Aalto-yliopisto
Sähkötekniikan korkeakoulu
Elektroniikka ja Sähkötekniikka

Projektidokumentaatio

Tornipuolustuspeli

Joel Lavikainen
479848
3. vuosikurssi
5.5.2017

Sisältö

1 Yleiskuvaus

Projektin aiheena oli tornipuolustuspeli, jossa joukko vihollisia pyrkii ennaltamäärättyä reittiä pitkin pääsemään maaliin. Pelaajan tehtävänä on tuhota viholliset ennen maaliin pääsemistä sijoittamalla strategisesti torneja, jotka ampuvat, hidastavat tai muuten heikentävät vihollisia.

Viholliset syntyvät aalloissa ja aallot sisältävät eri tyyppisiä vihollisia, joiden tuhoamiseen vaaditaan tietynlaisia torneja. Pelin edetessä aallot vaikeutuvat ja vihollisten määrä kasvaa. Vihollisten tuhoamisesta saadaan rahaa, jolla voidaan hankkia uusia torneja tai päivityksiä jo olemassa oleviin torneihin.

Pelissä on erilaisia kenttiä, joissa vihollisten kulkema reitti ja aaltojen vihollismäärät vaihtelevat. Kentät ladataan pelille ominaisista kenttätiedostoista. Kenttätiedostot eivät ole ihmisille luettavassa formaatissa. Kentät ovat ruudukoita, jolloin tornien sijoittelu on pelaajalle mahdollisimman selkeätä.

Pelaajan tukikohdalla eli maalilla on ennaltamäärätty määrä elämiä, jotka vähenevät kun vihollisia pääsee tornien ohi maaliin asti. Pelin häviää, jos elämät loppuvat kesken pelin. Pelaaja voi voittaa pelin selviämällä kaikista vihollisaalloista elossa.

Pelaajien sijoittamien tornien arvoja ja päivityksiä voidaan vapaasti muokata ihmiselle luettavassa xml-formaatissa. Tämä mahdollistaa pelitasapainon kannalta sopivien arvojen helpon iteroinnin. Peliin on toteutettu kaikki vaativamman asteen vaatimukset, eli siinä on graafinen käyttöliittymä ja suurin osa asioista on konfiguroitavissa erillisten tiedostojen avulla.

2 Käyttöohje

Peli käynnistetään suorittamalla main.py tiedosto, joka avaa suoraan oletuskentän graafiseen käyttöliittymään. Tämän jälkeen pelaaja voi sijoittaa alkuvaroilla muutaman tornin ja aloittaa ensimmäisen vihollisaallon painamalla Next wave-painiketta. Tällöin aloitusruutuun alkaa syntymään aaltoon konfiguroitu määrä eri tyyppisiä vihollisia pienen intervallin jälkeen. Käyttöliittymä toimiin täysin hiirellä klikkaamalla. Oikealla puolella peli-ikkunaa on tietopalkki, jossa näkyy valittujen yksikköjen tiedot, sijoitettavat tornit ja yleistä tietoa pelin kulusta. Pelin kulusta on esillä pelaajan elämät, aaltojen tilanne ja pelaajan käytössä olevat rahavarat.

Uusia torneja voidaan asettaa valitsemalla aktiiviseksi käyttöliittymän oikean puolen tietopalkista tornin tyyppi ja klikkaamalla pelikentälle haluttua tyhjää ruutua johon torni voidaan rakentaa. Pelaajan sijoittamat tornit hyökkäävät automaattisesti niiden kantaman sisällä oleviin vihollisiin. Pelaaja voi valita jo sijoitettuja torneja, jolloin niiden päivitystilanne näkyy oikealla olevassa tietopalkissa. Torneille voidaan päivittää kolmea asiaa: niiden kantamaa, hyökkäysnopeutta ja vauriota. Päivitystasojen määrää, arvoa ja hintaa voidaan säädellä suoraan konfiguraatiotiedostosta jokaisen tornin kohdalta erikseen. Liitteessä on esitetty pelin käyttöliittymä kuvassa 2.

3 Ohjelman rakenne

Peliin on toteutettu graafinen käyttöliittymä käyttämällä PyQT5-kirjastoa. Pelilogiikan ja graafinen käyttöliittymä on pidetty erillään toisistaan, jos grafiikan piirto halutaan myöhemmin toteuttaa hyödyntäen jotain muuta kirjastoa. Tämän takia pelin rakenne voidaan jakaa karkeasti kahteen osa-alueeseen: pelilogiikkaan ja graafiseen käyttöliittymään. Lisäksi on tiedostojen lukuun tehdyt apuluokat.

3.1 Pelilogiikka

Pelilogiikan kannalta tärkeitä luokkia ovat:

- Yksikkö (Unit)
- Torni (Tower)
- Vihollinen (Enemy)
- Pelimaailma (GameWorld)
- Ruutu (Square)
- Synnytin (Spawner)

3.1.1 Pelimaailma

Pelimaailma toimii yleiskuvauksena pelikentästä, ja se hallitsee pelin kulkua, vihollisia ja torneja. Pelimaailma pitää kirjaa kentän ruuduista, kuljettavasta reitistä, elossa olevista vihollisista ja torneista. Kentän tiedot luetaan pelin alussa kenttätiedostosta ja pelimaailma olioon alustetaan listaan vastaavat ruutu oliot, kuin kenttätiedostossa määritetään. Kenttätiedostosta luetaan myös vihollisaaltojen tiedot, jotka alustetaan pelimaailma olioon listaan.

Keskeisiä metodeja pelimaailmalle ovat:

- add_tower(), käytetään uusien tornien kenttään lisäämiseksi
- add_enemy(), käytetään vihollisten lisäämiseen pelikentälle
- remove_dead_enemies(), metodi tarkistaa mitkä viholliset ovat kuolleita ja poistaa nämä
- next_wave(), käynnistää seuraavan kierroksen kentässä ja luo synnytin olion reitin aloitusruutuun

3.1.2 Yksikkö

Yksikkö toimii pohjaluokkana pelikenttään sijoitettaville yksiköille eli torneille ja vihollisille. Nämä molemmat perivät yksikköluokalta tärkeimmät ominaisuudet, kuten sijainnin ja nykyisen pelimaailman. Yksikkö luokalla vältetään saman koodiin toistamista useaan kertaan.

3.1.3 Torni

Tornipuolustuspelin tärkein olio eli torni sisältää tietoa tornin ominaisuuksista ja sijainnista. Torni on myös tietoinen pelimaailmasta, johon se kuuluu. Myös tornin mahdolliset päivitykset ja nykyinen päivitystaso säilytetään torni oliossa. Eri tornit ja niitä vastaavat tiedot luetaan erillisestä konfiguraatiotiedostosta, joka on xml muodossa. Tämä mahdollistaa uusien tornien luonnin ja tasapainottamisen lennosta ilman, että arvoja oltaisiin kovakoodattu. Tornin tyypin kuvausta varten on olemassa lueteltu tyyppi TowerType.

Keskeisiä metodeja tornille ovat:

- find_enemies(), etsii tornin hyökkäyskantamalla olevat viholliset
- attack(), tornin hyökkäysmetodi, joka valitsee kantamalla olevista vihollisista yhden satunnaisesti
- upgrade(), päivittää halutun päivityksen torniin
- update(), tornin ampumisnopeutta säätelevä funktio, joka toimii yleisen logiikkakellon mukaan

3.1.4 Vihollinen

Pelissä kentällä olevaa reittiä pitkin kulkevat viholliset toteutetaan omana olionaan. Kyseinen olio sisältää tietoa vihollisen elämistä, nopeudesta, paikasta ja rahapalkkiosta. Vastaavasti kuin torneilla viholliset luetaan omasta konfiguraatiotiedostostaan, joka on myös xml formaatissa. Vihollisen eri tyyppien kuvaamiseen on käytössä lueteltu tyyppi EnemyType.

Keskeisiä metodeja viholliselle ovat:

- move(), vihollinen liikkuu seuraavaan ruutuun ennaltamäärätyllä reitillä
- damage(), kutsutaan, kun torni tekee vauriota viholliseen. Muuttaa myös vihollisen kuolleeksi, jos elämät menevät nollaan
- update(), vihollisen liikkumisnopeutta säätelevä funktio, joka toimii yleisen logiikkakellon mukaan

3.1.5 Ruutu

Ruutu on kuvaus pelikentän ruudukon yhdestä ruudusta. Ruudun eri tyyppejä kuvataan luetellulla tyypillä SquareType. Ruudun tyyppejä ovat reitti-, alku- ja maaliruudut sekä torniruudut, joihin voidaan sijoittaa torneja. Ruutuihin sisällytetään ruudun päällä oleva olio eli joko torni tai viholliset. Viholliset syntyvät alkuruutuun ja tuhoutuvat maaliruudussa vähentäen pelaajan pisteitä.

Keskeisiä metodeja ruudulle ovat:

- set_tower(), käytetään tornin asettamiseen ruutuun
- set_enemy(), vastaava kuin set_tower(), mutta viholliselle
- remove_unit(), poistaa ruudusta halutun yksikön

3.1.6 Synnytin

Synnytin on apuluokka, joka huolehtii kierroksen vihollisten synnyttämisestä. Synnytin saa tehtäväkseen aina yhden kierroksen synnyttämisen. Se arpoo jäljellä olevista vihollisista seuraavan ja lisää sen pelikentän aloitusruutuun.
Keskeisiä metodeja synnyttimelle ovat:

- next_enemy(), palauttaa kierroksella jäljellä olevista vihollisista yhden satunnaisesti
- spawn_enemy(), synnyttää arvotun vihollisen pelimaailmaan
- update(), vihollisten syntymisnopeutta säätelevä funktio, joka toimii yleisen logiikkakellon mukaan

3.2 Graafinen käyttöliittymä

Graafisen käyttöliittymän kannalta tärkeitä luokkia ovat:

- GUI
- QScene
- Ammus (Projectile)
- UnitGraphicsItem
- TowerGraphicsItem
- EnemyGraphicsItem

3.2.1 GUI

Erillään olevaa graafista käyttöliittymää kuvataan hallinnoivalla oliolla GUI, joka pitää sisällään tiedon kentän, ammusten, vihollisten ja tornien piirrosolioista. GUI oliossa on myös logiikan ja graafiikan päivittämistä hallinoivat ajastimet. Pelilogiikka liitetään graafiseen käyttöliittymään antamalla GUI oliolle oma pelimaailma olio. GUI olio kutsuu pelimaailman pelilogiikalle päivityksiä logiikkakellon mukaan.

Keskeisiä metodeja GUI:lle ovat:

- add_map_grid_items(), piirtää ruutujen graafisen esityksen

- add_enemy_graphics_items(), lisää graafisen esityksen vihollisille, joilta se puuttuu
- add_tower_graphics_items(), vastaava kuin add_enemy_graphics_items()
- create_projectile(), piirtää tornien hyökkäystä esittävät ammukset
- init_window(), alustaa graafisen käyttöliittymän pääikkunan ja pelin piirtoalueen
- init_info_layout(), alustaa käyttöliittymään painikkeet ja muut informaatiiviset tekstit oikeaan tietoreunukseen
- update_enemies(), päivittää vihollisten uuden sijainnin grafiikkaan
- update_spawner(), päivittää synnyttäjän logiikkaa
- update_towers(), päivittää tornien logiikkaa

3.2.2 QScene

QScene on QGraphicsScenen perivä pieni apuluokka, jota käytetään Scenen painallusfunktion uudelleenmäärittelyä varten. QScene on tietoinen GUI oliosta, koska se muokkaa siihen sisältyviä informatiivisiä tekstejä ja painikkeita. QScenellä on ainoastaa metodi mousePressEvent(), jossa on yksinkertainen tilakone määrittämään käyttäytymistä eri tilanteissa, kun pelikenttää napsautetaan hiirellä. Näitä painalluksia käytetään esimerkiksi tornien valintaa ja sijoittamista varten.

3.2.3 Ammus

Ammus on pelkästään graafinen efekti esittämään tornien hyökkäämistä vihollisiin. Tornien käyttämät ammukset luetaan tornien xml-tiedostosta. Ammusten tyyppien esittämistä varten on lueteltu tyyppi ProjectileType. Perii Qt:n QGraphicsPixmapItem luokan kuvien esittämistä varten.

Keskeisiä metodeja ammukselle ovat:

- set_graphics(), riippuen ammuksen tyypistä asettaa joko kuvan ammukselle tai piirtää sen QPainter-luokkaa hyödyntäen
- update(), ammusten graafiikan liikkumisen päivitysfunktio. Interpoloi lineaarisesti ammuksen liikkeen

3.2.4 UnitGraphicsItem

Yleisluokka grafiikka olioille. Sisältää peligrafiikkaa. Perii Qt:n QGraphicsPixmapItem luokan, jolloin sille on määritelty valmiiksi paljon perusominaisuuksia, kuten kuvien esittäminen.

3.2.5 TowerGraphicsItem

Tornien grafiikkaa varten on TowerGraphicsItem olio, johon liitetään aina sitä vastaava torni. TowerGraphicsItem perii UnitGraphicsItem luokan TowerGraphicsItem piirtää tornin tyypille määritellyn grafiikan yleisestä tilemap kuvasta oikealle paikalleen.

3.2.6 EnemyGraphicsItem

Vastaavasti vihollisten grafiikka esitetään EnemyGraphicsItem oliolla, johon liitetään vastaava vihollinen. Perii myöskin UnitGraphicsItem luokan. Lataa vihollistyypille vastaavan grafiikan yleisestä tilemap kuvasta. Tärkeänä metodina on update(), jolla päivitetään grafiikan sijainti vastaamaan liitetyn vihollisen koordinaatteja ja interpoloidaan graafisesti vihollisen liike.

3.3 Tiedostojen luku

Tiedostojen luvun kannalta tärkeitä luokkia ovat:

- Kentänlukija (MapReader)
- Konfiguraationlukija (ConfigReader)

3.3.1 Kentänlukija

Kentänlukija luokka vastaa kenttätiedostojen lukemisesta pelille sopivaan muotoon. Kentänlukija lukee xml-tiedostoista peliruutujen tyypin ja kierrokset. Se etsii lisäksi ruudukosta vihollisten kulkeman reitin. Näiden tietojen perusteella se alustaa pelimaailma olion.
Keskeisiä metodeja kentänlukijalle ovat:

- parse_waves(), parsii kenttätiedostosta kierrostiedot pelin hyödyntämään muotoon
- parse_route(), etsii luetusta pelikentästä vihollisten kulkeman reitin
- parse_map(), lukee peliruutujen tyypin ja alustaa ruudukon pelimaailmaan

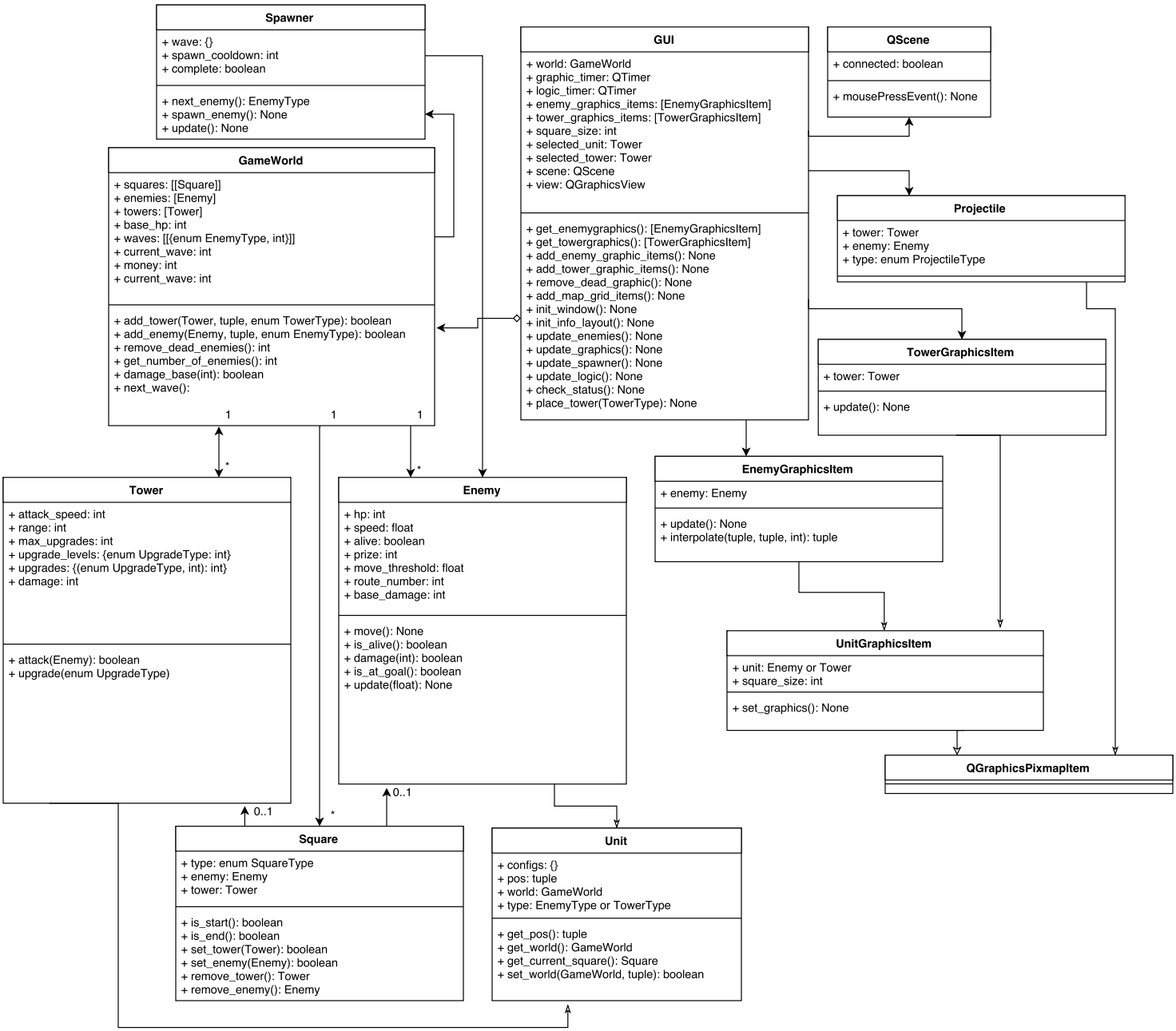
3.3.2 Konfiguraationlukija

Konfiguraationlukija luokka vastaa konfiguraatiotiedostojen lukemisesta pelille sopivaan muotoon. Konfiguraationlukija lukee xml-tiedostoista vihollisten ja tornien ominaisuudet. Se parsii lisäksi torneihin liittyvät päivitykset.
Keskeisiä metodeja konfiguraationlukijalle ovat:

- parse_enemies(), parsii vihollisten ominaisuudet tiedostosta
- parse_upgrades(), parsii tornien päivitystiedot sopivaan muotoon
- parse_towers(), parsii tornien ominaisuudet tiedostosta

- parse_config(), kokoaa edellisten funktioiden saaman tietoon yhdeksi konfiguraatio sanakirjaksi

3.4 UML-kaavio



Kuva 1: UML-kaavio luokkien välisistä suhteista

4 Algoritmit

Vihollisten liikkuminen alkuruudusta maaliruutuun on toteutettu hyvin yksinkertaisesta johtuen ennaltamääritetystä reitistä. Viholliset tietävät entuudestaan koko reitin eli mihin ruutuun seuraavaksi liikutaan. Vihollinen pitää kirjaa siitä missä ruudussa on tällä hetkellä ja move() metodia kutsuttaessa vihollinen siirtyy järjestyksessä seuraavaan ruutuun. Logiikkapäivitysten yhteydessä vihollista siirretään murto-osan verran eteenpäin ruudussa. Kun tämä kynnysarvo saavuttaa ykkösen arvon, voidaan vihollinen siirtää seuraavaan peliruutuun. Vihollisten eri nopeudet toteutetaan kertomalla tätä murto-osaa viholliselle asetetulla nopeudella. Vihollisten sijainnit oltaisiin voitu alunperinkin toteuttaa murtolukuina, mutta nykyinen toteutus helpottaa esimerkiksi hyökkäysalgoritmin toteutusta.

Tornit tietävät tarkalleen kaikkien vihollisten sijainnin, minkä takia ne voivat tarkastella toistuvasti ovatko ne ampumaetäisyydellä vihollisesta. Tornien ampumaetäisyys on ruudukossa määritetty etäisyys jokaiseen väli- ja pääilmansuuntaan. Torni etsii kaikki kantamalla olevat ruudut muodostamalla kantaman kokoisen neliön, jonka sisällä olevat ruudut otetaan mukaan tarkasteluun. Viholliset voivat kuitenkin vain sijaita yhden ruudun levyisellä ennaltamäärätyllä reitillä, joten algoritmin tarvitsee tarkastella vaan etäisyydellä olevia reitin ruutuja. Torni voi löytyy kantamaltaan useita vihollisia, joihin se ylettää. Tämän takia torni arpoo löydetyistä vihollisista hyökkäyskohteen. Torni olisi voinut myös valita hyökkättäväksi kohteeksi esimerkiksi vähiten elämiä omaavan vihollisen. Nykyinen toteutus kuitenkin luo miellyttävää vaihtelua hyökkäykseen eikä se suosi mitään tiettyä ruutua.

Tornien hyökkäyksissä muodostamat ammukset liikkuvat hyökkättyyn kohteeseen samankaltaisella logiikalla kuin viholliset liikkuvat. Jälleen lasketaan murto-osa tornin ja vihollisen sijainnin välillä. Tätä murto-osaa kasvatetaan nollasta ykköseen, kunnes kohde saavutetaan. Ammukset lentävät sitä kovempaa mitä kauempana ne ovat. Vauhti, jolla ammuksen täytyy lentää lasketaan oheisen kaavan mukaan:

$$v_{ammus} = \frac{\sqrt{(vihollinen_x - torni_x)^2 + (vihollinen_y - torni_y)^2}}{t_{elinaika}} \quad (1)$$

Ammuksena käytetään myös lasersädettä, joka piirretään suorana viivana tornin keskikohdasta vihollisen keskikohtaan.

Kenttätiedostoa luettaessa kentästä etsitään vihollisten kulkema reitti yksinkertaisella algoritmilla. Ensiksi kentästä etsitään aloitusruutu, jonka jälkeen tarkastellaan tämän pääilmansuunnissa olevia naapureita. Vain yksi näistä on tyypiltään reittiruutu ja se lisätään listaan. Edellisestä ruudusta pidetään kirjaa, jotta algoritmi ei jää jumiin kahden reittiruudun väliin. Löydetyt reittiruudut lisätään listaan järjestyksessä, jolloin liikkumiskoordinaatit ovat järjestyksessä. Reitin hakuun olisi voinut esimerkiksi käyttää Breadth First Searchia, mutta se ei ollut tarpeellista, koska pelikentän reitti oli rajoitettu yksiväyläiseksi.

5 Tietorakenteet

Pelimaailmassa vihollinen ja torni oliot säilytetään listassa, koska niiden määrä muuttuu dynaamisesti pelin edetessä. Eri tyyppien ilmaisemiseen käytetään lueteltua tyyppiä (enum), koska se on kuvaavampi tapa ilmaista kuin pelkät numerot. Lueteltu tyyppi toimii myös mukavasti sanakirjoissa avaimena. Samoja lueteltuja tyyppejä käytetään myös pelin kenttä- ja konfiguraatiotiedostoissa, jolloin ne voidaan helposti konvertoida sanakirjojen avaimiksi.

Kaikki vihollisten ja tornien sijainnit säilytetään muuttumattomina tupleina. Tämän sijasta olisi ehkä ollut parempi muodostaa koordinaateille oma apuluokka, jolle olisi määritetty esimerkiksi elementtikohtaiset plus- ja miinuslaskut.

Tornien päivitykset säilytetään sanakirjassa, jonka avaimena on lueteltun tyypin UpgradeType ja kokonaisluvun muodostama tuple. Sanakirjan arvona on päivitystä vastaava ominaisuuden nouseminen kokonaislukuna. Tornien päivitystilanteesta pidetään kirjaa upgrade_levels sanakirjalla, jonka avaimena on päivityksen tyyppi UpgradeType. Kun halutaan päivittää tornia, voidaan ensin kysyä päivitystä vastaava kokonaisluku tältä sanakirjalta, jonka jälkeen oikeaa tasoa vastaava päivitys saadaan toisesta sanakirjasta. Sanakirja on hyvä valinta, koska päivitykset eivät muutu kesken pelin vaan ne alustetaan pelin alussa kerran.

Pelin kierrokset säilytetään listassa, joka sisältää sanakirjoja, joiden avaimina on vihollistyyppi EnemyType ja arvoina vihollisten määrä kokonaislukuna. Esimerkiksi [{enemy_a:3, enemy_b:5}, {enemy_a:10, enemy_b:8}], jolloin ensimmäisellä kierroksella enemy_a olisi 3 kpl ja enemy_b 5 kpl. Vastaavat määrät seuraavalla kierroksella olisivat 10 ja 8 kpl.

Pelin konfiguraatiot säilytetään yhdessä moniulotteisessa sanakirjassa, jonka avaimina toimivat halutut ominaisuudet. Esimerkiksi vihollisten ominaisuudet säilötään vihollistyyppiä avaimena käyttäen. Konfiguraatioiden sanakirja on todella helppo muodostaa luetuista xml-tiedostoista käyttämällä xml-tageja avaimina.

6 Tiedostot

Pelissä on käytössä kahdenlaisia tiedostoja: kenttä- ja konfiguraatiotiedostoja. Nämä molemmat on toteutettu xml-formaatissa. Molemmissa tieto on esitetty käyttäjälle luettavassa muodossa ja niiden sisältämiä arvoja voikin muokata käsin. Kenttätiedosto sisältää tiedon kentän leveydestä ja korkeudesta, kenttäruudukon ruutujen arvot ja kierrostiedot. Kenttätiedostoissa kentän muodostaminen kuitenkin käsin on kovin työlästä, jonka takia on toteutettu muunnosskripti pelkästään numeroilla esitetystä ruudukosta. Kierrostiedot voidaan kuitenkin kirjoittaa helposti käsin.

Konfiguraatiotiedostossa on osiot erikseen tornien ja vihollisten tiedoille. Tornio-osiosta löytyy tornityypillä nimettynä kyseisen tornin ominaisuudet ja päivitykset. Osiossa on myös kerrottu mistä kohtaa tilemappia kyseisen tornin graafiikka löytyy. Lisäksi ammuksen tyyppi ja sen grafiikan sijainnit löytyvät

tiedostosta. Liitteestä eli projektin gitistä löytyy molemmista tiedostoista esimerkit.

7 Testaus

Pelilogiikka testattiin sijoittamalla vihollisia ja torneja kiellettyihin paikkoihin ja tulostamalla komentoriville pelikentän tilanne merkkeinä. Myös vihollisten liikkumisalgoritmia testattiin alustamalla yksi vihollinen alkuruutuun ja kutsumalla vihollisen liikkumiskäskyä toistuvasti. Pelilogiikka oli erotettu graafisesta toteutuksesta, joten sitä olisi ollut helppo testata yksikkötesteillä. Näin ei kuitenkaan ajanpuutteen takia tehty. Oletuksia muuttujien arvoista tulostettiin komentoriveistä, jolloin niistä kävi heti ilmi pelilogiikassa olleet virheet.

Graafista käyttöliittymää testattiin pelaamalla peliä ja painelemalla nappeja. Kaikkia mahdollisia kombinaatioita kokeiltiin pelaamalla peliä.

8 Puutteet ja viat

Pelistä puuttuu täysin erillinen aloitusvalikko ja pelikentän valinta. Käyttäjäkokemuksen parantamiseksi peliin oltaisiin voitu lisätä myös uuden pelin aloitus pelin loputtua. Myös jonkinlainen pistelista olisi voinut olla paikoillaan. Nämä eivät kuitenkaan olleet pelattuuden kannalta merkittäviä, joten ne priorisoitiin pois.

9 Kolme parasta ja heikointa kohtaa

Ohjelman toteutuksessa pidän erityisen onnistuneena konfiguraatiotiedostojen laajaa käyttämistä. Konfiguraatiotiedoston formaatti ja lukija ovat yksinkertaisia. Koska eri torneihin ja vihollisiin liittyy monia parametreja, on selkeämpää että ne luetaan erillisestä tiedostosta. Vihollisten liikkumisalgoritmin toteutus on mielestäni yksinkertainen, mutta todella toimiva ajatellen muita olioita, jotka riippuvat tästä. Yleisesti ottaen projektin onnistunut luokkajako pieniin palasiin on mielestäni työn vahvuus. Erityisesti ammuksen toteuttaminen täysin graafisena helpotti pelilogiikan toteuttamista.

Pelikenttien tiedostomuotoa olisi voinut jalostaa vielä enemmän eteenpäin. Nykyinen kenttäruudukon esittäminen vie reilusti tilaa ja tiedostosta tulee suuri. Ruudukon olisi voinut pakata yhden xml-tagin sisään tiiviisti, ja purkaa täyteen kokoon vasta luettaessa. Graafisen käyttöliittymän hiiren käsittelyn tilakoneesta tuli hiukan monimutkainen, ja se sisältää sekavaa koodia. Qt:sta olisi varmasti löytynyt jotain valmiita rakenteita, jolla tätä olisi saanut yksinkertaistettua ja selkeytettyä. Graafisen ikkunan elementit ja niiden keskeiset järjestelyt olisi voinut toteuttaa erikseen Qt Designerilla ja tämän rakenteen tuoda ohjelmakoodiin tiedostosta. Tällä hetkellä elementtien määrittämiseen menee useita rivejä koodia, joilta olisi voinut välttyä.

10 Poikkeamat suunnitelmasta

Kaikki suunnitellut ominaisuudet toteutettiin. Toteutusjärjestys vaihteli kuitenkin oman mielen mukaisesti niin, että ensin kaikki järjestelmän peruspalaset tehtiin valmiiksi. Suunnitelmasta puuttui pari luokkaa, joiden sisällöt olin alunperin suunnitellut sisällytettäväksi muihin luokkiin. Ne oli kuitenkin ohjelman selkeyden kannalta hyvä toteuttaa erillisinä luokkina. Alunperin oli suunniteltu, että yhdessä ruudussa voisi olla vain yksi vihollinen kerrallaan, mutta eri vauhtia kulkevien vihollisten toteuttamiseksi täytyi ruudut laajentaa sisältämään useita vihollisia kerrallaan. Aikataulu oli osittain hiukan alakanttiin arvioitu, erityisesti graafista käyttöliittymää toteuttaessa. Tämän uskon johtuvan siitä, että Qt on minulle uusi kirjasto ja PyQT:n dokumentaatio on hyvin puutteellinen.

11 Toteutunut työjärjestys ja aikataulu

14.3.2017
Pelin toteuttaminen lähti liikkeelle pelilogiikalle olennaisten perusolioiden, kuten vihollinen, ruutu ja torni, toteuttamisesta. Kattavan teknisen suunnitelman ansiosta näiden toteuttamiseen kului aikaa arvioiduin mukaisesti noin tunnin verran. Seuraavaksi toteutettiin pelimaailma olio. Tähän kului arviota vähemmän aikaa noin kaksi tuntia, vaikka luokkaan toteutettiin suunniteltua enemmän erilaisia apufunktioita.
21.3.2017
Seuraavaksi suunnitelmasta poiketen toteutin lisää apufunktiota yksiköiden hallintaa varten ja synnytin luokan. Tällöin lisättiin pelimaailman tuki kierroksiin. Tähän kului arviolta kaksi tuntia.
22.3.2017
Ensimmäinen versio kenttätiedostojen lukijasta saatiin toimimaan. Myös muunnosskripti yksinkertaisten kenttien kääntämistä varten. Aikaa kului kaksi tuntia.
31.3.2017
Lisättiin testauksen helpottamista varten komentoriviin piirrettävä esitys pelimaailmasta sekä korjattiin ongelmat yksiköiden sijoittamisessa. Aikaa kului tunnin verran.
3.4.2017
Toteutettiin liikkumisalgoritmin ensimmäinen versio vihollisille. Liikkuminen testattiin toimivaksi aikaisemmin toteutettua pelimaailman piirtofunktiota apuna käyttäen.

Tässä välissä oli noin kolmen viikon mittainen tauko projektista johtuen kandidaatintyön aiheuttamista kiireistä. Projekti ei edennyt tällä välillä ollenkaan.

25.4.2017
Viimeisteltiin kenttätiedostojen lukeminen ja toteutettiin ensimmäinen versio

graafisesta käyttöliittymästä. Ensimmäinen versio reitinetsinnästä kenttätiedostonlukijaan toteutettiin. Aikaa kului arviolta seitsemän tuntia.

26.4.2017

Reitinetsintäalgoritmi kirjoitettiin kokonaan uudestaan, kun havaittiin edellisen version heikkous monessa tilanteessa. Lisättiin logiikan päivityskutsut graafiseen käyttöliittymään. Tässä vaiheessa alkeellinen peli pyöri ruudulla. Aikaa kului arviolta kahdeksan tuntia.

2.5.2017

Lisättiin tornien sijoittaminen hiirtä käyttäen. Myös ensimmäiset informaatiota välittävät tekstit lisättiin informaatioruutuun. Aikaa kului arviolta neljä tuntia.

3.5.2017

Toteutettiin konfiguraatiotiedostojen lukija. Lisättiin vihollisten paikkannusalgoritmi torneille. Kierrosten synnyttämismekanismi saatiin täysin toimivaksi. Lisättiin interpolaatio vihollisten liikkeeseen. Aikaa kului arviolta 12 tuntia.

4.5.2017

Kaikki tornien ja vihollisten ominaisuudet hyödyntävät luettuja konfiguraatioita. Lisättiin laser- ja sähköammukset. Päivitysjärjestelmän painikkeet lisättiin. Aikaa kului 12 tuntia.

5.5.2017

Päivityksen hyödyntävät tiedostoista luettuja arvoja. Lisättiin peli ohi ja voitto dialogit. Lisättiin indikaattoreita, kuten elämien ja kierroksien tilanne. Aikaa kului 13 tuntia.

12 Arvio lopputuloksesta

Pelin toteutus on varsin onnistunut. Peliä on miellyttävä pelata ja hiiriohjaus on intuitiivinen eikä töki. Toteutuksessa on käytetty runsaasti olio-ohjelmointia, ja siinä on hyvin vähän toistuvaa koodia. Erityisesti pelin laaja konfiguroitavuus onnistui mielestäni odotettua paremmin. Kattavan teknisen suunnitelman ansiosta peliä oli helppo lähteä toteuttamaan ja rakenne tuntui aiheuttavan vain vähän harmia poiketen suunnitellusta. Käyttöliittymän hiiren käsittelyn olisi voinut toteuttaa vielä selkeämmin. Pääasiassa valitut tiedostorakenteet ja algoritmit osoittautuivat sopivaksi käyttötarkoituksiin.

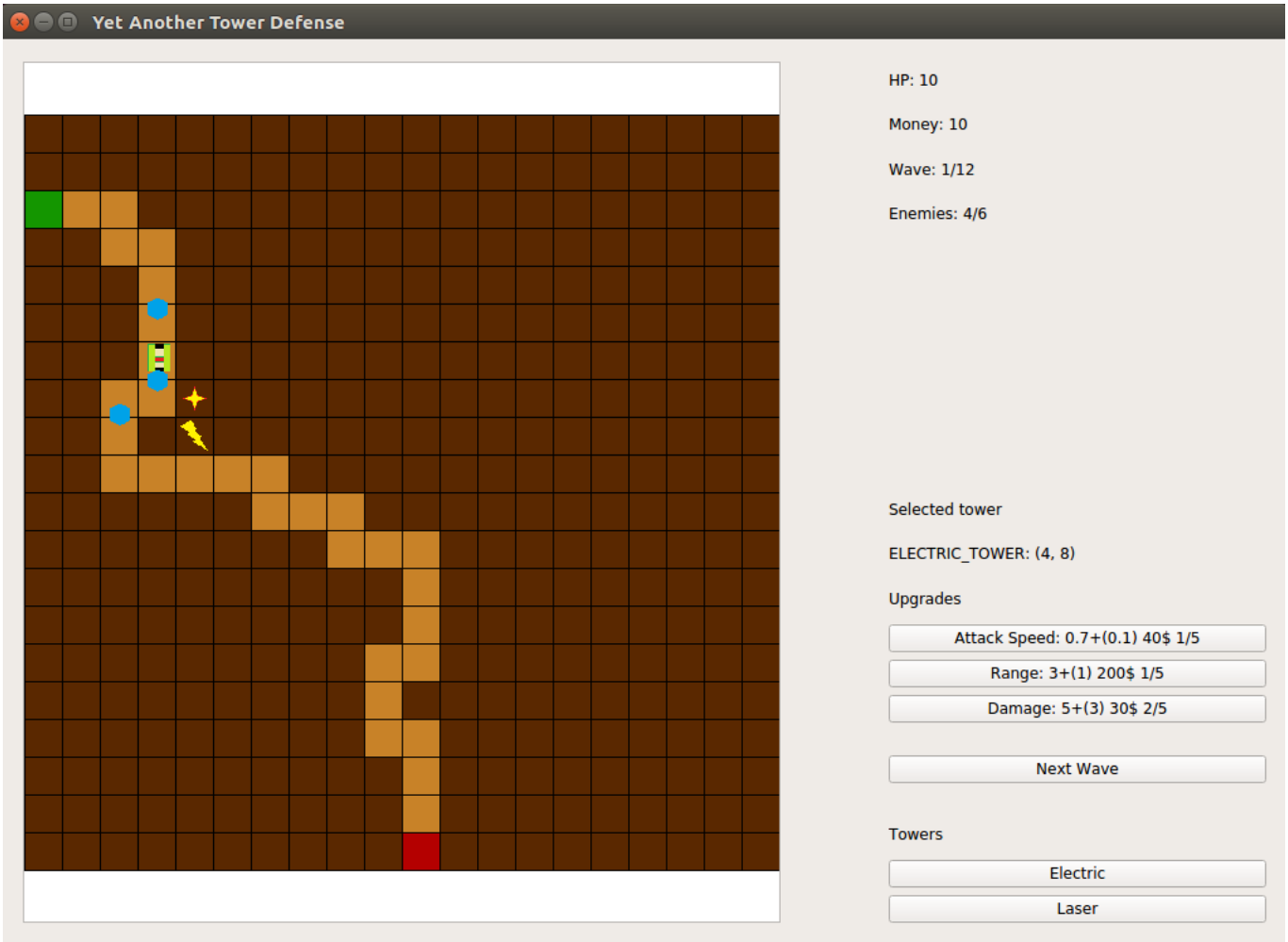
13 Viitteet

XML-tiedostojen lukuun:
`https://docs.python.org/3.6/library/xml.etree.elementtree.html`
PyQt5 dokumentaatio:
`http://pyqt.sourceforge.net/Docs/PyQt5/`
Qt 5.8 C++ dokumentaatio:
`https://doc.qt.io/qt-5/`
PyQt4 elementtien järjestely:
`http://zetcode.com/gui/pyqt4/layoutmanagement/`

Stackoverflow (monta!):
`http://stackoverflow.com`

14 Liitteet

Pelin lähdekoodi:
`https://version.aalto.fi/gitlab/lavikaj1/Towerdefence`



Kuva 2: Pelin käyttöliittymä