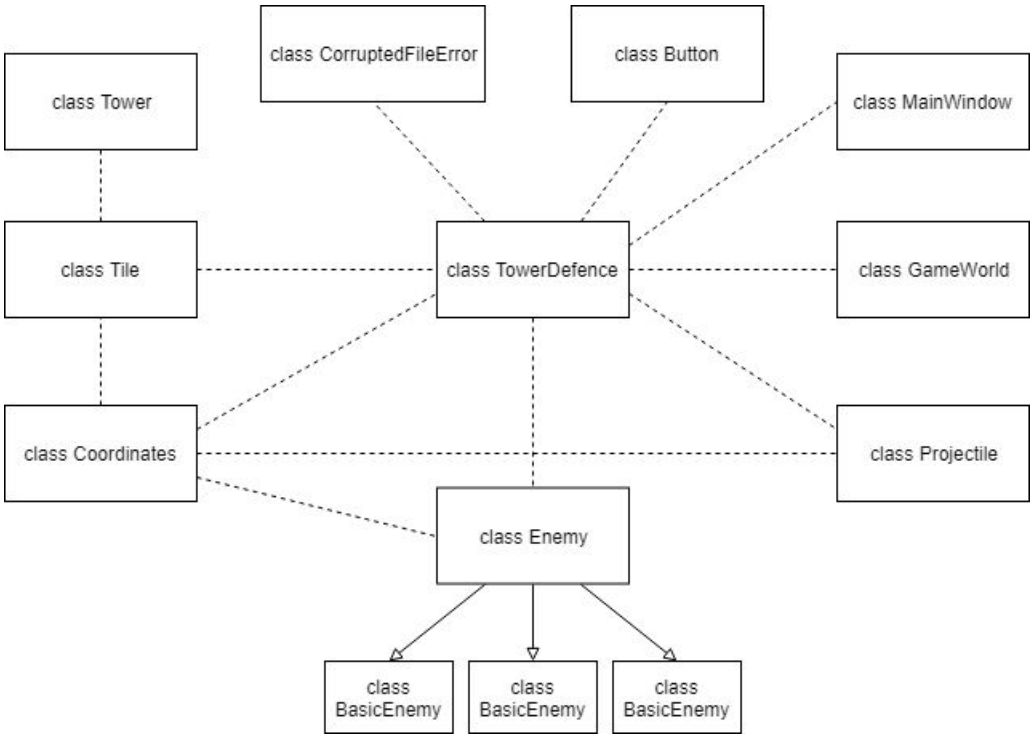
Ohjelma käynnistetään ajamalla main.py tiedosto Pythonilla. PyQt5 pitää olla asennettuna. Ohjelma on tornipuolustuspeli jota ohjataan hiirellä sekä näppäimistöllä.

Tarkempi käyttöohje löytyy gitin (https://version.aalto.fi/gitlab/pulkkij6/tornipuolustus) README-tiedostosta.

3. Ulkoiset kirjastot

PyQt5 on ainoa ulkoinen kirjasto jota projektissa on käytetty. Pelin grafiikat, ikkunat, käyttöliittymä sekä käyttäjän komentojen lukeminen on toteutettu PyQt:n avulla.

4. Ohjelman rakenne



Kuva 1: Ohjelman luokkarakenne

Ohjelman luokkarakenne on esitelty yllä. Peli pyörii TowerDefence luokan ympärillä joka päivittää peliä sekä käsittelee suurimman osan pelaajan antamista komennoista.

MainWindow luokkaa käytetään ohjelman ikkunan luomiseen sekä TowerDefence luokan alustukseen. MainWindow lisäksi käsittelee näppäimistöstä tulevat komennot, joita tosin on tällä hetkellä vain yksi: R näppäin aloittaa pelin alusta. Tarkoituksena oli alunperin laittaa näppäinkomentojen käsittely TowerDefence luokkaan jossa myös hiiren syöttöä luetaan, mutta jostain syystä PyQt:n keyPressEvent metodi ei tunnistanut näppäinkomentoja ollenkaan kun se oli toteutettuna TowerDefence luokassa.

TowerDefence luokka alustaa pelin valikon sekä pelimaailman itsessään kun peli aloitetaan. TowerDefence luokka hoitaa myös objektien päivityksen pelin aikana (esim. kutsuu metodeja objektien paikan päivitykselle sekä elämäpisteiden vähennykselle). TowerDefence luokka myös seuraa pelaajan piste- sekä kultatilannetta ja päivittää lukuja näytölle niiden muuttuessa.

Tärkeimpiä metodeja:

update_game(): kutsuu update_enemy, update_towers ja update_projectiles metodeja ja näin päivittää peliä. Update_game metodia kutsuu ajastin self.timer joka 20 millisekunti.

update_enemy(): päivittää ruudulla näkyvien vihollisten paikkaa sekä sisältää yksinkertaisen Polku - algoritmin jonka avulla joukot saadaan kävelemään lähdöstä maaliin.

update_towers(): päivittää pelaajan torneja, hoitaa tornien maalinhakemisen, kutsuu metodia vihollisen elämäpisteiden päivittämiselle sekä luo projektiilit tornin ampuessa. Poistaa kuolleet viholliset ruudulta.

update_projectiles(): päivittää tornien ampumien projektiilien positiota.

spawn_enemy(): luo uuden vihollisen pelikentälle tietyin väliajoin. Kutsutaan self.enemytimer ajastimella.

addGridItems(): luo pelimaailman lukemalla sen tekstitiedostosta sekä tallentaa sen GameWorld luokkaan 2-ulotteiseen listaan. Tekee myös visuaalisen esityksen pelimaailmasta.

drawUI(): piirtää pelin aikana näytettävän käyttöliittymän

mousePressEvent(): käsittelee pelaajan hiiren syötteitä. Kutsutaan aina kun klikkaus tapahtuu ruudulla.

Tile luokka kuvaa yhtä ruutua pelissä. Rakentaminen sekä kenttien luonti on pelissä tehty Tile pohjaisesti. Kentän koko on 20x15 ruutua. Tile luokassa luodaan myös tornit aina rakennuskäskyn tultua. Alustetaan tilen x ja y koordinaateilla.

Tärkeimpiä metodeja:

set_type(): määrittelee tilen tyypin. Alunperin kaikki tilet alustetaan “Basic” tyypiksi johon ei voi rakentaa, eikä viholliset voi kävellä sen päältä. Set_type():llä tile voidaan alustaa Tower - tileksi jolloin siihen voi rakentaa, Road - tileksi jolloin viholliset voivat kävellä siinä, tai Start tai End - tileksi jotka kuvaavat alku- ja loppupisteitä kentällä.

get_type(): palauttaa tilen tyypin (str).

set_tower(): luo uuden tornin sekä lisää tornin tileen rakennuskäskyn tultua, palauttaa bool arvon True tämän onnistuessa ja False jos rakentaminen epäonnistuu (metodi ei onnistu jos tilessä on jo torni).

get_tower(): palauttaa tilessä olevan tornin jos se sisältää sellaisen.

Tower luokka kuvaa yksittäistä tornia pelissä. Alustetaan tornin x ja y koordinaateilla.

Tärkeimpiä metodeja:

set_target(): asettaa kohteen jota torni ampuu. Ottaa kohteen id numeron parametrina (int).

delete_target(): poistaa tornin nykyisen kohteen.

get_target_status(): Palauttaa bool valuen joka kuvaa onko tornilla kohdetta vai ei.

shot(): Torni luokka sisältää ajastimen joka rajoittaa tornin ampumista. Aina kun torni ampuu shot() metodia kutsutaan joka käynnistää ajastimen joka estää tornia ampumasta sekuntiin.

Enemy luokka kuvaa yksittäistä vihollista ruudulla. Periytyy QGraphicsEllipseItemistä. Enemy luokka sisältää vihollisen eri vaadittavat parametrit (esim. Sijainti, elämäpisteet) sekä graafisen esityksen. Enemy luokasta periytyy vihollisten alaluokat **BasicEnemy, BigEnemy** sekä **FastEnemy** joita käytetään eri vihollistyyppien kuvaamiseen.

Tärkeimpiä metodeja:

set_pos(): päivittää vihollisen sijaintia ikkunassa PyQt:n setPos metodin avulla.

get_x_pos(): palauttaa vihollisen sijainnin x-akselilla (pikselikoordinaatti).

get_y_pos(): palauttaa vihollisen sijainnin y-akselilla (pikselikoordinaatti).

set_last_tile(): pitää yllä millä tilellä vihollinen oli viimeksi. Tämä tieto on tarpeellinen reitin löytämisessä (estää kävelyä taaksepäin).

set_dir(): pitää yllä suuntaa johon vihollinen on kävelemässä, tarvitaan reitin löytämisessä.

Projectile luokka kuvaa ammusta jonka torni ampuu. Periytyy QGraphicsEllipseItemistä.

Tärkeimpiä metodeja:

set_pos(): päivittää ammuksen sijaintia ikkunassa PyQt:n setPos metodin avulla.

get_x_pos(): palauttaa ammuksen sijainnin x-akselilla (pikselikoordinaatti).

get_y_pos(): palauttaa ammuksen sijainnin y-akselilla (pikselikoordinaatti).

set_target_x(): asettaa ammuksen kohteen x-akselilla (pikselikoordinaatti).

set_target_y(): asettaa ammuksen kohteen y-akselilla (pikselikoordinaatti).

get_target_x(): palauttaa ammuksen kohteen x-akselilla (pikselikoordinaatti).

get_target_y(): palauttaa ammuksen kohteen y-akselilla (pikselikoordinaatti).

Button luokkaa käytetään nappuloiden luomiseen pelissä. Tehty QGraphicsPolygonItemillä lähinnä tyylisyistä (nappulan muotoa ja väriä helpompi hallita).

Tärkeimpiä metodeja:

mousePressEvent(): tunnistaa jos hiirtä painetaan nappulan päällä, tummentaa nappulan väriä hieman jotta käyttäjä tietää että valinta on tehty.

GameWorld luokkaan tallenetaan pelin kaikki tilet 2-ulotteiseen listaan. Luokkaa käytetään jonkin tietyn tilen hakemiseen tietyistä koordinaateista.

Tärkeimpiä metodeja:

get_tile(): palauttaa tilen parametrina annetuista x ja y koordinaateista.

Coordinates luokka kuvaa koordinaatteja joita käytetään pelimaailman ruudukossa. Sisältää x sekä y arvon.

Tärkeimpiä metodeja:

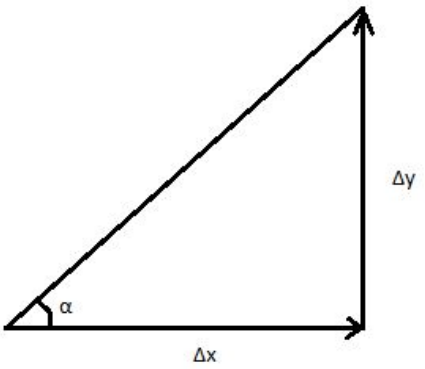
get_x(): palauttaa x-koordinaatin (int).

get_y(): palauttaa y-koordinaatin (int).

CorruptedFileError luokkaa käytetään poikkeuksien nappaamiseen pelikenttä - tiedostoa luettaessa.

5. Algoritmit

Tornien ammusten paikkaa päivitetään yksinkertaisen suunta/paikka-algoritmin avulla. Ammuksen lähtö- ja loppupaikka tiedetään, tästä voidaan muodostaa suorakulmainen kolmio vektoreiden avulla:



Δx ja Δy arvot saadaan vähentämällä loppu- ja alkupisteen saman akselin koordinaatit keskenään. Erotuksista otetaan itseisarvo:

$$\Delta x = abs\,(x_{end} - x_{start})$$
$$\Delta y = abs\,(y_{end} - y_{start})$$

Nyt voidaan laskea kulma α suorakulmaisesta kolmiosta tangentin avulla:

$$tan\,\alpha = \Delta y/\Delta x$$
$$\alpha = arctan(\,\Delta y/\Delta x\,)$$

Ammuksen kulkema matka x-akselilla yhden päivityksen aikana on:

$$x = cos\,\alpha \cdot v_{ammus}$$

Ja y-akselilla:

$$y = sin\,\alpha \cdot v_{ammus}$$

Joissa v_{ammus} on ammuksen nopeus. Jos ammus on päässyt x- tai y-akselin suhteen jo kohteeseensa, riittää että päivitetään vain toista akselia ammuksen vauhdin mukaisesti.

Tornien maalinhaussa käytetään pythagoraan lausetta tarkistamaan onko kohde tornin kantamalla:

$$\sqrt{\Delta x^2 + \Delta y^2} <= Tornin\ kantama$$

Vihollisen joukot löytävät maaliin asti yksinkertaisen reitinlöytämisen avulla. Vihollinen tietää aina edellisen ruudun jossa se oli, joka estää takaisinpäin kävelemisen. Päästyään jonkin ruudun keskipisteeseen, skannataan ympärillä olevat ruudut ja valitaan ensimmäinen tie-ruutu joka löydetään seuraavaksi kohteeksi, edellistä ruutua ei huomioida tässä tarkastuksessa jolloin takaisinpäin kävely estyy.

6. Tietorakenteet

Ohjelmassa käytetään muuttuvatilaisia rakenteita. Jokainen objekti sisältää omat tietonsa kuten niiden elämäpiste- ja nopeusarvot jotka ovat muutettavissa, sekä jokainen objekti tallennetaan listaan jonka avulla niitä päivitetään. Listaan lisätään ja poistetaan objekteja sitä mukaan kun niitä syntyy tai kun ne pitää poistaa. Listat sopivat tällaiseen yksinkertaiseen tiedonkäsittelyyn mainiosti koska oikeat kohteet on helppo löytää ja niitä on helppo käsitellä.

7. Tiedostot

Ohjelmassa käsitellään vain tekstitiedostoja jotka sisältävät pelikentän datan. Pelikenttää kuvataan tiedostossa seuraavaa formaattia noudattaen:

```
XXSXXXXXXXXXXXXXXXX
XX1XXX11111XXXXXXXXX
XX1XXX1XXX1XXXXXXXXX
XX1XXX1XXX1XXXXXXXXX
XX11111XXX1XXXXXXXXX
XXXXXXXXXX1XXXXXXXXX
XXXXXXXXXX1XXXXXXXXX
XXXXXXXXXX1XXXXXXXXX
XXX11111111XXXXXXXXX
XXX1XXXXXXXXXX1111E
XXX1XXXXXXXXXX1XXXX
XXX1XXXXXXXXXX1XXXX
XXX1111111111111XXXX
XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
```

Yksi kirjain/numero kuvaa yhtä ruutua pelikentällä. Kirjaimia tiedostossa on siis 20x15. Ohjelma tunnistaa seuraavat kirjaimet/numerot:

X kuvaa paikkaa johon tornin voi rakentaa
1 kuvaa tietä jolla vihollisen joukot kävelevät
S kuvaa aloituspistettä
E kuvaa loppupistettä
0 kuvaa ruutua jossa vihollinen ei pysty kävelemään, eikä pelaaja pysty rakentamaan siihen torneja.

Ohjelma osaa tunnistaa korruptoituneet kenttätiedostot jos niistä puuttuu jotain tai jos ne sisältävät ylimääräisiä rivejä/kolumneja. Ohjelma ei myöskään salli useita alku- ja loppupisteitä.

8. Testaus

Ohjelmaa testattiin eniten manuaalisesti. Esimerkiksi reitinlöytämistä luodessa katsottiin miten viholliset käyttäytyvät käännöskohdissa ja muutoksia tehtiin bugeja löydettäessä. Ohjelma sisältää paljon syötteenkäsittelyä jota oli helppo testata kokeilemalla vain mitä tiettyä asiaa tehdessä tapahtuu.

Kentän lukemiselle tiedostosta on tehty yksikkötestaukset eri tapauksille. Yksikkötesteissä testataan tunnistaako ohjelma jos tiedostosta:

- Löytyy tuntematon merkki
- Löytyy liikaa merkkejä yhdellä rivillä
- Löytyy liikaa rivejä
- Puuttuu merkkejä riviltä, tai puuttuu kokonaisia rivejä
- Löytyy usea alkupiste
- Löytyy usea loppupiste

Lisäksi ohjelma tunnistaa jos tiedostoa ei löydy ollenkaan. Tiedostot joita käytettiin testauksiin löytyvät src/test/ kansiosta.

9. Ohjelman tunnetut puutteet ja viat

Ohjelmassa on ainakin yksi bugi jonka seurauksena tornin ampuma ammus jää vain leijumaan tornin päälle eikä koskaan päädy määrättyyn kohteeseen. Tätä on koitettu korjata erilaisilla checkeillä (esim. tarkistetaan jos ammus on jo x-akselilla kohteessa päivitetään vain y-akselia) mutta mikään ei tunnu auttavan. Yksi mahdollinen selitys voi olla että Pythonin math kirjaston atan2 funktion tarkkuus ei riitä tarpeeksi pienillä luvuilla kulman laskemiseen, ja tämä jotenkin hajoittaa paikan päivityksen. Tätä voisi koittaa korjata käyttämällä jotain toista matematiikkakirjastoa kuten numpya.

Tornien kohteenetsintää voisi miettiä vähän paremmaksi. Tällä hetkellä torni skannaa listaa joka sisältää vihollisia ja etsii sieltä ensimmäisen joka on sen kantamalla ja asettaa sen itsensä kohteeksi. Tämä toimisi hyvin jos kaikkien vihollisten nopeus olisi sama. Nyt jos nopeampi vihollinen ohittaa hitaampia ja tornilla on monta kohdetta joista valita, se ei priorisoi tätä vihollista koska sen indeksi listassa on suurempi. Tämän korjaaminen vaatisi jonkin metodin joka selvittää mikä vihollisista on edennyt pisimmälle reittiä. Toisaalta tämä voi olla monimutkaista toteuttaa siten että se toimisi jokaisessa kentässä.

Peli hyväksyy kenttätiedostot joissa on epämääräisiä tie - rakenteita (esim. Monta tie tileä vierekkäin, reitti ei johda maaliin asti). Tiedoston lukemiseen tarvittaisiin jonkinlainen esitestaus että reitti maaliin asti löytyy jotakin kautta.

Pelissä ei ole kentän valinta - ruutua, vaan kentän nimi pitää manuaalisesti kirjoittaa addGridItems metodiin TowerDefence luokassa. Tämä ei ole kovin käyttäjäystävällistä jos joku haluaa tehdä omia kenttiä.

10. 3 parasta ja 3 heikointa kohtaa

3 parasta kohtaa

- Kentän lukeminen tiedostosta addGridItems() metodissa TowerDefence luokassa. Ohjelma tunnistaa korruptoituneet tiedostot ja heittää niistä poikkeuksen.
- Tornien päivitys update_towers() metodissa samassa luokassa. Yleisesti olen aika tyytyväinen siihen miten tornit ampuvat ja löytävät kohteensa.
- mousePressEvent() metodi samassa luokassa. Hiiren klikkausten lukeminen on kyllä aikalailla kovakoodattu tiettyjen pikselikoordinaattien sisään, mutta metodi ajaa asiansa hyvin ja tunnistaa mitä pelaaja on tekemässä.

3 heikointa kohtaa

- update_enemy() metodin loppupää TowerDefence luokassa. Jouduin käyttämään joka vihollisessa laskuria joka varmistaa että kävely tapahtuu aina tien keskellä.
- spawn_enemy() metodi samassa luokassa on hyvin yksinkertaisesti toteutettu. Tässä olisi voinut käyttää jotain satunnaisuutta hyväksi tai vaihtoehtoisesti suunnitella tietyt osajoukot valmiiksi (esim. tehdään hetki vain isoja joukkoja).
- Status viestien toimiminen TowerDefence luokassa. Status viestit toimivat QGraphicsTextItemillä johon viesti aina päivitetään haluttuun, ja tyhjennetään päivittämällä siihen tyhjä merkkijono. Tämän olisi voinut toteuttaa jotenkin fiksummin.

11. Poikkeamat suunnitelmasta

Projektin aikana suunniteltu luokkarakenne meni lähes kokonaan uusiksi. En ollut täysin käsittänyt projektin suunnitteluvaiheen aikana miten Qt sovellukset toimivat joten oletin että main funktiossa peliä pyörittävä loop - rakenne olisi toiminut. Qt kuitenkin itse pyörittää jonkinlaista sisäistä loop - rakennetta, joten peli oli helpointa rakentaa sen ympärille.

Ajankäyttöarvio osui suunnitelmassa melko oikeaan. Projektissa kului vähän vähemmän aikaa mitä olin suunnitellut (70h, suunnitelman arvio oli +80h).

12. Toteutunut työjärjestys ja aikataulu

8.3. - 12.3.

Pelimoottorin koodaus, PyQt testailut

12.3. - 14.3.

Pelimaailman luonti, karttojen rakennus

14.3. - 20.3.

Pelimekaniikkojen suunnittelu ja lisäys (rakentaminen), samalla grafiikoita

28.3. - 30.3.

Pelimekaniikkojen hiominen, pelimoottorin viimeistely, bugikorjauksia, grafiikoita

30.3. - 4.4.

Pelimekaniikkojen viimeistely (vihollisten joukot, ampuminen, rakentaminen, resurssit), grafiikoita

12.4. - 15.4.

Testailua, bugien korjausta

Työjärjestys pysyi suunnitelmassa hyvin. Pelin perusrakenne tuli tosin valmiiksi odotettua aikaisemmin joten grafiikkojen ja eri pelimekaniikkojen koodaamisen pystyi aloittamaan nopeammin.

13. Arvio lopputuloksesta

Yleisesti olen hyvin tyytyväinen projektin lopputulokseen. Projekti jäi sellaiseen vaiheeseen että se toimii oikeasti pelinä ja on helposti laajennettavissa jos joskus palaan tämän projektin pariin. Yksi ainoita asioita mitä olisin vielä halunnut lisätä olisi ollut ainakin toinen tornityyppi, mutta en keksinyt tähän mitään helppoa ratkaisua joka olisi PyQt:n rajallisilla grafiikoilla järkevä toteuttaa niin keskityin enemmän bugien löytämiseen ja pelin tasapainottamiseen.

Isoin ongelma oli ehkä reitinlöytämisen koodaus. Olen tyytyväinen että sain sen muotoon joka toimii kaikissa kentissä kunhan reitti alusta loppuun on olemassa, mutta en ole kovin varma kuinka hyvää koodia lopputuloksesta tuli. Tähän olisi voinut ehkä käyttää jotain valmista algoritmia, toisaalta ongelma oli kiva ratkaista itse.

Ratkaisumenetelmät, luokkajaot ja tietorakenteet joita projektissa on käytetty ajavat hyvin asiansa tässä yksinkertaisessa pelissä.

Arvosanaksi projektille antaisin 4.

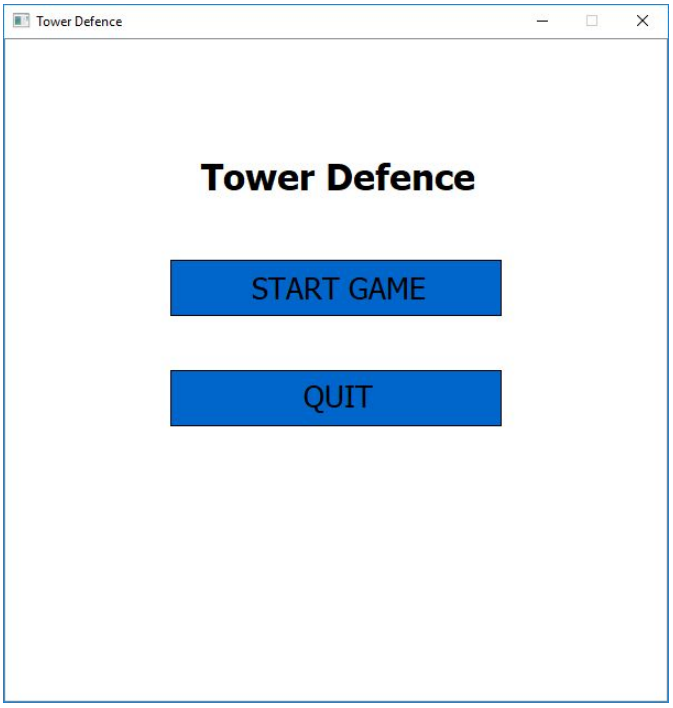
14. Viitteet

Projektissa käytettyjä lähteitä:

[1] PyQt5:n dokumentaatio: http://pyqt.sourceforge.net/Docs/PyQt5/
[2] Qt5:n dokumentaatio: https://doc.qt.io/qt-5/
[3] Python Standard Library: https://docs.python.org/3/library/
[4] Stackoverflow: https://stackoverflow.com/

15. Liitteet

Liite 1: kuva valikosta



Liite 2: kuva peli-ikkunasta