Pelitestauksen automatisointi

Turun yliopisto
Tietotekniikan laitos
Diplomityö
Ohjelmistotekniikka
Joulukuu 2024
Vesa Saarikko

TURUN YLIOPISTO
Tietotekniikan laitos

Vesa Saarikko: Pelitestauksen automatisointi

Diplomityö, 59 s., 10 liites.
Ohjelmistotekniikka
Joulukuu 2024

Testauksen tavoitteena on varmistaa tuotteen laatu, ja ohjelmistotestauksesta puhutaan silloin kun testattava tuote on ohjelmisto. Pelitestaus on ohjelmistotestauksen alatyyppi, jossa testataan videopelejä. Testaus on tärkeä osa videopelien kehitysprosessia, ja pelitestauksen automatisointi voi parantaa videopelien laatua. Tämän tutkielman aiheena on videopelien testaus automaattisesti ja manuaalisesti, näiden kahden testaustavan vertailu sekä automaattisen testauksen kehittäminen paremmaksi. Tavoitteena on tutkia automaattisen ja manuaalisen pelitestauksen eroja ja automaattisen pelitestauksen kehitysmahdollisuuksia.
Tässä tutkielmassa testataan yksinkertaista videopeliä, johon on injektoitu tarkoituksellisesti neljä erilaista bugia. Peliä testataan ensin manuaalisesti ja sen jälkeen automaattisesti. Manuaalinen testaus suoritetaan tätä tutkielmaa varten rekrytoitujen koehenkilöiden avulla, ja automaattinen testaus suoritetaan tähän tutkielmaan sopivaksi todetun automaattisen testauksen työkalun avulla. Sekä manuaalinen että automaattinen testaus koostuvat viidestä erilaisesta testitapauksesta.
Automaattisen ja manuaalisen testauksen tulosten analysoinnissa ja vertailussa käy ilmi että automaattisessa testauksessa löytyi tässä tutkimuksessa suurempi osa tarkoituksellisesti injektoiduista bugeista kuin manuaalisessa testauksessa. Tämän tutkimuksen tulosten perusteella voidaan todeta että automaattista testausta voidaan parantaa ainakin suorittamalla testitapaukset useamman kuin yhden kerran sekä rakentamalla testitapaukset niin, että ne vastaavat mahdollisimman hyvin manuaalisen testauksen testitapauksia.

Asiasanat: videopelit, pelitestaus, automaattinen testaus, manuaalinen testaus

Sisällys

Kuvat

Taulukot

1 Johdanto

Testaus on tärkeä osa videopelien kehitysprosessia, koska videopelien täytyy olla laadukkaita jotta ne voivat menestyä, ja siitä tulee koko ajan tärkeämpää videopelien kasvaessa ja monimutkaistuessa. Tämän tutkielman aiheena on videopelien testaus automaattisesti ja manuaalisesti, näiden kahden testaustavan vertailu sekä automaattisen testauksen kehittäminen paremmaksi.

Pelitestauksen automatisointi on merkittävä aihe, koska se voi auttaa bugien löytämisessä ja tarjota näin ollen hyvän mahdollisuuden parantaa videopelien laatua nostamatta kustannuksia. Automaattinen testaus ei voi videopelialalla korvata manuaalista testausta kokonaan, koska pelitestauksessa täytyy ottaa huomioon myös asioita joita vain manuaalinen testaaja voi arvioida, kuten pelin viihdyttävyys ja estetiikka. Se voi kuitenkin korvata merkittävän osan tällä hetkellä manuaalisesti tehtävästä testauksesta. Pelitestauksen automatisointi on kuitenkin haastava ja tutkimusta vaativa tehtävä.

Tämän tutkielman tavoitteena on tutkia automaattisen ja manuaalisen pelitestauksen eroja ja automaattisen pelitestauksen kehitysmahdollisuuksia. Tähän tavoitteeseen pyritään pääsemään vastaamalla kahteen tutkimuskysymykseen:

1. Saadaanko yksinkertaisen videopelin testauksessa paremmat tulokset automaattisella vai manuaalisella testauksella?

2. Miten automaattista testausta voisi parantaa?

Näihin tutkimuskysymyksiin etsitään vastauksia tätä tutkielmaa varten toteutetun yksinkertaisen videopelin ja sen ympärille rakennetun kokeellisen asetelman avulla. Tätä tutkielmaa varten toteutettavan videopelin ei tarvitse olla iso tai monimutkainen, vaan yksinkertainen peli riittää, koska tutkimuksen kohteena ei ole itse peli vaan pelin testaaminen. Sen sijaan pelissä täytyy olla jonkinlaisia bugeja, jotta sen testaaminen voisi tuottaa riittävästi tuloksia automaattisen ja manuaalisen testauksen vertailua varten.

Tässä tutkielmassa testataan yksinkertaista Unity-pelimoottorilla ja C#-ohjelmointikielellä toteutettua videopeliä, johon on injektoitu tarkoituksellisesti bugeja. Peliä testataan ensin manuaalisesti ja sen jälkeen automaattisesti. Manuaalinen testaus suoritetaan tätä tutkielmaa varten rekrytoitujen koehenkilöiden avulla, ja automaattinen testaus suoritetaan tähän tutkielmaan sopivaksi todetun automaattisen testauksen työkalun AltTesterin avulla. Jokaisen bugin ympärille rakennetaan oma testitapaus, ja sekä manuaalinen että automaattinen testaus koostuvat näistä testitapauksista sekä avoimesta testauksesta, jossa ei keskitytä minkään tietyn bugin löytämiseen. Testauksen jälkeen analysoidaan ja vertaillaan manuaalisen ja automaattisen testauksen tuloksia.

Luvussa 2 tutustutaan ohjelmistotestaukseen. Alaluvussa 2.1 käydään läpi ohjelmistotestauksen perusasioita ja alaluvussa 2.2 erilaisia ohjelmistotestauksen tasoja. Luvussa 3 perehdytään pelitestaukseen, joka on ohjelmistotestauksen alatyyppi. Alaluvussa 3.1 käsitellään pelitestauksen ja yleisen ohjelmistotestauksen eroja ja alaluvussa 3.2 pelitestauksen automatisointia ja sen haasteita.

Luvussa 4 esitellään automaattisen pelitestauksen työkaluja. Tässä tutkielmassa käytetyn työkalun lisäksi esiteltävänä on kolme muuta työkalua, jotka olivat vaihtoehtoina tässä tutkielmassa käytettävää työkalua valittaessa. Luvussa 5 esitellään tutkimusasetelma, mikä tarkoittaa sitä että ensin esitellään tätä tutkielmaa varten

toteutettu videopeli ja sen jälkeen peliin injektoidut bugit. Pelin esittelyssä käydään läpi pelin perusidea ja säännöt ja kerrotaan miten peliä pelataan.

Luvussa 6 käydään läpi edellä mainitun pelin testaamista varten rakennetut testitapaukset sekä näiden testitapausten mukaisesti suoritettu testausprosessi, joka sisältää sekä automaattisen testauksen että manuaalisen testauksen. Luvussa 7 esitetään testausprosessin tulokset ja analysoidaan niitä sekä vertaillaan automaattisen ja manuaalisen testauksen tuloksia keskenään. Luvussa 8 tehdään yhteenveto tutkimuksesta ja sen tuloksista ja pohditaan miten pelitestauksen automatisointiin tähtäävää tutkimusta voitaisiin jatkaa.

2 Ohjelmistotestaus

Sanalla testaus tarkoitetaan yleensä sitä, että suoritetaan tietyntyyppisiä kokeita, joiden tarkoituksena on varmistaa tuotteen laatu. Ohjelmistotestauksesta puhutaan silloin kun tuote, jonka laatu halutaan varmistaa, on nimenomaan ohjelmisto. Motivaatio tuotteen laadun varmistamiselle on yleensä se, että tuotteen ostajalle halutaan tarjota paras mahdollinen käyttökokemus ja mahdollisuus nauttia tuotteesta ja sen käyttämisestä ilman ongelmia. [1]

Ohjelmistotestauksessa suoritetaan äärellinen määrä testitapauksia ja varmistetaan dynaamisesti, että ohjelma toimii odotetulla tavalla näissä testitapauksissa. Suoritettavat testitapaukset valitaan sopivalla tavalla kaikkien mahdollisten testitapausten joukosta, joka on yleensä ääretön. [2, Luku 4.]

Sanalla dynaaminen tarkoitetaan sitä, että ohjelmaa testataan suorittamalla se valituilla syötteillä. Syötteen arvo ei kuitenkaan yksinään riitä testitapauksen määrittelemiseen, koska monimutkainen ohjelmisto voi reagoida samaan syötteeseen erilaisilla tavoilla riippuen ohjelmiston tilasta. [2, Luku 4.]

Suoritettavien testitapausten määrä on äärellinen, koska yksinkertaisellakin ohjelmalla on niin monta teoreettisesti mahdollista testitapausta, että kattavaan testaukseen kuluisi kuukausia tai jopa vuosia. Näin ollen kaikkien mahdollisten testitapausten joukkoa voidaan pitää äärettömänä. Ohjelmistotestauksessa käytetään kaikkien mahdollisten testitapausten joukon äärellistä osajoukkoa, koska käytettä-

vissä olevien resurssien rajallisuuden ja ohjelman vaatimien testien rajattomuuden välillä on pakko tehdä jonkinlainen kompromissi. [2, Luku 4.]

Testitapausten valitsemisessa on olennaisia eroja erilaisten ohjelmistotestaustekniikoiden välillä. Tämä tarkoittaa sitä, että ohjelmistoinsinöörien täytyy tiedostaa, että valintakriteereillä voi olla merkittävä positiivinen tai negatiivinen vaikutus testauksen tehokkuuteen. Sopivimpien valintakriteerien tunnistaminen on monimutkainen ongelma, jonka ratkaisemiseksi käytetään riskianalyysitekniikoita ja ohjelmistotekniikan asiantuntemusta. [2, Luku 4.]

Testaamisen jälkeen täytyy olla mahdollista päättää, onko testin tulos hyväksyttävä, koska muuten testaaminen on hyödytöntä. Päätöksen tekemisen ei kuitenkaan tarvitse aina olla helppoa. Ohjelman havaittua toimintaa voidaan verrata käyttäjän tarpeisiin, ohjelman määrittelyyn tai epäsuorien vaatimusten tai odotusten perusteella ennustettuun toimintaan. [2, Luku 4.]

2.1 Ohjelmistotestauksen perusasioita

Tässä alaluvussa käydään läpi ohjelmistotestauksen peruskäsitteitä. Näiden käsitteiden läpikäynnin tarkoituksena on auttaa ymmärtämään myöhemmin tässä tutkielmassa käsiteltäviä asioita. Koska pelitestaus on ohjelmistotestauksen alatyyppi, nämä käsitteet liittyvät myös pelitestaukseen.

Virhetoiminta, virhe ja vika

Virhetoiminta (engl. failure) tapahtuu kun järjestelmän havaittu toiminta ei vastaa järjestelmän toiminnallista määrittelyä. Virhe (engl. error) on järjestelmän tila. Jos virhetilaa ei korjata, se voi johtaa virhetoimintaan. Vika (engl. fault) on virheen taustalla oleva syy. [3, Luku 1.] Nämä kolme käsitettä on tärkeää erotella toisistaan. Järjestelmässä voi olla vikoja, jotka eivät koskaan johda virhetoimintaan, ja toisaalta

virhetoiminnan syytä ei ole aina mahdollista määritellä yksiselitteisesti, koska ei ole olemassa teoreettisia kriteerejä, joiden avulla voitaisiin yleisellä tasolla määritellä havaitun virhetoiminnan aiheuttanut vika. [2, Luku 4.]

Testin valinta- ja riittävyyskriteerit

Testin valintakriteerit ovat erilaisia tapoja valita testitapauksia tai todeta, että jokin testitapausten sarja on tai ei ole riittävä tiettyyn tarkoitukseen. Testin riittävyyskriteerejä voidaan käyttää, jos halutaan arvioida, onko suunniteltu tai jo suoritettu testitapausten sarja riittävä. [2, Luku 4.]

Testauksen tehokkuus

Testauksen tehokkuutta on hyvä arvioida jollain tavalla. Yksi yleisesti käytetty testauksen tehokkuuden mittari on sellaisten asiakkaan löytämien virheiden määrä, joita ei löydetty ohjelmaa testattaessa. Näiden virheiden lukumäärän laskeminen on vaikea tehtävä, mutta tätä lukumäärää voidaan arvioida esimerkiksi laskemalla niiden virheiden lukumäärä, jotka asiakkaat ovat löytäneet ensimmäisten kuuden kuukauden aikana. Tätä mittaria voidaan käyttää sen jälkeen kun ohjelma on julkaistu asiakkaiden käytettäväksi. [3, Luku 13.]

Testaus virheiden löytämiseksi

Kun järjestelmää testataan virheiden löytämiseksi, testiä voidaan pitää onnistuneena, jos se aiheuttaa järjestelmän virheellisen toiminnan. Testaus virheiden löytämiseksi on näin ollen eri asia kuin sellainen testaus jonka tarkoituksena on osoittaa, että järjestelmän toiminta vastaa sen toiminnallista määrittelyä eli sitä miten järjestelmän on tarkoitus toimia. Kun järjestelmää testataan sen toiminnallista määrittelyä vastaan, onnistunut testi ei aiheuta ohjelman virheellistä toimintaa realistisissa testitapauksissa ja testiympäristöissä. [2, Luku 4.]

Oraakkeliongelma

Oraakkeli on ihminen, ohjelma, prosessi tai tietokokonaisuus, joka kertoo tietyn testin tai testien sarjan odotetun lopputuloksen. Testi on merkityksellinen vain silloin kun on mahdollista päättää onko sen lopputulos hyväksyttävä. Ideaalissa tilanteessa testin odotettu lopputulos lasketaan testin suunnitteluvaiheessa eli ennen kuin ohjelma suoritetaan valitulla testisyötteellä. Ideana on, että ohjelman vaatimukset ymmärtävän henkilön pitäisi pystyä laskemaan testin odotettu lopputulos. Jos tämä toteutuu, testin lopputuloksen hyväksyttävyyden päättäminen on mahdollista myös järjestelmän kehittäjien suunnittelemissa testitapauksissa. [3, Luku 1.]

Testaukseen liittyvät rajoitteet

Ohjelmistotestaukseen liittyy tiettyjä rajoitteita. Yksi rajoite on se, että testauksella ei voida aukottomasti todistaa, että ohjelma toimii oikein. Toisin sanoen ohjelmaa voidaan testata sopivalla kaikkien mahdollisten syötteiden joukon osajoukolla havaitsematta virhetoimintaa, mutta testauksen perusteella ei voida tehdä johtopäätöstä, että ohjelmassa ei ole ollenkaan vikoja. Toinen rajoite on se, että useissa testitapauksissa ei tiedetä mikä on odotettu tuloste, jolloin havaitun tulosteen oikeellisuutta ei voida todistaa. Jos jonkin testisyötteen kohdalla on sellainen tilanne, että odotettua tulostetta ei tiedetä ja sen määritteleminen veisi kohtuuttoman paljon aikaa, kyseisen testin suorittamisesta ei ole juurikaan hyötyä. [3, Luku 2.]

Mahdottomien polkujen ongelma

Toimintopolku tarkoittaa toimintojen sarjaa, joka suoritetaan kun lähdetään liikkeelle ohjelman alkamispisteestä ja päädytään ohjelman päättymispisteeseen. Ohjelmalla voi olla suuri tai jopa ääretön määrä erilaisia mahdollisia toimintopolkuja, joista jokaiseen liittyy jonkinlainen syöte ja odotettu tuloste. [3, Luku 4.] Mahdottomat polut ovat sellaisia toimintopolkuja, joita ei ole mahdollista suorittaa millään

syötteellä. Ne ovat merkittävä ongelma polkuihin perustuvassa ohjelmistotestauksessa, erityisesti silloin kun pyritään luomaan automatisoidusti testisyötteitä erilaisten toimintopolkujen suorittamiseksi. [2, Luku 4.]

Testattavuus

Termillä ”ohjelmiston testattavuus” on kaksi toisiinsa liittyvää mutta erilaista merkitystä, joista molemmat ovat tärkeitä: sillä voidaan tarkoittaa joko sitä, kuinka helposti jokin testauksen kattavuuteen liittyvä kriteeri voidaan täyttää, tai sitä, kuinka todennäköisesti jokin testien sarja paljastaa ohjelmiston virhetoiminnan, jos ohjelmistossa on yksi tai useampi vika. Tämä todennäköisyys on joissain tapauksissa mahdollista mitata tilastollisesti. [2, Luku 4.]

Testattavuus voidaan määritellä myös niin, että se tarkoittaa mahdollisuutta varmistaa järjestelmän toiminnalle asetettujen vaatimusten toteutuminen. On tärkeää, että kaikkien tällaisten vaatimusten toteutuminen on mahdollista varmistaa. Testattavuuden näkökulma on tärkeää ottaa huomioon jokaisessa ohjelmistokehitysprosessin vaiheessa. Jokaisen vaatimuksen kohdalla on tärkeää miettiä millä tavalla ja kuinka helposti kyseisen vaatimuksen toteutuminen voidaan varmistaa. Testattavuus voi vaatia sitä, että järjestelmään sisällytetään ominaisuuksia, jotka eivät ole asiakkaan käytettävissä. [3, Luku 17.]

Testauksen suhde muihin aktiviteetteihin

Ohjelmistotestaus on käsitteenä lähellä staattisia laadunhallintatekniikoita, oikeellisuuden todistamista, debuggausta ja ohjelman rakentamista, mutta se eroaa kuitenkin jollain tavalla kaikista näistä käsitteistä. Tästä huolimatta on hyödyllistä tarkastella ohjelmistotestausta ohjelmiston laadunvarmistuksesta vastaavien analyytikoiden ja todistajien näkökulmasta. [2, Luku 4.]

2.2 Testauksen tasot

Ohjelmistotestausta tehdään yleensä erilaisilla tasoilla koko ohjelmistokehitys- ja ylläpitoprosessin ajan. Erilaiset tasot voidaan erottaa toisistaan testauksen kohteen tai testauksen tavoitteen perusteella. [2, Luku 4.]

Testauksen kohde

Testauksen kohde voi vaihdella ja testaus voidaan jakaa kohteen perusteella kolmeen vaiheeseen, jotka ovat yksikkötestaus, integraatiotestaus ja järjestelmätestaus. Yksikkötestauksessa kohteena on järjestelmän yksittäinen moduuli eli yksikkö, integraatiotestauksessa kohteena on joukko toisiinsa esimerkiksi käyttötarkoitukseltaan, toiminnaltaan tai rakenteeltaan liittyviä yksiköitä ja järjestelmätestauksessa kohteena on koko järjestelmä. Kaikki kolme vaihetta ovat yhtä tärkeitä. [2, Luku 4.]

Yksikkötestaus

Yksikkötestauksessa testataan järjestelmän yksittäisten osien, kuten proseduurien, funktioiden, metodien ja luokkien, toimintaa erillään järjestelmän muista osista. Näin tehdään sen takia, että tällaisessa testauksessa tiedetään heti mistä järjestelmän osasta löydetyt virheet ovat peräisin, mikä helpottaa virheiden korjaamista. Yksikkötestauksen suorittaa se ohjelmoija, joka on kirjoittanut testattavan osan koodin, koska hän tuntee testattavan osan yksityiskohdat. [3, Luku 1.] Testaajalla on yksikkötestauksessa yleensä mahdollisuus päästä tarvittaessa käsiksi testattavan osan koodiin ja käyttää debuggaustyökaluja. [2, Luku 4.]

Integraatiotestaus

Integraatiotestauksessa valitaan järjestelmästä joukko toisiinsa liittyviä osia ja testataan niiden välistä vuorovaikutusta. Testauksen suorittavat ohjelmistokehittäjät

yhdessä testaukseen erikostuneiden insinöörien kanssa. Tavoitteena on tehdä järjestelmästä mahdollisimman vakaa. [3, Luku 1.] Integraatiotestausta tehdään yleensä kaikissa sellaisissa ohjelmistokehitysprosessin vaiheissa, joissa järjestelmän osien integrointi etenee alemmalta tasolta ylemmälle tasolle. [2, Luku 4.] Integraatiotestauksen sanotaan olevan valmis, kun kaikki järjestelmän osat on integroitu täysin yhtenäiseksi järjestelmäksi, kaikki testitapaukset on suoritettu, kaikki järjestelmästä löydetyt merkittävät virheet on korjattu ja järjestelmä on testattu tämän jälkeen vielä uudelleen. [3, Luku 1.]

Järjestelmätestaus

Järjestelmätestauksessa testataan koko järjestelmän toimintaa. Tehokas yksikkö- ja integraatiotestaus on yleensä paljastanut suuren osan järjestelmän vioista ennen kuin järjestelmätestausta päästään tekemään. Näin ollen järjestelmätestauksessa pyritään yleensä arvioimaan sitä, miten hyvin järjestelmä täyttää sille asetetut ei-toiminnalliset vaatimukset, joita voivat olla esimerkiksi turvallisuus, tarkkuus, nopeus ja luotettavuus. [2, Luku 4.]

Testauksen tavoitteet

Testaukseen liittyy tiettyjä tavoitteita, jotka voidaan ilmaista enemmän tai vähemmän selkeästi ja tarkasti. Tavoitteiden ilmaiseminen tarkasti ja kvantitatiivisesti auttaa testausprosessin mittaamisessa ja kontrolloimisessa. Testaus voidaan kohdistaa erilaisiin järjestelmän ominaisuuksiin ja niiden oikeanlaisen toiminnan varmistamiseen. Testitapauksia voidaan suunnitella niin, että niiden avulla voidaan tarkistaa onko järjestelmän toiminnallinen määrittely toteutettu oikein, mutta on myös mahdollista testata järjestelmän ei-toiminnallisia ominaisuuksia, kuten suorituskykyä, luotettavuutta ja käytettävyyttä. Muita tärkeitä testauksen tavoitteita ovat esimerkiksi tietoturvauhkien tunnistaminen ja järjestelmän hyväksymiskriteerien täyttä-

misen varmistaminen. Testauksen tavoitteet vaihtelevat sen mukaan, mikä on testauksen kohde. Testauksen eri tasoilla tavoitellaan erilaisia asioita. [2, Luku 4.]

Hyväksymistestaus

Hyväksymistestaus määrittelee sen, täyttääkö järjestelmä sille asetetut hyväksymiskriteerit. Yleensä hyväksymistestaus tapahtuu vertaamalla järjestelmän haluttua toimintaa asiakkaan asettamiin vaatimuksiin. Asiakas tai asiakkaan edustaja määrittelee tai suorittaa toimintoja, joilla tarkistetaan, täyttääkö järjestelmä asiakkaan asettamat vaatimukset, tai jos kyseessä on kuluttajatuote, täyttääkö tuotteen kehittänyt organisaatio tuotteen kohderyhmään liittyvät vaatimukset. Järjestelmän kehittäjät voivat osallistua hyväksymistestaukseen, mutta hyväksymistestaus voidaan myös suorittaa ilman järjestelmän kehittäjien osallistumista. [2, Luku 4.]

Asennustestaus

Asennustestauksessa tarkistetaan, toimiiko järjestelmä oikein, kun se asennetaan kohdeympäristöön. Tämä tapahtuu yleensä järjestelmä- ja hyväksymistestauksen jälkeen. Asennustestaus voidaan nähdä järjestelmätestauksena, joka toteutetaan laitteiston ja muiden toiminnallisten rajoitteiden määrittelemässä toimintaympäristössä. Asennusproseduurien toiminta voidaan myös tarkistaa. [2, Luku 4.]

Alfa- ja betatestaus

Alfa- ja betatestaus ovat vaiheita, joista jompikumpi tai molemmat suoritetaan ennen kuin ohjelmisto julkaistaan, jos se koetaan tarpeelliseksi. [2, Luku 4.] Alfatestauksessa valittu joukko ohjelmiston mahdollisia käyttäjiä työskentelee tiiviisti kehitystiimin kanssa ja testaa ohjelmiston varhaisia versioita. Betatestauksessa ohjelmiston versio annetaan käytettäväksi suuremmalle joukolle mahdollisia käyttäjiä, jotka saavat testata ohjelmistoa ja voivat raportoida löytämistään ongelmista kehitystii-

mille. [4, Luku 8.] Alfa- ja betatestaus ovat yleensä kontrolloimattomia ja niihin ei välttämättä tarvitse viitata testaussuunnitelmassa. [2, Luku 4.]

Luotettavuuden saavuttaminen ja arviointi

Testaaminen on tärkeää järjestelmän luotettavuuden kannalta. Testaamalla järjestelmää voidaan tunnistaa ja korjata järjestelmässä olevia vikoja ja näin ollen parantaa järjestelmän luotettavuutta. Tämän lisäksi järjestelmän luotettavuutta voidaan arvioida tilastollisesti generoimalla satunnaisia testitapauksia järjestelmän toiminnallisen profiilin perusteella. [2, Luku 4.]

Regressiotestaus

Regressiotestauksessa ei suunnitella uusia testitapauksia, vaan testitapaukset valitaan olemassa olevien testitapausten joukosta, jotta voidaan varmistaa, että järjestelmän uudessa versiossa ei ole sellaisia vikoja, joita ei ollut aiemmassa versiossa. Regressiotestauksen tarkoituksena on varmistaa, että järjestelmän muuttumattomaan osaan ei ole ilmestynyt vikoja sen takia, että muualla järjestelmässä on tehty muutoksia. Regressiotestaus on paljon resursseja vaativa tehtävä, joten testitapaukset valitaan huolellisesti, jotta voidaan maksimoida vikojen löytymisen todennäköisyys ja minimoida testauksessa tarvittavien resurssien määrä. [3, Luku 8.]

Suorituskykytestaus

Suorituskykytestauksen tarkoituksena on määrittää järjestelmän suorituskyky verrattuna odotettuun suorituskykyyn. Tarvittavat suorituskyvyn mittarit eivät ole jokaiselle järjestelmälle samat vaan ne vaihtelevat järjestelmäkohtaisesti. [3, Luku 8.] Suorituskyvyn mittareita voivat olla esimerkiksi kapasiteetti ja vasteaika. [2, Luku 4.] Jotta järjestelmän suorituskykyä voidaan arvioida, suorituskykytestauksessa täytyy määritellä selkeästi, millaista dataa testauksen aikana kerätään. Jos suori-

tuskykytestauksen tulos ei ole tyydyttävä, järjestelmän suorituskykyä pyritään parantamaan esimerkiksi kirjoittamalla järjestelmän tai sen osan koodi uudelleen tai suunnittelemalla järjestelmä uudelleen. [3, Luku 8.]

Tietoturvatestaus

Tietoturvatestauksessa varmistetaan, että järjestelmä täyttää tietoturvavaatimukset, jotka ovat luottamuksellisuus, koskemattomuus ja saatavuus. Luottamuksellisuus tarkoittaa sitä, että järjestelmän dataa ja prosesseja ei voida hyödyntää luvattomasti. Koskemattomuus tarkoittaa sitä, että järjestelmän dataa ja prosesseja ei voida muuttaa luvattomasti. Saatavuus tarkoittaa sitä, että järjestelmän data ja prosessit on suojattu palvelunestohyökkäyksiä vastaan. Monen järjestelmän toiminnallinen määrittely ei sisällä negatiivisia tai rajoittavia vaatimuksia, mutta tietoturvatestauksen olisi hyvä sisältää negatiivisia tilanteita, kuten järjestelmän väärinkäyttö. Tietoturvatestauksen tavoitteena on varmistaa muun muassa se, että järjestelmä toimii kaikissa tilanteissa - sekä odotetuissa että odottamattomissa - tietoturvallisesti ja johdonmukaisesti, ja se, että virhetilanteet ja virheiden ja poikkeusten käsittelymekanismit toimivat kaikissa olosuhteissa niin, että ne eivät altista järjestelmää tai siihen liittyvää dataa tai resursseja tietoturvahyökkäyksille. [3, Luku 8.]

Stressitestaus

Stressitestauksessa arvioidaan järjestelmän toimintaa tilanteessa, jossa kuormitus ylittää järjestelmän suunnitellun kapasiteetin. Järjestelmää kuormitetaan tarkoituksellisesti ja se viedään suunnitelluille rajoille ja niiden yli. Stressitestaus sisältää tarkoituksellisesti aiheutettua kilpailua resursseista ja mahdollisten yhteensopivuusongelmien etsimistä. Stressitestauksen tarkoituksena on varmistaa, että järjestelmä toimii hyväksyttävästi huonoimmissa mahdollisissa olosuhteissa kuormituksen ollessa niin suuri kuin sen voidaan odottaa suurimmillaan olevan. [3, Luku 8.]

Back-to-Back-testaus

Back-to-Back-testauksessa suoritetaan vähintään kaksi erilaista järjestelmän varianttia samalla syötteellä ja vertaillaan näiden varianttien antamia tulosteita toisiinsa. Tilanteessa, jossa tulosteiden välillä ilmenee epäjohdonmukaisuutta, pyritään löytämään asialle selitys ja korjaamaan mahdolliset virheet. [5]

Palautumistestaus

Palautumistestauksen tarkoituksena on testata sitä, miten hyvin järjestelmä pystyy palautumaan virhetilannetta edeltävään tilaan, kun se on kaatunut virhetilanteen takia ja käynnistyy uudestaan. [2, Luku 4.] Palautumistestauksessa voidaan arvioida esimerkiksi sitä, kuinka paljon dataa järjestelmä pystyy palauttamaan kaatumisen jälkeen ja onko palautettu data johdonmukaista. [3, Luku 14.]

Käyttöliittymätestaus

Käyttöliittymätestauksen tarkoituksena on varmistaa, että järjestelmän komponenttien ja käyttöliittymän välinen yhteys toimii oikein, jolloin data kulkee järjestelmässä oikealla tavalla. Käyttöliittymän liittyvät ongelmat ovat yleisiä monimutkaisissa järjestelmissä, joten käyttöliittymätestausta tarvitaan. Testitapaukset generoidaan yleensä käyttöliittymän määrittelyn perusteella. [2, Luku 4.]

Konfiguraatiotestaus

Konfiguraatiolla tarkoitetaan tässä yhteydessä erilaisten laitteisto- ja ohjelmistokomponenttien yhdistelmiä. Komponentteja voivat olla esimerkiksi käyttöjärjestelmät ja niiden versiot, erikokoiset muistit sekä erilaiset selaimet, ajurit, kovalevyt ja prosessorit. [6] Konfiguraatiotestauksen tarkoituksena on varmistaa järjestelmän toiminta erilaisissa konfiguraatioissa. Näin ollen konfiguraatiotestausta tehdään sellai-

sissa tapauksissa, joissa testattava järjestelmä on suunniteltu käytettäväksi erilaisille käyttäjille ja erilaisissa konfiguraatioissa. [2, Luku 4.]

Käytettävyystestaus

Käytettävyystestauksessa tutkitaan ihmisen ja tietokoneen välisen vuorovaikutuksen toimivuutta. Käytettävyystestauksen tärkein tehtävä on arvioida sitä, miten helppoa järjestelmän käyttäjien on oppia käyttämään järjestelmää ja käyttää sitä. Käytettävyystestauksessa voidaan testata esimerkiksi toimintoja, joiden tarkoituksena on tukea käyttäjän suorittamia tehtäviä, dokumentaatiota, jonka tarkoituksena on auttaa käyttäjää mahdollisissa ongelmatilanteissa, ja järjestelmän kykyä palautua käyttäjän tekemien virheiden aiheuttamista ongelmatilanteista. [2, Luku 4.]

3 Pelitestaus

Pelitestaus on ohjelmistotestauksen alatyyppi. Pelitestauksesta puhutaan silloin kun ohjelmisto, jonka laatu halutaan varmistaa, on nimenomaan videopeli. Pelitestaus on tärkeä osatekijä videopelien menestyksen taustalla ja sen merkitys kasvaa koko ajan sitä mukaa kun videopeleistä tulee isompia ja monimutkaisempia. Videopelin laadunvarmistus on prosessi, jossa videopeli testataan perinpohjaisesti ja yritetään löytää virheitä sen sisältä. Pelitestaus pitää sisällään monia erilaisia osa-alueita, joiden painotus vaihtelee sen mukaan, onko testattavan videopelin alustana tietokone, konsoli vai kannettava laite. [1]

Videopelialalla ensivaikutelma on erittäin tärkeä. Näin ollen vain laadukkaiden videopelien voidaan odottaa menestyvän. Riippumatta budjetista ja kehitystiimin koosta, videopeleissä on kuitenkin usein bugeja vielä niiden julkaisuvaiheessa. Joissakin tapauksissa tällaiset bugit onnistutaan korjaamaan, mutta joissakin tapauksissa videopeli unohdetaan tällaisten bugien takia. [7]

Ohjelmiston laatu voidaan määritellä sen mukaan kuinka hyvin ohjelmisto suorittaa ne toiminnot, joita varten se on kehitetty. Videopelin laatu sisältää tämän lisäksi pelaajalle tarjotun pelikokemuksen laadun ja sen kuinka hyvin pelin ominaisuudet on toteutettu. Pelikokemus ei ole jokaiselle pelaajalle samanlainen vaan se vaihtelee pelaajakohtaisesti, koska erilaiset pelaajat arvostavat erilaisia asioita videopeleissä. Pelikokemuksen laatuun vaikuttavia tekijöitä voivat olla esimerkik-

si tarina, pelimekaniikat, ääniefektit, visuaalinen ilme, huumori, liioittelu, tekoälyn ohjaamien hahmojen ihmismäisyys sekä käyttöliittymän selkeys. [8]

Pelitestaus sisältää usein testien sarjan, joka kattaa videopelin jokaisen osa-alueen muotoilusta ja rakenteesta suorituskykyyn ja käyttökokemukseen. Testeistä saatavan datan ja palautteen perusteella voidaan korjata pelistä löytyneitä bugeja ja tehdä peliin parannuksia. Pelitestaus on usein iteratiivista, mikä tarkoittaa sitä, että samat testit suoritetaan monta kertaa ja testien välissä peliä pyritään parantelemaan edellisellä testikerralla saadun datan ja palautteen pohjalta. Testejä toistetaan niin monta kertaa, että peli saadaan riittävän hyvään kuntoon. [9]

Pelitestauksessa täytyy ottaa huomioon videopelien erityispiirteet, joita yleisessä ohjelmistotestauksessa ei tarvitse huomioida. Pelitestaajien ja ohjelmistotestaajien pitäisi työskennellä yhdessä ja täydentää toistensa taitoja. Ei ole olemassa yhtä kaikille peliprojekteille sopivaa testausprosessia, koska erilaisia ja erityyppisiä videopelejä on olemassa valtava määrä. Pelistudiot tiedostavat testauksen tärkeyden, mikä tarjoaa mahdollisuuden videopelien testaamiseen avoimen lähdekoodin avulla. Suunnitelmallisuuden puute, testauksen huono kattavuus ja muut vastaavanlaiset ongelmat ovat erittäin hyviä syitä siihen, että videopelejä pitäisi testata jo aikaisessa vaiheessa niiden kehitysprosessia. [7]

3.1 Pelitestauksen ja yleisen ohjelmistotestauksen erot

Videopelit vaativat testaamista siinä missä muutkin ohjelmistot, ja sekä pelitestauksen että yleisen ohjelmistotestauksen kysyntä kasvaa koko ajan. Pelitestauksella ja yleisellä ohjelmistotestauksella on aina sama päämäärä, mutta niiden välillä on myös monenlaisia eroavaisuuksia. [10] Tässä alaluvussa käydään läpi näitä eroavaisuuksia ja niiden vaikutusta pelien ja muiden ohjelmistojen testaamiseen.

Pelien erityispiirteet

Pelitestaus ei voi keskittyä ainoastaan yleisen ohjelmistotestauksen näkökulmiin, vaan videopelien laadun ja onnistuneen testausprosessin takaamiseksi sen täytyy sisältää myös pelin tasapainoon, fysiikkaan ja viihdyttävyyteen liittyvien asioiden havainnointia ja tutkimista. Ihmisen ja tietokoneen välinen vuorovaikutus on videopeleissä tyypillisesti monimutkaisempaa kuin muissa ohjelmistoissa. Videopeleille voidaan asettaa mittareita, joita on vaikea havainnoida ja testata, kuten käyttäjän toiminta, viihdyttävyys ja hauskuus. Näillä mittareilla voi olla suora vaikutus pelitestaukseen liittyviin aktiviteetteihin, kuten testaamisen suunnitteluun, testitapausten kehittämiseen ja jopa testien suorittamiseen. Näin ollen pelitestaajien täytyy olla tietoisia videopelien ja muiden ohjelmistojen välisistä eroista, jotta he voivat parantaa testausprosessia. [11]

Monimutkaisuus

Pelitestaus on monimutkaisempaa kuin yleinen ohjelmistotestaus. Siinä missä yleinen ohjelmistotestaus on systemaattista ja suhteellisen yksinkertaista, pelitestaus sisältää paljon monimutkaisia asioita. Pelitestaus voi vaikuttaa hauskalta tehtävältä, mutta todellisuudessa se vaatii omistautumista, pienten yksityiskohtien huomioimista ja paljon keskittymistä. Manuaalisen testauksen tarve lisää pelitestauksen monimutkaisuutta yleiseen ohjelmistotestaukseen verrattuna. [10]

Automaattinen ja manuaalinen testaus

Pelitestaus ei tällä hetkellä sisällä juurikaan automaattista testausta vaan suurin osa testeistä tapahtuu manuaalisesti, kun taas yleisessä ohjelmistotestauksessa asia on päinvastoin. Yleisessä ohjelmistotestauksessa pyritään suorittamaan mahdollisimman suuri osa testeistä automaattisesti. [10]

Testaustiimi

Pelitestaus vaatii myös testaustiimiltä erilaisia asioita ja ominaisuuksia kuin yleinen ohjelmistotestaus. Koska pelitestaus painottuu tällä hetkellä manuaaliseen testaukseen, kun taas yleinen ohjelmistotestaus painottuu automaattiseen testaukseen, pelitestaus vaatii tällä hetkellä suuremman testaustiimin kuin yleinen ohjelmistotestaus. Toisaalta pelitestaus ei kuitenkaan testaustiimin suuremmasta koosta huolimatta vaadi tiimin jäsenten välistä yhteistyötä, kun taas yleinen ohjelmistotestaus vaatii tiimityöskentelyä. [10]

Alustat

Erilaisten alustojen määrä on myös yksi merkittävä ero pelitestauksen ja yleisen ohjelmistotestauksen välillä. Videopelejä käytetään yleensä useammalla erilaisella alustalla kuin muita ohjelmistoja, ja näin ollen videopelejä täytyy yleensä myös testata useammalla erilaisella alustalla. Yleisessä ohjelmistotestauksessa ohjelmistoja on yleensä tarpeellista testata ainoastaan verkko- ja mobiilialustoilla, kun taas videopelejä on usein tarpeellista testata verkko- ja mobiilialustojen lisäksi myös PC:llä ja erilaisilla pelikonsoleilla. [10]

Ammattitaito

Yleinen ohjelmistotestaus vaatii paljon enemmän ammattitaitoa kuin pelitestaus. Vaikka yleinen ohjelmistotestaus painottuu automaattiseen testaukseen, sitä voivat tehdä vain ammattilaiset, joilla on paljon tietoa teknisistä asioista ja riittävä määrä ohjelmistotestauksessa tarvittavaa osaamista. Pelitestauksessa ei ole tällaisia ammattitaitoon liittyviä vaatimuksia. [10]

3.2 Pelitestauksen automatisointi ja sen haasteet

Videopelejä testataan tällä hetkellä pääasiassa manuaalisesti. [7] Automaattista testausta pitäisi hyödyntää enemmän, koska se mahdollistaisi tiettyjen testien suorittamisen monta kertaa peräkkäin ilman ihmistestaajan osallistumista, mikä säästäisi aikaa ja tekisi harvinaisten bugien löytymisestä todennäköisempää. [9] Pelitestauksen automatisointi on kuitenkin vaikeaa, joten pelinkehittäjät palkkaavat erikoistuneita ihmistestaajia testaamaan videopelejä manuaalisesti. Tämä johtaa siihen, että videopelialalla luotetaan pelien testaamiseen liittyvissä asioissa lähes yksinomaan manuaaliseen työhön ja ihmisten ammattitaitoon. [7]

Nykyiset videopelien testausprosessit jäävät vajaiksi automaation puutteen takia. Testauksen automatisointi olisi luonnollinen seuraava askel kohti tilannetta, jossa videopelien laatu paranee, mutta kustannukset eivät nouse. Pelinkehittäjillä ja tutkijoilla ei kuitenkaan tällä hetkellä ole prosesseja, rakenteita ja työkaluja, jotka auttaisivat testauksen automatisoinnissa. Tämä johtaa tiettyä tarkoitusta varten kehitettyihin tekniikoihin, joita on vaikea yleistää erilaisille peleille sopiviksi. [7]

Pelitestauksen automatisointi vaatii tutkimusta ja työkaluja, jotka voidaan integroida testausprosesseihin. Tutkimuksen ja sopivien työkalujen avulla voidaan päästä tilanteeseen, jossa pelinkehittäjillä on mahdollisuus tehdä videopeleistä systemaattisempia, toistettavampia ja luotettavampia, ja pelitestaajat voivat samaan aikaan keskittyä pelaajakeskeisiin näkökulmiin, kuten pelattavuuteen ja estetiikkaan. Testaus on joka tapauksessa avain laadukkaisiin videopeleihin ja testauksen automatisointi tarjoaisi hyvän mahdollisuuden parantaa videopelien laatua. [7]

4 Automaattisen pelitestauksen työkalut

Erilaisia automaattisen pelitestauksen työkaluja ei ole tällä hetkellä tarjolla kovin suurta määrää, varsinkaan sellaisia työkaluja, jotka on tarkoitettu nimenomaan pelien testaamiseen sen sijaan että ne tarjoaisivat mahdollisuuden testata mitä tahansa ohjelmistoja. Tässä luvussa esitellään neljä vaihtoehtoa, joiden väliltä tässä tutkielmassa käytettävä työkalu valittiin. Tässä tutkielmassa käytettäväksi työkaluksi valikoitui AltTester, koska se on varmuudella yhteensopiva testattavan pelin toteuttamisessa käytetyn Unity-pelimoottorin kanssa ja se oli valintahetkellä helposti saatavilla Unity Asset Storesta nimellä AltUnity Tester. Sen nimi muuttui valintahetken ja varsinaisen tutkimuksen tekemisen välissä, jolloin sen uudeksi nimeksi tuli AltTester, ja samassa yhteydessä se poistui Unity Asset Storesta. Se on kuitenkin edelleen täysin ilmainen työkalu, kun taas osa muista vaihtoehdoista on ilmaista kokeilujaksoa lukuun ottamatta maksullisia.

4.1 AltTester

AltTester on avoimen lähdekoodin automaattisen pelitestauksen työkalu, joka auttaa löytämään pelin objekteja ja vuorovaikuttamaan näiden objektien kanssa käyttäen joko C#-, Python- tai Java-ohjelmointikielellä kirjoitettuja testejä. Nämä tes-

tit voidaan suorittaa joko jollakin laitteella, jolla peliä on tarkoitus pelata, kuten tietokoneella tai älypuhelimella, tai Unity Editorin sisällä. [12]

AltTesterin tärkeimpiin ominaisuuksiin kuuluvat muun muassa pelin objektien löytäminen ja niihin liittyvien metodien ja ominaisuuksien käyttäminen ja muokkaaminen, minkä tahansa pelaajan antaman syötteen simuloiminen, testidatan manipuloiminen ja generoiminen, näyttökuvien ottaminen pelistä, peliin liittyvien mittausten tekeminen, C#-, Python- tai Java-ohjelmointikielellä kirjoitettujen testien suorittaminen, syötteen tuottamiseksi suoritettujen toimintojen visualisoiminen ja testitulosten ja raporttien tarkasteleminen. Objektien ominaisuuksia voivat olla esimerkiksi koordinaatit, teksti, arvot ja Unityn komponentit. [12]

4.2 Muita työkaluja

AltTesterin lisäksi on olemassa myös joitakin muita automaattisen pelitestauksen työkaluja. Tässä alaluvussa esitellään kolme työkalua, jotka olivat vaihtoehtoina tässä tutkielmassa käytettävää työkalua valittaessa.

GameDriver

GameDriver on automaattisen pelitestauksen työkalu, jonka avulla voidaan löytää ja manipuloida pelin objekteja, simuloida syötteitä ja vuorovaikuttaa pelimaailman kanssa. Testien tulokset voidaan nähdä heti testien suorittamisen jälkeen ja testejä voidaan parannella ja päivittää. Kuten AltUnity Testerin testit, myös GameDriverin testit voidaan suorittaa joko jollakin laitteella, jolla peliä on tarkoitus pelata, kuten tietokoneella tai älypuhelimella, tai Unity Editorin sisällä. [13]

Appium

Appium on avoimen lähdekoodin automaattisen testauksen työkalu, jolla voidaan testata mobiili- ja PC-sovelluksia, mukaan lukien pelejä. Se on järjestelmäriippumaton työkalu, mikä tarkoittaa sitä, että testejä voidaan suorittaa monella erilaisella alustalla käyttämällä samaa ohjelmointirajapintaa. Tämä mahdollistaa koodin uudelleenkäyttämisen erilaisilla alustoilla suoritettavien testien välillä. [14]

T-Plan Robot

T-Plan Robot on automaattisen testauksen työkalu, jolla voidaan Appiumin tavoin testata mobiili- ja PC-sovelluksia, mukaan lukien pelejä. T-Plan Robot järjestelmäriippumaton työkalu, jonka avulla voidaan suorittaa automaattisesti samoja toimintoja kuin mitä ihminen suorittaisi manuaalisesti. Tämä helpottaa huomattavasti monelle eri alustalle tarkoitettujen sovellusten testaamista. T-Plan Robot ei vaadi minkäänlaista sisäistä muokkausta testattavaan sovellukseen. T-Plan Robotin avulla voidaan tallentaa testejä ja suorittaa niitä yhdellä tai useammalla alustalla. [15]

5 Testiasetelman esittely

Tässä tutkielmassa testataan yksinkertaista Unity-pelimoottorilla ja C#-ohjelmointikielellä toteutettua videopeliä, johon on injektoitu tarkoituksellisesti bugeja, jotta sen testaaminen tuottaisi riittävästi tuloksia automaattisen ja manuaalisen testauksen vertailua varten. Tässä luvussa esitellään ensin itse peli ja sen jälkeen peliin injektoidut bugit.

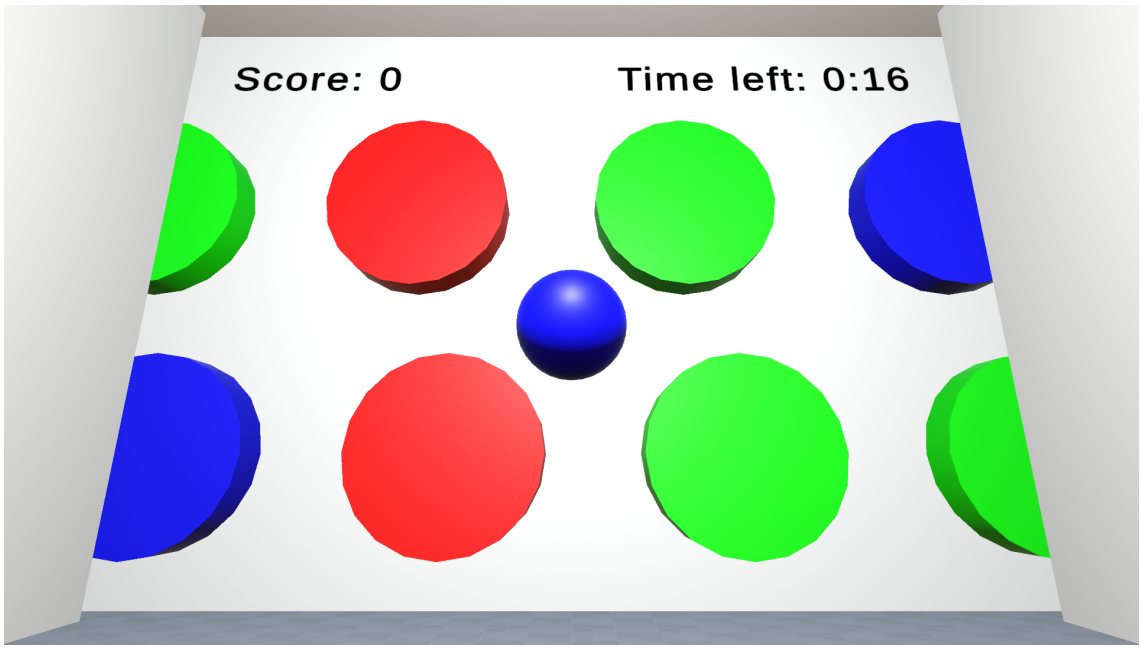
5.1 Pelin esittely

Peli on nimeltään Precision Color Throwing. Pelin perusidea on se, että pelaaja heittää palloja ja yrittää jokaisella heitolla osua oikeanväriseen maalitauluun. Oikealla värillä tarkoitetaan sitä, että maalitaulu on samanvärinen kuin pelaajan heittämä pallo. Jos pallo osuu maalitauluun, pelaaja joko saa tai menettää pisteitä, tai toisin sanoen pelaaja saa joko plus- tai miinuspisteitä, riippuen siitä ovatko pallo ja maalitaulu samanväriset vai eriväriset. Pelaaja kerää siis itselleen pisteitä heittämällä palloja niin, että ne osuvat oikeanväriseen maalitauluun.

Pelissä on punaisia, sinisiä ja vihreitä palloja sekä punaisia, sinisiä ja vihreitä lieriön muotoisia maalitauluja. Jokaisen pallon ja maalitaulun väri arvotaan satunnaisesti kolmen edellä mainitun värin väliltä. Maalitaulut liikkuvat hitaasti pelialueen poikki muodostaen kaksi riviä. Ylemmässä rivissä olevat maalitaulut liikkuvat vasemmalta oikealle, kun taas alemmassa rivissä olevat maalitaulut liikkuvat oikealta vasemmalle. Pelialueella on pelin ensimmäisiä sekunteja lukuun ottamatta koko ajan

yhtäaikaisesti vähintään neljä ja enintään kuusi kokonaan näkyvissä olevaa maalitaulua, joten pelissä on teoreettisesti mahdollisuus siihen että pelialueella on koko ajan vähintään yksi punainen, yksi sininen ja yksi vihreä maalitaulu.

Pelinäkymän oikeassa yläkulmassa on kello, joka määrittää pelin kokonaiskeston. Kellossa on pelin alkaessa 1 minuutti aikaa, ja peli päättyy kun kellosta loppuu aika. Pelaajan keräämien pisteiden määrä näkyy pelin aikana pelinäkymän vasemmassa yläkulmassa, ja 10 sekuntia pelin päättymisen jälkeen, kun pelialueella ei ole enää maalitauluja, lopullinen pistemäärä tulee näkyviin keskelle pelinäkymää. Pelin käyttöliittymä näkyy kuvassa 5.1.



Kuva 5.1: Pelin käyttöliittymä ja näkymä

Pelaajalla on käytössään yksi pallo kerrallaan. Pelaajan tavoitteena on heittää pallo niin, että se osuu oikeanväriseen maalitauluun. Tarkemmin sanottuna pelaajan tavoitteena on saada pallo osumaan lieriön muotoisen maalitaulun ympyrän muotoiseen etuosaan, koska kyseessä on 3D-peli. Kun pelaaja on heittänyt pallon, hän saa käyttöönsä uuden pallon, jos kellossa on vielä aikaa jäljellä.

Jos pelaajan heittämä pallo osuu oikeanväriseen maalitauluun, pelaaja saa pisteitä ja kelloon lisätään 1–5 sekuntia aikaa. Pelaajan saamien pisteiden ja kelloon lisättävän ajan määrä riippuu siitä, kuinka lähellä maalitaulun keskustaa osumakohta on. Mitä lähempänä maalitaulun keskustaa osumakohta on, sitä enemmän pisteitä pelaaja saa ja sitä enemmän sekunteja kelloon lisätään.

Jos pelaajan heittämä pallo osuu vääränväriseen maalitauluun, pelaaja menettää pisteitä. Pelaajan menettämien pisteiden määrä pallon osuessa vääränväriseen maalitauluun lasketaan samalla tavalla kuin pelaajan saamien pisteiden määrä pallon osuessa oikeanväriseen maalitauluun. Mitä lähempänä maalitaulun keskustaa osumakohta on, sitä enemmän pisteitä pelaaja menettää.

Jos pelaajan heittämä pallo ei osu mihinkään maalitauluun, pelaaja ei saa eikä menetä pisteitä. Pelaajaa ei missään tilanteessa rangaista epäonnistuneesta heitosta ottamalla kellosta sekunteja pois, vaan pisteiden menettäminen pallon osuessa vääränväriseen maalitauluun on ainoa mahdollinen rangaistus. Peli kestää siis vähintään minuutin. Pelin kokonaiskesto on enemmän kuin 1 minuutti, jos pelaaja osuu pelin aikana vähintään yhden kerran oikeanväriseen maalitauluun, mutta se ei voi missään tilanteessa olla vähemmän kuin 1 minuutti.

Pelin pelaamiseen tarvitaan nuolinäppäimiä ja välilyöntinäppäintä. Nuolinäppäimiä käytetään heiton suuntaamiseen. Pelaaja voi siirtää heiton tähtäystä ylös, alas, vasemmalle ja oikealle. Kun pelaaja on suunnannut heiton haluamaansa suuntaan ja haluaa heittää pallon, hän painaa välilyöntinäppäimen pohjaan. Mitä pidemmän ajan pelaaja pitää välilyöntinäppäintä pohjassa, sitä enemmän heitossa tulee olemaan voimaa. Heitossa olevan voiman määrällä on kuitenkin yläraja, jonka jälkeen heittoon ei enää tule lisää voimaa, vaikka pelaaja pitäisi edelleen välilyöntinäppäintä pohjassa. Tätä ylärajaa vastaava lukuarvo on 100. Pallon heittäminen tapahtuu niin, että kun pelaaja on pitänyt välilyöntinäppäintä pohjassa niin kauan että heitos-

sa on hänen haluamansa määrä voimaa, hän vapauttaa välilyöntinäppäimen, jolloin pallo lentää pelaajan määrittämällä voimalla pelaajan määrittämään suuntaan.

Välilyöntinäppäintä käytetään myös siirtymiseen pelin eri näkymien välillä ennen pelin alkamista ja pelin päättymisen jälkeen. Pelin aloitusnäkymässä on pelin nimi ja kehotus siirtyä eteenpäin painamalla välilyöntinäppäintä. Seuraavissa neljässä näkymässä on ohjeita pelin pelaamiseen, joten pelaajan täytyy painaa välilyöntinäppäintä neljä kertaa päästäkseen lukemaan kaikki pelin ohjeet. Välilyöntinäppäimen painaminen viidennen kerran käynnistää 5 sekunnin valmistautumisajan, jonka jälkeen peli alkaa. Pelin päättyminen käynnistää 10 sekunnin odotusajan, jonka aikana pelin päättymishetkellä näkyvissä olevat maalitaulut liikkuvat pelialueen ulkopuolelle, ja tämän odotusajan päätyttyä pelaaja voi välilyöntinäppäintä painamalla siirtyä takaisin pelin aloitusnäkymään.

Pelin tarkoituksena on kerätä mahdollisimman paljon pisteitä. Suurin mahdollinen pistemäärä, jonka pelaaja voi saada tai menettää yhdellä heitolla pallon osuessa maalitauluun on 2000, ja vastaavasti pienin mahdollinen pistemäärä on 1. Pelin alkaessa pelaajalla on 0 pistettä. Tämä ei kuitenkaan ole pienin mahdollinen kokonaispistemäärä, vaan kokonaispistemäärä voi olla negatiivinen, jos pelaaja osuu yhteen tai useampaan vääränväriseen maalitauluun. Pelin tarkoituksena on kuitenkin pyrkiä mahdollisimman suureen positiiviseen kokonaispistemäärään.

5.2 Injektoidut bugit

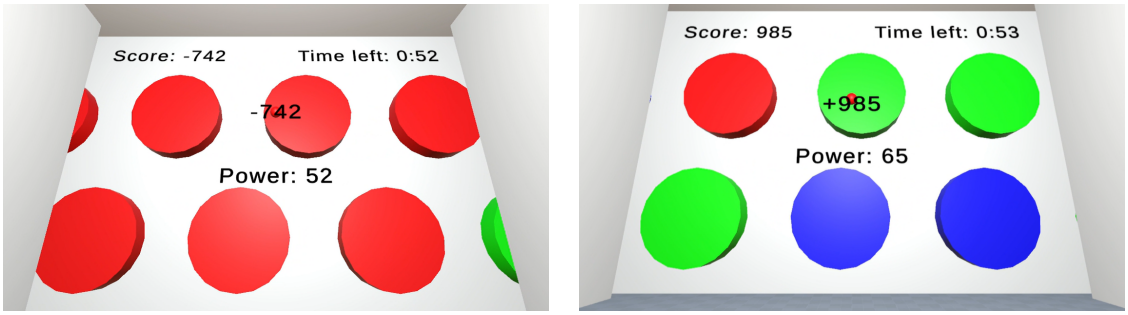
Peliin injektoitiin tarkoituksellisesti neljä erilaista bugia, jotka esitellään tässä alaluvussa. Kolme näistä neljästä bugista liittyi tietynväriseen palloon, joten jokainen väri sai oman buginsa. Neljäs bugi liittyi pelin päättymiseen. Taulukko 5.1 sisältää bugit ja niiden aiheet sekä lyhyen kuvauksen jokaisesta bugista.

Taulukko 5.1: Bugien aiheet ja lyhyet kuvaukset

Bugi	Aihe	Lyhyt kuvaus
Bugi 1	Punainen pallo	Värit toimivat väärin
Bugi 2	Sininen pallo	Osumakohdan etäisyys maalitaulun keskustasta toimii käänteisesti
Bugi 3	Vihreä pallo	Pelaaja saa pisteitä vain osumalla tiettyyn osaan maalitaulusta
Bugi 4	Pelin päättyminen	Peli ei pääty jos kokonaispistemäärä on 0

Bugi 1

Ensimmäinen tarkoituksellisesti injektoitu bugi on sellainen, että punaista palloa heitettäessä värit toimivat päinvastoin kuin niiden pitäisi toimia. Tämä tarkoittaa sitä että punaisen pallon osuessa punaiseen maalitauluun pelaaja menettää pisteitä, ja punaisen pallon osuessa siniseen tai vihreään maalitauluun pelaaja saa pisteitä ja kelloon lisätään sekunteja. Kuvassa 5.2 visualisoidaan tätä bugia.



(a) Bugi 1, pelaaja saa miinuspisteitä.

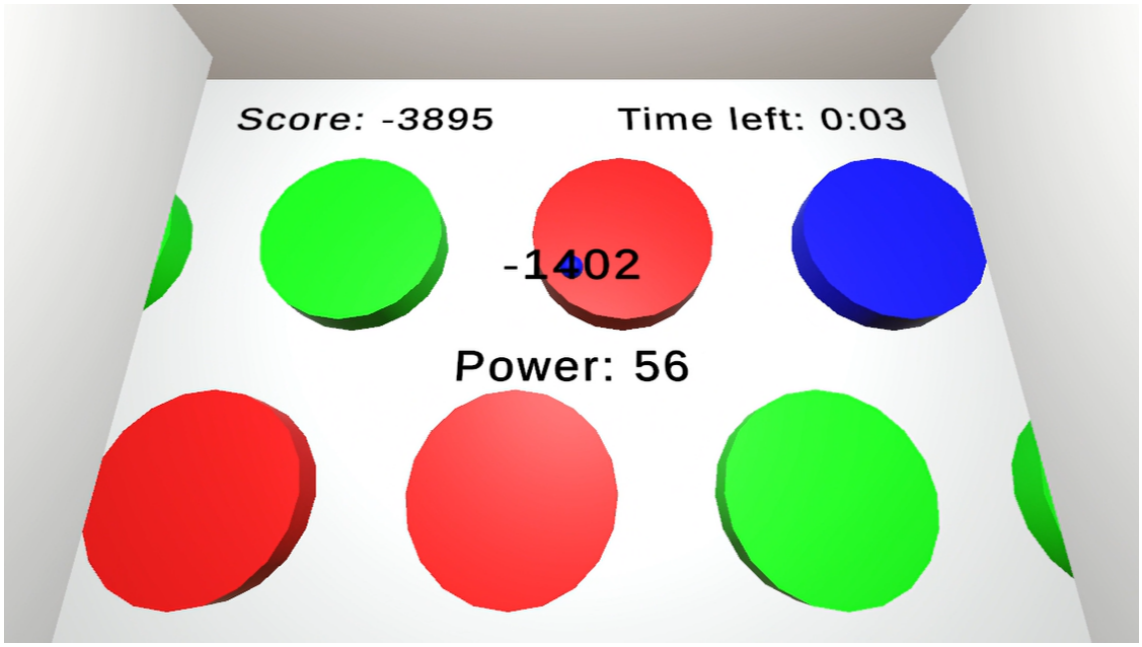
(b) Bugi 1, pelaaja saa pluspisteitä.

Kuva 5.2: Bugi 1, miinus- ja pluspistetilanteet

Yllä olevan kuvan osassa (a) pelaaja saa miinuspisteitä punaisen pallon osuessa punaiseen maalitauluun, ja osassa (b) pelaaja saa pluspisteitä punaisen pallon osuessa vihreään maalitauluun. Pelin toimiessa oikein pelaaja saisi osan (a) tilanteessa pluspisteitä, koska pallo osuu oikeanväriseen maalitauluun, ja osan (b) tilanteessa miinuspisteitä, koska pallo osuu vääränväriseen maalitauluun.

Bugi 2

Toinen tarkoituksellisesti injektoitu bugi on sellainen, että sinistä palloa heitettäessä osumakohdan etäisyys maalitaulun keskustasta toimii päinvastoin kuin sen pitäisi toimia. Tämä tarkoittaa sitä että sinisen pallon osuessa maalitauluun pelaaja saa tai menettää sitä enemmän pisteitä, mitä kauempana maalitaulun keskustasta osumakohta on, ja lisäksi sinisen maalitaulun tapauksessa kelloon lisätään sitä enemmän sekunteja, mitä kauempana maalitaulun keskustasta osumakohta on. Tätä bugia visualisoidaan kuvassa 5.3.

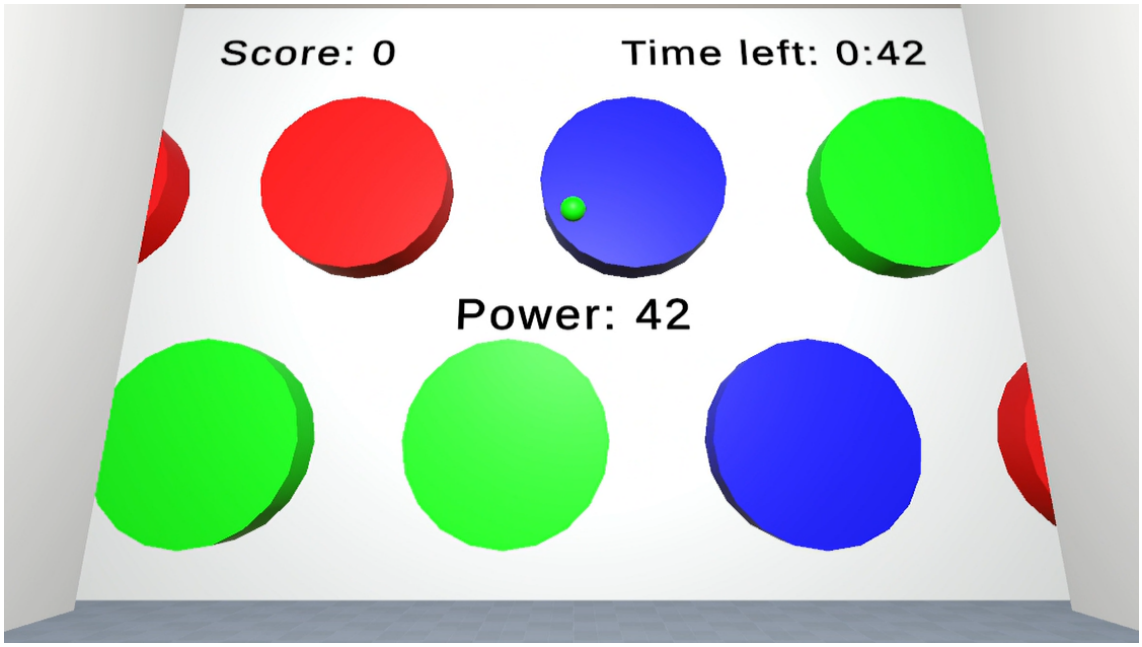


Kuva 5.3: Bugi 2, pelaaja saa miinuspisteitä

Yllä olevassa kuvassa pelaaja saa yli 1000 miinuspistettä sinisen pallon osuessa punaiseen maalitauluun, vaikka osumakohta on lähempänä maalitaulun reunaa kuin sen keskustaa. Pelaaja saisi kuvan tilanteessa miinuspisteitä myös pelin toimiessa oikein, mutta pelaajan saamien pisteiden määrä olisi osumakohdan perusteella pienempi kuin 1000, jos se laskettaisiin oikealla tavalla.

Bugi 3

Kolmas tarkoituksellisesti injektoitu bugi on sellainen, että vihreän pallon osuessa maalitauluun pelaaja saa tai menettää pisteitä vain, jos pallo osuu tiettyyn osaan maalitaulusta. Tarkemmin sanottuna osumakohdan täytyy olla lähempänä maalitaulun keskustaa kuin sen reunaa, tai toisin sanoen osumakohdan etäisyyden maalitaulun keskustasta täytyy olla alle puolet maalitaulun säteestä, jotta pelaaja saisi tai menettäisi pisteitä. Lisäksi vihreän maalitaulun tapauksessa kelloon lisätään sekunteja vain, jos pallo osuu edellä määriteltyyn osaan maalitaulusta. Kuvassa 5.4 visualisoidaan tilannetta, jossa pelaaja ei saa tämän bugin takia pisteitä.



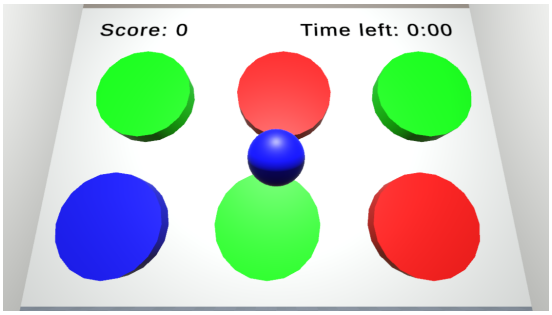
Kuva 5.4: Bugi 3, pelaaja ei saa miinuspisteitä

Yllä olevassa kuvassa pelaaja ei saa miinuspisteitä vihreän pallon osuessa siniseen maalitauluun, kun osumakohta on lähempänä maalitaulun reunaa kuin sen keskustaa. Pelin toimiessa oikein pallon ei tarvitsisi osua tiettyyn osaan maalitaulusta, vaan pallon osuminen maalitauluun tuottaisi pelaajalle joko plus- tai miinuspisteitä

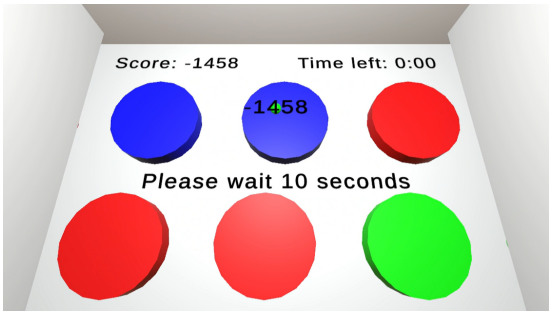
osumakohdasta riippumatta, ja näin ollen pelaaja saisi kuvan tilanteessa miinuspisteitä, koska pallo osuu vääränväriseen maalitauluun.

Bugi 4

Neljäs tarkoituksellisesti injektoitu bugi on sellainen, että peli ei pääty ajan loppuessa kellosta, jos kokonaispistemäärä on 0. Peli siis jatkuu niin kauan kuin pelaaja ei ole osunut kertaakaan maalitauluun eikä ole näin ollen saanut tai menettänyt yhtään pistettä, vaikka kellossa ei ole enää aikaa jäljellä. Kokonaispistemäärä voi teoriassa olla 0 myös siinä tapauksessa että pelaaja on osunut pelin aikana maalitauluihin ja saanut täsmälleen saman määrän plus- ja miinuspisteitä, mutta käytännössä tällaiseen tilanteeseen päätyminen on kuitenkin erittäin epätodennäköistä. Tätä bugia visualisoidaan kuvassa 5.5 kahden erilaisen pelin päättymiseen liittyvän tilanteen kautta. Ensimmäisessä tilanteessa peli on käynnissä vaikka sen pitäisi olla päättynyt, ja toisessa tilanteessa peli päättyy.



(a) Bugi 4, peli on käynnissä.



(b) Bugi 4, peli päättyy.

Kuva 5.5: Bugi 4, pelin päättymiseen liittyvät tilanteet

Yllä olevan kuvan osassa (a) peli on käynnissä, ja osassa (b) pelaaja saa miinuspisteitä vihreän pallon osuessa siniseen maalitauluun, vaikka kellossa ei ole kummassakaan tilanteessa aikaa jäljellä. Osan (a) tilanteessa kokonaispistemäärä on 0, ja osan (b) tilannetta edeltävän kokonaispistemäärän täytyy olla 0, koska kuvassa näkyvä kokonaispistemäärä on sama kuin pelaajan yksittäisellä heitolla saamien

miinuspisteiden määrä. Peli päättyy osan (b) tilanteessa samalla hetkellä kun pallo osuu maalitauluun. Jos pelaaja saisi osan (b) tilanteessa pluspisteitä, peli ei päättyisi välittömästi pallon osuessa maalitauluun, vaan kelloon lisättäisiin vielä 1–5 sekuntia aikaa. Pelin toimiessa oikein pelaaja ei saisi osan (b) tilanteessa miinuspisteitä pallon osuessa maalitauluun, koska peli olisi päättynyt joko juuri ennen heittoa tai heiton aikana ennen pallon osumista maalitauluun. Näin ollen lopullinen pistemäärä olisi 0. Jos pelin päättymisestä olisi kulunut yli 5 sekuntia, pelaaja ei enää pystyisi edes heittämään palloa, koska pelialueella ei enää olisi yhtään palloa.

6 Pelin testaaminen

Luvussa 5 esiteltyä peliä testattiin ensin manuaalisesti ja sen jälkeen automaattisesti. Sekä manuaalinen että automaattinen testaus koostuivat viidestä erilaisesta testitapauksesta. Manuaalista testausta varten rekrytoitiin koehenkilöitä testaamaan peliä ja automaattisessa testauksessa peliä testattiin luvussa 4 esitellyn automaattisen pelitestauksen työkalun AltTesterin avulla. Tässä luvussa esitellään ensin testitapaukset ja sen jälkeen käydään läpi testausprosessi eli kerrotaan miten manuaalinen ja automaattinen testaus käytännössä tapahtuivat.

6.1 Testitapaukset

Testausprosessi koostui viidestä erilaisesta testitapauksesta, joista neljä ensimmäistä rakentui luvussa 5 esiteltyjen tarkoituksellisesti injektoitujen bugien ympärille. Jokaisessa näistä neljästä testitapauksesta oli tavoitteena löytää yksi neljästä tarkoituksellisesti injektoidusta bugista, ja näin ollen näiden neljän testitapauksen numerointi vastaa tarkoituksellisesti injektoitujen bugien numerointia. Viides testitapaus ei rakentunut minkään bugin ympärille, vaan siinä sekä manuaaliset testaajat että tietokone saivat testata peliä vapaasti. Taulukko 6.1 kertoo mitä asiaa missäkin testitapauksessa testattiin ja mikä bugi liittyi mihinkin testitapaukseen.

Taulukko 6.1: Testitapaukset ja niihin liittyvät bugit

Testitapaus	Testattava asia	Bugi
Testitapaus 1	Punainen pallo	Bugi 1
Testitapaus 2	Sininen pallo	Bugi 2
Testitapaus 3	Vihreä pallo	Bugi 3
Testitapaus 4	Pelin päättyminen	Bugi 4
Testitapaus 5	Avoin testaus	-

Testitapaus 1

Testitapauksessa 1 testattiin punaista palloa, koska bugi 1 liittyi punaiseen palloon. Tavoitteena oli saada punainen pallo osumaan vähintään kerran punaiseen maalitauluun ja vähintään kerran siniseen tai vihreään maalitauluun, jotta bugi 1 tulisi esille ja näin ollen voitaisiin todeta että värit toimivat päinvastoin kuin niiden pitäisi toimia sekä punaisen pallon osuessa oikeanväriseen maalitauluun että punaisen pallon osuessa vääränväriseen maalitauluun.

Testitapaus 2

Testitapauksessa 2 testattiin sinistä palloa, koska bugi 2 liittyi siniseen palloon. Tavoitteena oli saada sininen pallo osumaan vähintään kerran siniseen maalitauluun ja vähintään kerran punaiseen tai vihreään maalitauluun, jotta bugi 2 tulisi esille ja näin ollen voitaisiin todeta että osumakohdan etäisyys maalitaulun keskustasta toimii päinvastoin kuin sen pitäisi toimia sekä pelaajan saadessa pisteitä että pelaajan menettäessä pisteitä.

Testitapaus 3

Testitapauksessa 3 testattiin vihreää palloa, koska bugi 3 liittyi vihreään palloon. Tavoitteena oli saada vihreä pallo osumaan vähintään kerran vihreään maalitauluun ja vähintään kerran punaiseen tai siniseen maalitauluun, jotta bugi 3 tulisi esille ja näin ollen voitaisiin todeta että vihreän pallon täytyy osua tiettyyn osaan

sekä oikeanvärisestä vääränvärisestä maalitaulusta, jotta pelaaja saisi tai menettäisi pisteitä ja kelloon lisättäisiin aikaa.

Testitapaus 4

Testitapauksessa 4 testattiin pelin päättymistä. Tavoitteena oli saada kokonaispistemäärä pysymään nollassa siihen asti että kellosta loppuu aika, jotta bugi 4 tulisi esille ja näin ollen voitaisiin todeta että peli ei pääty ajan loppuessa kellosta ja mahdollisesti saada selville miten peli saadaan siinä tilanteessa päättymään.

Testitapaus 5

Testitapaus 5 oli nimeltään "Vapaa/Avoin testaus", mikä tarkoitti sitä että ei testattu mitään tiettyä asiaa eikä pyritty löytämään mitään tiettyä bugia, vaan peliä testattiin vapaasti. Tavoitteena oli vahvistaa aiempien testitapausten aikana tehtyjä havaintoja ja mahdollisesti havaita jotain sellaista mikä jäi aiempien testitapausten aikana havaitsematta.

6.2 Manuaalinen testaus

Manuaalisen testauksen testitilanne aloitettiin kertomalla koehenkilölle, että tässä tutkimuksessa on tarkoitus testata yksinkertaista videopeliä. Tämän jälkeen koehenkilölle annettiin paperilappu, johon oli tulostettu ohjeet pelin testaamista varten, ja koehenkilöä pyydettiin testaamaan peliä näiden ohjeiden mukaan. Koehenkilölle ei kerrottu ennen testausta, että peliin oli injektoitu tarkoituksellisesti bugeja. Koehenkilön toimintaa tarkkailtiin testisession aikana. Koehenkilöä pyydettiin puhumaan ajatuksiaan ääneen testisession aikana, mutta koehenkilön kysymyksiin ja pohdintoihin vastattiin vain silloin kun vastaaminen tuntui välttämättömältä. Testisessio päättyi siihen että koehenkilölle annettiin palautelomake, jossa koehenkilöltä ky-

syttiin toimiko peli testisession aikana niin kuin sen oli tarkoitus toimia, ja jos ei toiminut niin millä tavalla se toimi väärin.

Koehenkilöitä oli yhteensä neljä. Testisessiot suoritettiin kahden viikon aikana, jolloin oli mahdollista käyttää testisessioiden suorittamispaikaksi valittua tilaa. Ennen tätä kahden viikon jaksoa ja sen aikana mainostettiin mahdollisuutta osallistua tutkimukseen. Neljä koehenkilöä oli riittävä määrä tätä tutkimusta varten, mutta se ei missään nimessä ollut maksimimäärä, vaan koehenkilöitä olisi voinut olla enemmänkin. Tutkimukseen ei kuitenkaan saatu edellä mainitun kahden viikon jakson puitteissa osallistumaan enempää koehenkilöitä.

Kaikki tutkimukseen osallistuneet koehenkilöt olivat IT-alan opiskelijoita. Kaksi koehenkilöä rekrytoitiin käytettävyyskurssilta ja kaksi koehenkilöä ohjelmistotekniikan kurssilta. Neljästä koehenkilöstä kaksi oli suomalaisia, kun taas kaksi koehenkilöä ei puhunut suomea, joten koehenkilölle annetuista testausohjeista ja palautelomakkeesta oli tarpeellista tehdä suomenkielisen version lisäksi myös englanninkielinen versio. Itse peli toteutettiin jo alun perin englanninkielisenä, joten pelin ohjeita ja muuta pelissä näkyvää tekstiä ei tarvinnut kääntää suomesta englanniksi. Testausohjeiden suomenkielinen versio on liitteessä A.

Koehenkilölle annetut testausohjeet alkoivat yleisillä ohjeilla, joissa koehenkilöä pyydettiin aluksi lukemaan pelin ohjeet ja jatkamaan sen jälkeen testausohjeiden lukemista ennen pelin aloittamista. Tämän jälkeen yleisissä ohjeissa kerrottiin, että jos peli antaa erivärisen pallon kuin mitä testitapauksessa pyydetään heittämään, koehenkilön pitäisi pyrkiä heittämään pallo niin, että se ei osu mihinkään maalitauluun. Yleisten ohjeiden kolmannessa ja viimeisessä kohdassa koehenkilöä ohjeistettiin aloittamaan peli uudestaan ja jatkamaan testitapauksen suorittamista siitä mihin hän jäi edellisen pelin päättyessä, jos hän ei ehdi suorittaa testitapausta loppuun asti ennen kuin kellosta loppuu aika.

Yleisten ohjeiden jälkeen koehenkilölle annettiin ohjeet testitapausten suorittamiseen. Jokaisen testitapauksen kohdalla kerrottiin ensin mitä koehenkilön pitäisi kyseisessä testitapauksessa tehdä, ja sen jälkeen testitapausta 5 lukuun ottamatta jokaisen testitapauksen kohdalla kerrottiin odotettu toiminta eli se, miten pelin on tarkoitus toimia ilman bugeja koehenkilöiden toimiessa ohjeiden mukaan. Koska koehenkilöille ei kerrottu tarkoituksellisesti injektoiduista bugeista ennen testausta, he eivät tienneet etukäteen että pelin toiminta testisession aikana ei vastannut heille kerrottua odotettua toimintaa.

Testitapauksessa 1 koehenkilöä pyydettiin heittämään niin monta punaista palloa kuin hän ehtii heittää ennen kuin kellosta loppuu aika ja pyrkimään siihen että punainen pallo osuu vähintään kerran punaiseen maalitauluun ja vähintään kerran siniseen tai vihreään maalitauluun. Odotetuksi toiminnaksi kerrottiin, että punaisen pallon osuessa maalitauluun pelaaja saa tai menettää pisteitä ja kelloon lisätään aikaa luvussa 5 kuvatulla tavalla.

Testitapauksissa 2 ja 3 koehenkilölle annettiin muuten samanlaiset ohjeet kuin testitapauksessa 1, mutta pallojen ja maalitaulujen värit vaihtuivat sen mukaan minkäväristä palloa missäkin testitapauksessa testattiin. Myös odotettu toiminta oli testitapauksissa 2 ja 3 samanlainen kuin testitapauksessa 1, koska ilman bugeja pelin on tarkoitus toimia kaikilla väreillä samalla tavalla.

Testitapauksessa 4 koehenkilöä pyydettiin olemaan heittämättä palloja ja odottamaan niin kauan, että kellosta loppuu aika. Odotetuksi toiminnaksi kerrottiin, että peli toimii samalla tavalla kuin se toimi testitapauksissa 1–3 ajan loppuessa kellosta, eli peliin ei enää ilmesty uusia maalitauluja, pelaaja ei voi enää saada lisää pisteitä eikä sekunteja kelloon, pelialueella olevat pallot katoavat ja pelin päättymiseen liittyvät tekstit tulevat näkyviin.

Testitapauksessa 5 koehenkilö sai testata peliä vapaasti haluamallaan tavalla. Koehenkilöä pyydettiin siirtymään testitapauksesta 4 testitapaukseen 5 heittämällä

minkä tahansa värinen pallo niin, että se osuu mihin tahansa maalitauluun. Tämä johtui testitapaukseen 4 liittyvästä tarkoituksellisesti injektoidusta bugista, jonka takia koehenkilön piti saada testitapauksen 4 päätteeksi kokonaispistemääräksi jotain muuta kuin 0, jotta hän pääsi siirtymään seuraavaan testitapaukseen. Muita ohjeita testitapauksen 5 suorittamiseen ei annettu.

6.3 Automaattinen testaus

Automaattinen testaus aloitettiin kirjoittamalla AltTesterin avulla jokaista manuaalisen testauksen testitapausta mahdollisimman hyvin vastaava ohjelmakoodi. Nämä ohjelmakoodit kirjoitettiin itse pelin tavoin C#-ohjelmointikielellä. Kun jokaiseen testitapaukseen tarvittava ohjelmakoodi oli kirjoitettu, peliä testattiin näiden ohjelmakoodien avulla. Testitapausten aikana tehtiin havaintoja siitä mitä pelissä tapahtui ja mitä ei tapahtunut, ja jokaisen testitapauksen jälkeen kirjoitettiin muistiin kyseisen testitapauksen aikana tehdyt havainnot.

Testitapauksia 4 ja 5 vastaavan ohjelmakoodin kirjoittaminen onnistui suhteellisen helposti, mutta testitapauksia 1–3 vastaavan ohjelmakoodin kirjoittaminen osoittautui haastavaksi tehtäväksi, koska maalitaulut liikkuvat ja niiden värit määräytyvät satunnaisesti. Ideaalissa tilanteessa testitapaukset 1–3 olisivat olleet automaattisessa testauksessa sellaisia, että tietokone etsii heitettävän pallon kanssa samanvärisen maalitaulun ja heittää pallon niin, että se osuu löydettyyn maalitauluun. Tällaisen testitapauksen toteuttaminen on todennäköisesti mahdollista, mutta se osoittautui sen verran haastavaksi tehtäväksi että siihen ei ollut tässä tutkimuksessa järkevää käyttää aikaa. Jokaiselle värille toteutettiin kuitenkin oma testitapaus, koska ei ollut vaikeaa ohjelmoida sellaista testitapausta, jossa tietokone ei heitä koko ajan samanlaisia heittoja vaan heitettävän pallon väri määrittää sen, millaisen heiton tietokone heittää. Automaattisessa testauksessa käytetyt ohjelmakoodit ovat kokonaisuudessaan liitteessä B.

Ohjelmakoodit koostuivat pääasiassa komennoista joilla tietokonetta ohjeistettiin painamaan pelin pelaamiseen tarvittavia näppäimiä eli nuolinäppäimiä ja välilyöntinäppäintä ja pitämään niitä pohjassa tietyn ajan. Nämä komennot mahdollistivat pallojen heittämisen halutulla voimalla haluttuun suuntaan ja siirtymisen pelin eri näkymien välillä. Pallojen heittämiseen tarvittiin myös komentoa jolla tietokonetta ohjeistettiin etsimään pelissä olevat pallot, ja aina kun tietokone löysi pelialueelle ilmestyneen pallon jota se ei ollut vielä heittänyt, se heitti kyseisen pallon. Testitapauksissa 1-3 tietokonetta ohjeistettiin ennen heittoa selvittämään heitettävän pallon väri, koska tietynväristä palloa testattaessa ei ollut tarkoitus heittää kaikilla palloilla samanlaisia heittoja. Koska testitapauksessa 4 ei haluttu tietokoneen tekevän mitään ennen kuin kellosta loppuu aika, ja koska jokaiseen testitapaukseen sisältyi pelin alkamista edeltävä 5 sekunnin valmistautumisaika, ohjelmakoodeissa käytettiin myös komentoa jolla säädeltiin kahden peräkkäisen komennon suorittamisen välillä kuluvaa aikaa. Tätä komentoa käytettiin myös ohjelmakoodien lopussa, jotta testitapaukset eivät päättyisi välittömästi pelin päättymishetkellä, tai jopa ennen pelin päättymishetkeä jos testitapauksessa 4 lisättäisiin kelloon aikaa pallon osuessa maalitauluun.

Edellä mainittujen komentojen lisäksi tietokonetta ohjeistettiin seuraamaan testitapauksessa 4 kokonaispistemäärää ja muissa testitapauksissa kellossa jäljellä olevaa aikaa. Kokonaispistemäärän seuraaminen oli tarpeellista testitapauksessa 4, koska tietokoneen piti heittää palloja niin kauan kunnes kokonaispistemääräksi tuli jotain muuta kuin 0, kun taas muissa testitapauksissa kokonaispistemäärä ei vaikuttanut siihen mitä tietokoneen piti tehdä. Sen sijaan muissa testitapauksissa oli mahdollisuus siihen että kelloon lisätään sekunteja ennen kuin siitä loppuu aika, kun taas testitapauksessa 4 ei ollut tätä mahdollisuutta. Näin ollen muissa testitapauksissa oli tarpeellista seurata kellossa jäljellä olevaa aikaa, koska pelin alkamisesta ajan loppumiseen kuluva aika ei ollut tiedossa ennen testitapausten suorittamista,

kun taas testitapauksessa 4 tiedettiin tämän ajan olevan tasan 1 minuutti, joka voitiin asettaa pallojen heittämisen aloittavan komennon ja sitä edeltävän komennon suorittamisen väliseksi ajaksi.

Automaattisessa testauksessa käytettiin kahta erilaista heittoa, joista käytetään tässä tutkielmassa nimityksiä ”korkea heitto” ja ”matala heitto”. Korkea heitto tarkoittaa heittoa, jossa on riittävä määrä voimaa ja joka suuntautuu tarpeeksi ylös, jotta pallo ei putoa liian aikaisin maalitaulujen alapuolelle, vaan osuu maalitauluun, jos sellainen on vaakasuunnassa sopivassa kohdassa heiton lähtiessä liikkeelle. Matala heitto tarkoittaa heittoa, jossa on niin vähän voimaa ja joka suuntautuu niin alas, että pallo ei voi osua mihinkään maalitauluun.

Testitapauksessa 1 sääntönä oli se, että jos heitettävä pallo on punainen, tietokone tekee korkean heiton, ja jos heitettävä pallo on sininen tai vihreä, tietokone tekee matalan heiton. Näin ollen vain punaisilla palloilla oli mahdollisuus osua maalitauluun. Tietokone heitti niin monta palloa kuin se ehti heittää ennen kuin kellosta loppui aika. Testitapaukset 2 ja 3 olivat muuten samanlaisia kuin testitapaus 1, mutta punaisten pallojen sijaan tietokone teki korkeita heittoja testitapauksessa 2 sinisillä palloilla ja testitapauksessa 3 vihreillä palloilla.

Testitapauksissa 4 ja 5 heitettävien pallojen väreillä ei ollut merkitystä, joten kaikki tietokoneen tekemät heitot olivat korkeita heittoja. Testitapauksessa 4 tietokone ei tehnyt mitään ennen kuin kellosta loppui aika, minkä jälkeen se heitti palloja niin kauan kunnes joku heitetyistä palloista osui johonkin maalitauluun ja kokonaispistemääräksi tuli jotain muuta kuin 0. Testitapauksessa 5 tietokone heitti niin monta palloa kuin se ehti heittää ennen kuin kellosta loppui aika.

Testitapauksissa 1–4 kaikki heitot suuntautuivat suoraan eteenpäin, mutta testitapaukseen 5 sisällytettiin myös muihin suuntiin tähdättyjä heittoja. Testitapauksen 5 heitot jakautuivat niin, että joka kolmas heitto suuntautui suoraan eteenpäin ja muut heitot suuntautuivat vuorotellen pelialueen vasempaan ja oikeaan reunaan.

Tietynväristä palloa tai ajan loppumista testattaessa ei ollut varsinaista tarvetta eri suuntiin tähdätyille heitoille, mutta automaattisen testauksen avointa testausta edustavaan testitapaukseen sisällytettiin kuitenkin heittoja jotka eivät suuntaudu suoraan eteenpäin, koska myös manuaalisessa testauksessa oli sellaisia heittoja.

7 Tulokset

Luvussa 6 läpikäyty testausprosessi tuotti riittävästi tuloksia automaattisen ja manuaalisen testauksen vertailua varten. Tässä luvussa esitetään testausprosessin tulokset ja analysoidaan niitä sekä vertaillaan automaattisen ja manuaalisen testauksen tuloksia keskenään.

7.1 Manuaalisen testauksen tulokset

Tässä alaluvussa esitetään manuaalisen testauksen tulokset. Jokaisen koehenkilön löytämät tarkoituksellisesti injektoidut bugit näkyvät taulukossa 7.1.

Taulukko 7.1: Manuaalisessa testauksessa löydetyt bugit

	Bugi 1	Bugi 2	Bugi 3	Bugi 4
Koehenkilö 1	X	X	X	X
Koehenkilö 2	X		X	X
Koehenkilö 3	X		X	X
Koehenkilö 4	X		X	X

Kuten yllä olevasta taulukosta nähdään, manuaalisen testauksen kattavuus oli bugia 2 lukuun ottamatta hyvä. Koehenkilöiden välillä oli kuitenkin pieniä eroja bugien löytämisessä ja määrittelemisessä, ja näistä eroista kerrotaan seuraavaksi lisää manuaalisen testauksen tulosten tarkemmassa läpikäynnissä.

Testitapaus 1

Kaikki koehenkilöt huomasivat, että punainen pallo antaa pelaajalle miinuspisteitä osuessaan punaiseen maalitauluun ja pluspisteitä osuessaan siniseen tai vihreään maalitauluun. Tämä bugi oli tietynvärisiin palloihin liittyvistä bugeista selkeästi helpoin löytää ja määritellä, vaikka se tapahtui heti ensimmäisen testitapauksen aikana, jolloin testaajat pelasivat peliä ensimmäistä kertaa.

Testitapaus 2

Kukaan koehenkilöistä ei huomannut, että sinisen pallon osuessa maalitauluun pelaaja saa sitä enemmän plus- tai miinuspisteitä, mitä kauempana maalitaulun keskustasta osumakohta on. Yksi koehenkilö huomasi, että pelaaja ei saa täsmälleen oikeaa määrää plus- tai miinuspisteitä sinisen pallon osuessa maalitauluun, mutta hänkään ei osannut sanoa tarkemmin millä tavalla sininen pallo toimi väärin. Tämä bugi oli tietynvärisiin palloihin liittyvistä bugeista selkeästi vaikein löytää.

Testitapaus 3

Kaikki koehenkilöt huomasivat, että vihreän pallon osuessa maalitauluun pelaaja saa plus- tai miinuspisteitä vain joissakin tapauksissa. Kukaan koehenkilöistä ei kuitenkaan osannut sanoa, että vihreä pallo antaa plus- tai miinuspisteitä osuessaan lähelle maalitaulun keskustaa ja ei anna plus- eikä miinuspisteitä osuessaan kauas maalitaulun keskustasta. Tämän bugin löytäminen oli siis koehenkilöille helppoa, mutta sen tarkka määritteleminen oli vaikeaa.

Testitapaus 4

Kaikki koehenkilöt huomasivat, että peli ei päättynyt ajan loppuessa kellosta. Tämä asia tuli niin selkeästi esille, että sitä oli käytännössä mahdotonta olla huomaamatta,

varsinkin kun kyseessä ei ollut testisession ensimmäinen testitapaus, vaan koehenkilöt olivat jo nähneet mitä kolmessa aiemmassa testitapauksessa tapahtui, kun kellosta loppui aika. Kaikki koehenkilöt myös miettivät minkä takia peli ei päättynyt, mitä voidaan pitää positiivisena asiana.

Kaksi koehenkilöä päätyi täysin oikeaan johtopäätökseen eli siihen, että kokonaispistemäärän täytyy olla jotain muuta kuin 0, jotta peli päättyy. Yksi koehenkilö päätteli, että pelaajalla täytyy olla pluspisteitä, jotta peli päättyy. Tämä johtui todennäköisesti ainakin osittain siitä, että kyseinen koehenkilö sai testitapauksen 4 päätteeksi pluspisteitä, eikä näin ollen tullut ajatelleeksi, että peli voisi päättyä myös pelaajan saadessa miinuspisteitä, kun taas täysin oikeaan johtopäätökseen päätyneet koehenkilöt saivat testitapauksen 4 päätteeksi miinuspisteitä.

Yksi koehenkilö oli selvästi muita koehenkilöitä kauempana oikeasta johtopäätöksestä, koska hän päätteli että peli ei pääty, jos pelaaja ei ole heittänyt yhtään palloa. Tämä johtui todennäköisesti ainakin osittain siitä, että kyseinen koehenkilö osui heti ensimmäisellä heitolla maalitauluun ja sai pluspisteitä, joten hän ei päässyt näkemään että peli ei olisi päättynyt ensimmäisen heiton jälkeen jos hän ei olisi saanut plus- eikä miinuspisteitä. Kaksi kolmesta muusta koehenkilöistä pääsivät näkemään tämän tilanteen, koska he eivät saaneet ensimmäisellä heitolla plus- eivätkä miinuspisteitä, jolloin ensimmäinen heitto ei johtanut pelin päättymiseen.

Kolme neljästä koehenkilöstä siirtyi testitapauksesta 4 testitapaukseen 5 jo ennen testitapauksen 5 ohjeiden lukemista. Kaksi näistä kolmesta koehenkilöstä halusi vain kokeilla mitä tapahtuu, jos pelaaja heittää pallon kun kellosta on loppunut aika ja kokonaispistemäärä on 0, mutta yksi koehenkilö ei tyytynyt testaamaan tätä asiaa ilman mitään erityistä syytä. Kyseinen koehenkilö muisti testausohjeissa esitetyn väitteen, että pelaaja ei voi enää saada pisteitä eikä sekunteja kelloon, ja halusi testata pitääkö tämä väite paikkansa. Tätä voidaan pitää positiivisena asiana.

Testitapaus 5

Testisession päätteeksi suoritettu avoin testaus vahvisti todennäköisesti koehenkilöiden käsitystä siitä että pelissä oli jotain pielessä, mutta se ei juurikaan auttanut koehenkilöitä löytämään aiemmin löytämättä jääneitä bugeja tai ymmärtämään aiemmin löydettyjä bugeja paremmin. Koehenkilöt kommentoivat testitapauksen 5 aikana pitkälti samoja asioita kuin mitä he olivat kommentoineet aiempien testitapausten aikana. Yksi koehenkilö huomasi testitapauksen 5 aikana että punainen pallo antaa pelaajalle pluspisteitä osuessaan siniseen tai vihreään palloon, kun hän oli aiemmin huomannut vain sen että punainen pallo antaa pelaajalle miinuspisteitä osuessaan punaiseen tauluun.

Muita huomioita

Pelistä löytyi manuaalisessa testauksessa yksi tarkoitukseton bugi. Koehenkilö 1 sanoi testitapauksen 2 päätteeksi unohtaneensa tarkkailla kelloon lisättävää aikaa ja muisti tarkkailla sitä testitapauksen 3 aikana sen verran, että huomasi yhden kerran vihreän pallon osuessa vihreään maalitauluun, että pelaaja ei saanut plus- eikä miinuspisteitä, mutta kelloon lisättiin silti aikaa. Hän ei siis ajatellut että kyseessä olisi toistuva bugi, vaan oletti sen tapahtuneen vain yhden heiton kohdalla. Havainto oli kuitenkin merkittävä, koska kyseessä oli bugi, jota ei ollut injektoitu peliin tarkoituksellisesti. Pelin oli tarkoitus toimia niin, että kelloon lisätään aikaa silloin kun pelaaja saa pluspisteitä, mutta koehenkilön 1 kohdalla peli toimi tämän tarkoituksettoman bugin takia niin, että kelloon lisättiin aikaa aina kun pallo osui maalitauluun, riippumatta pallon ja maalitaulun väristä. Kelloon lisättiin aikaa myös silloin kun pelaaja ei saanut testitapauksen 3 tarkoituksellisesti injektoidun bugin takia plus- eikä miinuspisteitä, vaikka pallo osui maalitauluun.

Koehenkilön 1 löytämä tarkoitukseton bugi korjattiin ennen kuin muut kolme koehenkilöä pääsivät testaamaan peliä, koska kyseessä oli usein toistuva bugi, jolla

oli merkittävä vaikutus tarkoituksellisesti injektoituihin bugeihin. Jos tämä tarkoitukseton bugi olisi ollut täysin erillinen tarkoituksellisesti injektoidusta bugeista, se olisi voitu jättää korjaamatta yhteismitallisuuden takia. Koska automaattinen testaus tapahtui manuaalisen testauksen jälkeen, tämä tarkoitukseton bugi ei ollut pelissä myöskään siinä vaiheessa kun peliä testattiin automaattisesti, vaan ainoastaan koehenkilö 1 pääsi näkemään tämän bugin.

Kaikki koehenkilöt tarkkailivat pisteitä, mutta kolme neljästä koehenkilöstä ei muistanut tarkkailla sitä, lisätäänkö kelloon aikaa pallon osuessa maalitauluun. Yksi koehenkilö ei ilmeisesti huomioinut testausohjeiden lukemisen aikana sitä, että kelloon lisätään aikaa pelaajan saadessa pisteitä, joten hän ei ollut valmistautunut tarkkailemaan kelloa, mutta toisin kuin muut kolme koehenkilöä, hän kiinnitti kuitenkin toistuvasti huomiota siihen, että kelloon lisättiin aikaa.

Kaksi koehenkilöä ei lukenut testitapauksen 1 ohjeita ennen pelin aloittamista. Toinen näistä koehenkilöistä sanoi ettei ehtinyt lukea testausohjeita ennen pelin alkamista, mutta todellisuudessa hän aiheutti pelin alkamisen ennen aikojaan painamalla välilyöntinäppäintä liian monta kertaa, kun taas toinen koehenkilö ei joko tajunnut tai muistanut, että pelin ohjeiden lukemisen jälkeen oli tarkoitus lukea testitapauksen 1 ohjeet ja vasta sen jälkeen oli tarkoitus aloittaa peli. Tämä ei kuitenkaan ollut suuri ongelma, koska nämä koehenkilöt siirtyivät pelin päätyttyä lukemaan testausohjeita ja suorittamaan testitapauksia niiden mukaan.

Kaksi koehenkilöä ei testausohjeiden lukemisen jälkeen joko tajunnut tai muistanut, että testitapauksissa 1–3 oli tarkoituksena heittää vain tietynvärisiä palloja kohti maalitauluja. Tätä asiaa ei mainita testausohjeissa jokaisen testitapauksen kohdalla erikseen, vaan se mainitaan vain kerran yleisissä ohjeissa. Toinen näistä koehenkilöistä tajusi tämän asian testitapauksen 1 jälkeen, kun taas toinen koehenkilö ei tajunnut tätä asiaa missään vaiheessa, vaan suoritti käytännössä avointa testausta kaikissa testitapauksissa lukuun ottamatta testitapausta 4, jossa koehen-

kilöä pyydettiin olemaan heittämättä palloja ja odottamaan ajan loppumista kellosta. Tämä virhe ei estänyt näitä kahta koehenkilöä löytämästä bugeja, mutta se ei todennäköisesti myöskään tehnyt bugien löytämisestä helpompaa.

Koehenkilöillä oli myös ristiriitaisia kokemuksia pelin pelaamisesta. Kaksi koehenkilöä oli sitä mieltä, että heiton suunnan ja voiman hahmottaminen oli helppoa, kun taas yhden koehenkilön mielestä näiden asioiden hahmottaminen oli vaikeaa. Yksi koehenkilö ei kommentoinut tätä asiaa millään tavalla. Toinen kahdesta positiivisen kommentin antaneesta koehenkilöstä oli koehenkilö 1, joka löysi tarkoituksettoman bugin ja oli lisäksi ainoa bugin 2 löytänyt koehenkilö. Voidaan siis sanoa että heiton suunnan ja voiman hyvällä hahmottamisella oli merkitystä manuaalisen testauksen tulosten kannalta.

Manuaalisen testauksen tulokset olisivat todennäköisesti olleet ainakin hieman paremmat, jos koehenkilöitä olisi pyydetty suorittamaan testitapaukset useamman kuin yhden kerran, mutta näin ei tässä tutkimuksessa tehty. Testausohjeissa annettiin toki koehenkilölle mahdollisuus aloittaa peli uudestaan ja jatkaa testitapauksen suorittamista, jos hän ei ehdi suorittaa sitä loppuun asti ennen kuin kellosta loppuu aika, mutta kukaan koehenkilöistä ei kokenut minkään testitapauksen kohdalla tarvetta aloittaa peliä uudestaan, vaikka koehenkilöt ymmärsivätkin joissain tapauksissa, että heidän suorituksensa ei ollut täysin testausohjeiden mukainen.

Kolme neljästä koehenkilöstä sanoi jossain vaiheessa unohtaneensa tehdä jonkin asian, joka testausohjeissa pyydettiin tekemään, joten jos koehenkilöt olisivat suorittaneet testitapaukset useamman kuin yhden kerran, he olisivat todennäköisesti muistaneet ensimmäisellä suorituskerralla unohtuneet asiat paremmin seuraavilla suorituskerroilla. Kaikki koehenkilöt olisivat lisäksi saaneet mahdollisuuden kokeilla testitapauksessa 4, johtavatko sekä plus- että miinuspisteet pelin päättymiseen. Koehenkilöillä oli toki myös mahdollisuus testata tätä asiaa avoimessa testauksessa tai muistaa mitä testitapauksissa 1–3 tapahtui, mutta tämän asian testaaminen

olisi ollut helpointa nimenomaan pelin päättymiseen liittyvän bugin ympärille rakennetussa testitapauksessa. Koehenkilöillä olisi useamman kuin yhden suorituskerran myötä ollut myös enemmän aikaa tutustua peliin ja miettiä millä tavalla peli toimii väärin. Tässä tutkimuksessa käytetty peli on toki sen verran yksinkertainen, että tutustumiseen ei ollut syytä olettaa menevän paljon aikaa, mutta hieman pidempi tutustumisaika olisi silti voinut parantaa manuaalisen testauksen tuloksia.

Manuaalisen testauksen tuloksiin voi vaikuttaa myös tarkkailu. Tässä tutkimuksessa koehenkilöiden toimintaa tarkkailtiin testisessioiden aikana ja tarkkailun avulla nähtiin ja kuultiin asioita joita ei olisi saatu tietää ilman tarkkailua. Koska koehenkilöille annettiin palautelomake vasta testisession lopussa, on mahdollista että he eivät muistaneet palautetta kirjoittaessa kaikkia niitä asioita jotka he olisivat muistaneet, jos heillä olisi ollut mahdollisuus täyttää palautelomaketta omaan tahtiin testisession alusta lähtien. Lisäksi yksi koehenkilö jätti kirjoittamatta palautelomakkeeseen yhden asian, joka ei hänen mielestään ollut niin tärkeä että siitä pitäisi antaa kirjallista palautetta, mutta hän kertoi kuitenkin tästä asiasta suullisesti. Koehenkilö oli tässä tapauksessa oikeassa ja kyse ei ollut tutkimuksen kannalta tärkeästä asiasta, mutta joskus tällaisellakin asialla voi olla merkitystä, ja tarkkailun avulla voidaan joka tapauksessa saada tietoa asioista joita koehenkilöt eivät joko muista sisällyttää kirjalliseen palautteeseen tai koe niin tärkeiksi että niistä pitäisi antaa kirjallista palautetta. Tarkkailijalla on myös mahdollisuus vastata koehenkilöiden esittämiin kysymyksiin ja pohdintoihin. Kaikkiin kysymyksiin ei välttämättä kannata vastata, mutta joskus voi olla hyvä selventää koehenkilöille epäselviksi jääneitä asioita. Kysymyksiin vastaaminen ja asioiden selventäminen on joka tapauksessa tarkkailun avulla mahdollista. Lisäksi tarkkailija voi tehdä testisession aikana omia havaintojaan, mikä voi helpottaa manuaalisen testauksen tulosten analysointia verrattuna siihen että tuloksia pitäisi analysoida pelkästään koehenkilöiden antaman kirjalli-

sen palautteen perusteella. Ainakin tässä tutkimuksessa tarkkailun avulla tehdyistä havainnoista oli hyötyä.

Sen sijaan että koehenkilöt saivat mahdollisuuden kirjallisen palautteen antamiseen vasta testisession lopussa, heille olisi voitu antaa palautelomake heti testisession alussa. Tällöin koehenkilöillä olisi ollut mahdollisuus kirjoittaa asioita muistiin jokaisen testitapauksen jälkeen, ja olisi ollut epätodennäköisempää että koehenkilöt olisivat unohtaneet antaa kirjallista palautetta jostain asiasta. Toisaalta ei tiedetä olisivatko koehenkilöt hyödyntäneet tätä mahdollisuutta. Tähän asia olisi todennäköisesti voitu vaikuttaa pyytämällä koehenkilöitä antamaan kirjallista palautetta jokaisen testitapauksen jälkeen ja muotoilemalla palautelomake niin, että siinä ei olisi ollut ainoastaan yleisiä kysymyksiä vaan jokaiselle testitapaukselle olisi ollut erillinen kysymys. Palautelomakkeen erilainen muotoilu olisi varmistanut sen että koehenkilöt olisivat antaneet palautetta jokaisesta testitapauksesta erikseen, mutta ei sitä että koehenkilöt olisivat täyttäneet palautelomaketta jokaisen testitapauksen jälkeen. Jälkimmäisen asian varmistamiseksi koehenkilöitä olisi pitänyt muutenkin kuin palautelomakkeen kysymysten avulla pyytää antamaan kirjallista palautetta jokaisen testitapauksen jälkeen ennen seuraavan testitapauksen aloittamista.

7.2 Automaattisen testauksen tulokset

Tässä alaluvussa esitetään automaattisen testauksen tulokset. AltTesterin löytämät tarkoituksellisesti injektoidut bugit näkyvät taulukossa 7.2.

Taulukko 7.2: Automaattisessa testauksessa löydetyt bugit

	Bugi 1	Bugi 2	Bugi 3	Bugi 4
AltTester	Osittain	X	X	X

Kuten yllä olevasta taulukosta nähdään, AltTester löysi kaikki neljä bugia ainakin osittain. Seuraavaksi käydään tarkemmin läpi sitä miten yllä olevassa taulukossa esitettyihin automaattisen testauksen tuloksiin päädyttiin.

Testitapaus 1

Punaista palloa testattaessa kävi siinä mielessä huono tuuri, että tietokone pääsi heittämään punaisen pallon vain neljä kertaa, ja ainoastaan yksi näistä neljästä pallosta osui maalitauluun. Kyseisen maalitaulun väri oli vihreä, joten pelaaja sai pluspisteitä ja kelloon lisättiin aikaa. Punaiseen palloon liittyvä bugi löytyi siis vain osittain. Jos punainen pallo olisi osunut edes yhden kerran punaiseen maalitauluun, olisi ollut helppo todeta että pelaaja saa miinuspisteitä, mutta koska näin ei tapahtunut, bugin löytäminen kokonaan olisi vaatinut testitapauksen 1 suorittamista useamman kuin yhden kerran. Paremmalla tuurilla bugi olisi kuitenkin löytynyt kokonaisuudessaan yhden suorituskerran aikana.

Testitapaus 2

Siniseen palloon liittyvä bugi tuli selkeästi esille, koska sininen pallo osui kolme kertaa lähelle punaisen tai vihreän maalitaulun reunaa, kaksi kertaa lähelle punaisen tai vihreän maalitaulun keskustaa ja yhden kerran lähelle sinisen maalitaulun keskustaa. Ainoastaan sinisen pallon osuminen lähelle sinisen maalitaulun reunaa jäi puuttumaan, mutta tällainenkin osuma olisi varmasti tapahtunut jossain vaiheessa, jos testitapaus 2 olisi suoritettu useamman kuin yhden kerran. Sinistä palloa testattaessa tuli joka tapauksessa selkeästi esille, että kauas maalitaulun keskustasta osuneet pallot antavat pelaajalle enemmän plus- tai miinuspisteitä kuin lähelle maalitaulun keskustaa osuneet pallot. Kelloon lisättävän ajan virheellisyys ei näkynyt yhtä selkeästi kuin pisteiden virheellisyys, koska kelloon lisättävän ajan vaihtelu oli pienempää kuin yksittäisen heiton antaman pistemäärän vaihtelu, mutta sekin tuli kyllä esille.

Testitapaus 3

Vihreään palloon liittyvä bugi tuli myös selkeästi esille, koska vihreä pallo osui kolme kertaa lähelle vihreän maalitaulun keskustaa, kaksi kertaa lähelle vihreän maalitaulun reunaa ja kaksi kertaa lähelle punaisen tai sinisen maalitaulun reunaa. Ainoastaan vihreän pallon osuminen lähelle punaisen tai sinisen maalitaulun keskustaa jäi puuttumaan, joten pelaaja ei saanut kertaakaan miinuspisteitä, mutta tällainenkin osuma olisi varmasti tapahtunut jossain vaiheessa, jos testitapaus 3 olisi suoritettu useamman kuin yhden kerran. Vihreää palloa testattaessa tuli joka tapauksessa selkeästi esille, että pelaaja saa pluspisteitä ja kelloon lisätään aikaa silloin kun pallo osuu lähelle vihreän maalitaulun keskustaa, ja ei saa plus- eikä miinuspisteitä silloin kun pallo osuu lähelle minkä tahansa maalitaulun reunaa.

Testitapaus 4

Kuten manuaalisessa testauksessa, myös automaattisessa testauksessa tuli hyvin selkeästi esille, että peli ei päättynyt ajan loppuessa kellosta. Tämän asian toteamista varten ei olisi välttämättä tarvinnut edes kirjoittaa koodia, mutta koodin kirjoittaminen mahdollisti kuitenkin sen, että tietokone heitti testitapauksen päätteeksi palloja. Tietokone heitti yhteensä kaksi palloa, joista ensimmäinen ei antanut plus- eikä miinuspisteitä, ja toinen antoi miinuspisteitä. Oli siis selvää, että pelkkä pallon heittäminen ei johda pelin päättymiseen, vaan pelin päättyminen vaatii myös pisteitä, mutta näiden pisteiden ei tarvitse olla pluspisteitä.

Testitapaus 5

Testisession päätteeksi suoritetussa avoimessa testauksessa tuli enemmän osumia maalitauluihin kuin aiemmissa testitapauksissa, koska kaikki heitot olivat korkeita heittoja ja tietokone heitti palloja samaan tahtiin kuin testitapauksissa 1–3. Tästä

huolimatta avoin testaus täydensi aiempia testitapauksia vain sen verran, että vihreä pallo osui yhden kerran lähelle sinisen maalitaulun keskustaa, jolloin pelaaja sai miinuspisteitä. Punaisen pallon osuminen punaiseen maalitauluun ja sinisen pallon osuminen lähelle sinisen maalitaulun reunaa jäivät edelleen puuttumaan, mutta tällaisetkin osumat olisivat varmasti tapahtuneet jossain vaiheessa, jos avoin testaus tai testitapaukset 1 ja 2 olisi suoritettu useamman kuin yhden kerran.

Muita huomioita

Pelistä ei löytynyt automaattisessa testauksessa yhtään tarkoituksetonta bugia. Tutkimuksessa ei tosin saatu varmaa tietoa siitä, olisiko manuaalisessa testauksessa löytynyt tarkoitukseton bugi tullut esille myös automaattisessa testauksessa, koska kyseinen bugi korjattiin alaluvussa 7.1 mainituista syistä ennen automaattista testausta, joten kyseinen bugi ei ollut pelissä siinä vaiheessa kun peliä testattiin automaattisesti. Jokaisessa testitapauksessa oli lähtökohtaisesti mahdollista syntyä tilanne jossa tämä tarkoitukseton bugi olisi tullut esille, mutta jokaisessa testitapauksessa oli yhtä lailla mahdollisuus siihen että tällaista tilannetta ei syntyisi ja tämä bugi jäisi löytymättä. Jälkeenpäin voidaan todeta että tämä bugi ei olisi löytynyt testitapauksessa 1, koska pelaaja ei saanut siinä ollenkaan miinuspisteitä ja näin ollen bugin löytymiseen vaadittavaa tilannetta ei syntynyt, mutta kaikkien muiden testitapausten aikana tämän bugin löytyminen olisi ollut mahdollista myös jälkeenpäin ajateltuna. Paras mahdollisuus tämän bugin löytymiseen olisi jälkeenpäin ajateltuna ollut testitapauksessa 4, koska pelaaja sai siinä miinuspisteitä ja virheellinen ajan lisääminen kelloon olisi todennäköisesti tullut kaikkein selkeimmin esille silloin kun kellossa ei olisi ollut sitä ennen ollenkaan aikaa jäljellä. Toisaalta testitapaus 4 sisälsi joka tapauksessa vain yhden maalitauluun osuneen heiton, ja jos pelaaja olisi saanut tällä yhdellä heitolla pluspisteitä, testitapauksessa 4 ei olisi syntynyt tilannetta

jossa tämä bugi olisi tullut esille. Näin ollen on vaikea sanoa missä testitapauksessa olisi lähtökohtaisesti ollut suurin todennäköisyys tämän bugin löytymiseen.

Automaattisen testauksen tuloksia olisi pystytty lähes varmasti parantamaan suorittamalla testitapaukset useamman kuin yhden kerran. Testitapaus 4 olisi manuaalisen testauksen tavoin vaatinut kaksi suorituskertaa, jotta olisi päästy näkemään johtavatko miinuspisteiden lisäksi myös pluspisteet pelin päättymiseen. Automaattisessa testauksessa oli toki samat mahdollisuudet tämän asian testaamiseen avoimessa testauksessa tai päättelemiseen testitapausten 1–3 perusteella kuin manuaalisessa testauksessa, mutta tämän asian testaaminen olisi myös automaattisessa testauksessa ollut helpointa testitapauksessa 4. Kaikki muut tarkoituksellisesti injektoitujen bugien löytymiseen vaadittavat asiat olisivat voineet hyvällä tuurilla tapahtua yhden suorituskerran aikana. Tässä tutkimuksessa ei käynyt näin, mutta jos testitapaukset olisi suoritettu useamman kuin yhden kerran, kaikki nämä asiat olisivat varmasti tapahtuneet jossain vaiheessa. Testitapauksen suorittaminen useamman kuin yhden kerran on mielestäni myös yleisesti ottaen perusteltua, jos yhden suorituskerran aikana ei onnistuta testaamaan kaikkia niitä asioita joita kyseisessä testitapauksessa on tarkoitus testata.

On myös hyvä ottaa huomioon, että tässä tutkimuksessa etsittiin pääasiassa tarkoituksellisesti injektoituja bugeja. Mahdollisten tarkoituksettomien bugien löytäminen ei välttämättä olisi ollut yhtä helppoa kuin tarkoituksellisesti injektoitujen bugien löytäminen, koska mahdolliset tarkoituksettomat bugit eivät olleet tiedossa ennen testausta ja niiden ympärille ei rakennettu testitapauksia. Tätä tutkimusta varten rakennettujen testitapausten suorittaminen useamman kuin yhden kerran olisi joka tapauksessa antanut enemmän aikaa mahdollisten tarkoituksettomien bugien löytämiseen ja määrittelemiseen.

7.3 Automaattisen ja manuaalisen testauksen vertailu

Automaattisessa testauksessa löytyi suurempi osa tarkoituksellisesti injektoiduista bugeista kuin manuaalisessa testauksessa. Voidaan siis sanoa että automaattinen testaus tuotti kokonaisuutena paremmat tulokset kuin manuaalinen testaus. Manuaalisessa testauksessa koehenkilöt löysivät yksinkertaiset bugit ja osasivat määritellä ne, mutta he eivät osanneet määritellä monimutkaisempia bugeja, vaikka huomasivatkin että peli toimii jollain tavalla väärin. Automaattisessa testauksessa monimutkaisemmatkin bugit tulivat hyvin esille. Manuaalinen testaus saa kuitenkin erityismaininnan tarkoituksettoman bugin löytymisestä.

Automaattisen ja manuaalisen testauksen tulosten välillä oli tässä tutkimuksessa jo lähtökohtaisesti erona se, että manuaalisessa testauksessa koehenkilöt pääsivät antamaan kirjallista palautetta vasta testisession lopussa, kun taas automaattisessa testauksessa testitapausten aikana tehdyt havainnot kirjoitettiin muistiin jokaisen testitapauksen jälkeen ennen seuraavan testitapauksen aloittamista. Näin ollen automaattisessa testauksessa oli pienempi mahdollisuus siihen että jokin asia unohtuisi ja jäisi kirjoittamatta muistiin. Manuaalinen testaus on kuitenkin mahdollista toteuttaa niin, että koehenkilöillä on mahdollisuus antaa kirjallista palautetta jokaisen testitapauksen jälkeen, joten tätä ei voida pitää yleisellä tasolla merkittävänä erona automaattisen ja manuaalisen testauksen välillä. Tässä tutkimuksessa koehenkilöiden antamaa kirjallista palautetta täydennettiin tarkkailun avulla tehdyillä havainnoilla, ja näistä havainnoista olisi varmasti ollut hyötyä myös siinä tapauksessa että koehenkilöillä olisi ollut mahdollisuus antaa kirjallista palautetta jokaisen testitapauksen jälkeen tai heitä olisi jopa pyydetty tekemään niin.

Tietokone pystyy tarvittaessa heittämään täsmälleen samanlaisia heittoja koko testitapauksen ajan. Manuaalinen testaaja pystyy tarvittaessa pitämään tähtäyksen

samana koko testitapauksen ajan, koska tähtäyksen pitäminen samana vaatii ainoastaan sen että testaaja ei koske nuolinäppäimiin ensimmäisen heiton jälkeen, mutta manuaalisen testaajan on käytännössä mahdotonta pitää jokaisen heiton kohdalla välilyöntinäppäintä pohjassa täsmälleen yhtä pitkään, vaikka peli esittääkin heiton voiman numeroiden avulla eikä graafisesti. Sen sijaan tietokone pystyy tarvittaessa pitämään sekä heiton tähtäyksen että voiman koko ajan täsmälleen samana, ja tietokoneelle on jopa yksinkertaisempi tehtävä heittää täsmälleen samanlaisia heittoja kuin heittää erilaisia heittoja. Lisäksi tietokone pystyy tarvittaessa pitämään tähtäyksen täsmälleen samana eri testitapausten ja suorituskertojen välillä, kun taas manuaalinen testaaja pystyy tähän vain siinä tapauksessa, että hän ei koske ollenkaan nuolinäppäimiin, jolloin tähtäys pysyy koko ajan sellaisena kuin se on pelin alkaessa. Manuaalisen testaajan on käytännössä mahdotonta siirtää tähtäystä jokaisessa testitapauksessa tai jokaisella suorituskerralla täsmälleen samalla tavalla. Tietokone pystyy myös heittämään yksittäisen pallon täsmälleen haluttuun suuntaan täsmälleen halutulla voimalla, kun taas manuaaliselle testaajalle tämä on huomattavasti vaikeampi tehtävä. Näin ollen tietokone osuu tarvittaessa jokaisella heitolla maalitauluun, kun taas manuaalinen testaaja voi epäonnistua heiton suunnan tai voiman määrittämisessä niin, että pallo ei osu maalitauluun.

Manuaalisessa testauksessa on helppoa pyytää testaajaa tähtäämään tietynväriseen maalitauluun. Tämä on todennäköisesti mahdollista myös automaattisessa testauksessa, mutta maalitaulujen liikkuminen yhdistettynä niiden värien satunnaisuuteen tekee tästä tehtävästä huomattavasti haastavamman kuin mitä se on manuaalisessa testauksessa. Tehtävä olisi hieman helpompi, jos maalitaulut eivät liikkuisi tai niiden värit eivät määräytyisi satunnaisesti, ja huomattavasti helpompi, jos sekä maalitaulujen liikkuminen että niiden värien satunnaisuus poistettaisiin. Vaikka näiden asioiden poistaminen helpottaisikin automaattista testaamista, se ei kuitenkaan olisi järkevää, koska se muuttaisi itse peliä merkittävästi.

Jos maalitaulun värillä ei ole väliä, vaan mihin tahansa maalitauluun osuminen riittää, tehtävä on myös automaattisessa testauksessa suhteellisen helppo. Toisaalta pallon osuminen jokaisella heitolla täsmälleen samaan kohtaan maalitaulussa ei ole kaikissa testitapauksissa hyvä asia, eikä ole helppoa koodata testitapausta, jossa pallo osuu jokaisella heitolla johonkin maalitauluun mutta ei kuitenkaan osu jokaisella heitolla samaan kohtaan maalitaulussa. Esimerkiksi tässä tutkimuksessa käytettyä peliä testattaessa pallon täytyy voida osua eri heitoilla eri kohtiin maalitaulussa, koska pelin ei ole tarkoitus toimia niin, että pelaaja saa täsmälleen saman määrän plus- tai miinuspisteitä jokaisella kerralla pallon osuessa maalitauluun, vaan yksittäisen heiton antaman pistemäärän on tarkoitus riippua siitä mihin kohtaan maalitaulussa pallo osuu. Tästä syystä automaattisen testauksen testitapauksia ei toteutettu niin, että pallo osuu jokaisella heitolla johonkin maalitauluun, ja koska myös manuaalisessa testauksessa oli heittoja jotka eivät osuneet mihinkään maalitauluun, voidaan todeta että tämä asia ei aiheuttanut automaattiselle testaukselle merkittävää haittaa automaattisen ja manuaalisen testauksen vertailua ajatellen.

Tietokone ehtii heittää yksittäisen testitapauksen aikana enemmän palloja kuin manuaalinen testaaja, koska se ei odottele heittojen välillä vaan heittää palloja tasaiseen tahtiin, mutta koska tietynväriseen maalitauluun tähtääminen on tietokoneelle vaikeampi tehtävä kuin manuaaliselle testaajalle, tietokone ei välttämättä ehdi heittää yksittäisen testitapauksen aikana enempää tietynväriseen maalitauluun osuvia palloja, tai ylipäätään maalitauluun osuvia palloja, kuin manuaalinen testaaja. Tämä asia riippuu toki myös manuaalisen testaajan suorituksesta ja voi vaihdella eri testitapausten ja suorituskertojen välillä.

8 Yhteenveto

Tässä tutkielmassa tutkittiin automaattisen ja manuaalisen pelitestauksen eroja ja automaattisen pelitestauksen kehitysmahdollisuuksia. Tutkimuksessa etsittiin vastauksia kahteen tutkimuskysymykseen:

1. Saadaanko yksinkertaisen videopelin testauksessa paremmat tulokset automaattisella vai manuaalisella testauksella?

2. Miten automaattista testausta voisi parantaa?

Näihin tutkimuskysymyksiin etsittiin vastauksia testaamalla tätä tutkielmaa varten toteutettua yksinkertaista videopeliä, johon injektoitiin tarkoituksellisesti bugeja. Peliä testattiin ensin manuaalisesti ja sen jälkeen automaattisesti, ja lopuksi analysoitiin ja vertailtiin manuaalisen ja automaattisen testauksen tuloksia.

Ensimmäiseen kysymykseen saatiin vastaukseksi että automaattinen testaus tuotti kokonaisuutena paremmat tulokset kuin manuaalinen testaus. Manuaalisessa testauksessa on omat hyvät puolensa, ja automaattinen testaus ei voi videopelialalla korvata automaattista testausta kokonaan, mutta automaattisessa testauksessa löytyi tässä tutkimuksessa suurempi osa tarkoituksellisesti injektoiduista bugeista kuin manuaalisessa testauksessa. Manuaalisessa testauksessa löytyi kuitenkin tarkoitukseton bugi, joten manuaalinen testaus saa siitä erityismaininnan.

Voidaan myös todeta että ohjeiden antaminen manuaaliselle testaajalle on ainakin joissain tapauksissa helpompaa kuin koodin kirjoittaminen automaattista tes-

tausta varten, mutta jos koodin kirjoittaminen onnistuu, tietokone on testien suorittamisessa parempi kuin manuaalinen testaaja. Tietokone tekee juuri ne asiat jotka sitä pyydetään tekemään, kun taas manuaalinen testaaja voi ymmärtää hänelle annetut ohjeet väärin. Tietokone pystyy myös tekemään asioita jotka ovat manuaaliselle testaajalle haastavia tai lähes mahdottomia.

Toiseen kysymykseen saatiin vastaukseksi että automaattista testausta voidaan tämän tutkimuksen valossa parantaa ainakin suorittamalla testitapaukset useamman kuin yhden kerran. On hyvin mahdollista että kaikki bugit eivät löydy yhdellä suorituskerralla, ja suorituskertojen määrän kasvattaminen nostaa bugien löytymisen todennäköisyyttä. Tämä pätee myös manuaaliseen testaukseen, vaikka tässä tutkimuksessa suoritettiin manuaalisessa testauksessa jokainen testitapaus neljä kertaa koska manuaaliseen testaukseen osallistui neljä koehenkilöä. Yksittäinen koehenkilö suoritti kuitenkin jokaisen testitapauksen vain yhden kerran.

Automaattista testausta voidaan tämän tutkimuksen valossa parantaa myös rakentamalla testitapaukset niin, että ne vastaavat mahdollisimman hyvin manuaalisen testauksen testitapauksia. Videopelejä ei tehdä tietokoneiden vaan ihmisten pelattaviksi, joten automaattisessa testauksessa on hyvä pyrkiä jäljittelemään mahdollisimman tarkasti ihmispelaajan pelaamista. Mitä paremmin automaattisen testauksen testitapaus vastaa sitä miten sama testitapaus toteutettaisiin manuaalisessa testauksessa, sitä tarkemmin pystytään testaamaan juuri sitä asiaa jota kyseisessä testitapauksessa halutaan testata.

Testitapauksia rakennettaessa on myös hyvä miettiä millaisia bugeja testattavassa videopelissä voisi olla. Tässä tutkimuksessa käytettiin tarkoituksellisesti injektoituja bugeja, koska haluttiin varmistaa että pelin testaaminen tuottaisi riittävästi tuloksia automaattisen ja manuaalisen testauksen vertailua varten. Normaalissa testitilanteessa peliin ei ole injektoitu tarkoituksellisesti bugeja, mutta mahdollisia bugeja on silti hyvä miettiä testitapauksia rakennettaessa. Avointa testausta on

kuitenkin myös hyvä tehdä tiettyä tarkoitusta varten rakennettujen testitapausten lisäksi, koska kaikkien mahdollisten bugien ennakoiminen ei ole realistista.

Tämän tutkimuksen manuaalisen testauksen osuutta voitaisiin jatkaa kasvattamalla koehenkilöiden määrää, jolloin saataisiin suurempi määrä tuloksia, ja tutkimalla mitä tapahtuu ja saadaanko erilaisia tuloksia jos koehenkilöillä on mahdollisuus antaa kirjallista palautetta testisession alusta lähtien. Suurempi koehenkilöiden määrä mahdollistaisi myös jonkinlaisen variaation testisession etenemisessä eri koehenkilöiden välillä. Tämä variaatio voisi tarkoittaa esimerkiksi sitä, että osalle koehenkilöistä annettaisiin palautelomake vasta testisession lopussa, osa saisi palautelomakkeen heti testisession alussa mutta heitä ei varsinaisesti pyydettäisi täyttämään sitä jokaisen testitapauksen jälkeen, ja osaa pyydettäisiin antamaan kirjallista palautetta jokaisesta testitapauksesta erikseen. Palautelomakkeesta voisi myös tehdä ainoastaan yleisiä kysymyksiä sisältävän version lisäksi sellaisen version jossa on jokaiselle testitapaukselle erillinen kysymys, jolloin saataisiin vielä lisää erilaisia yhdistelmiä tutkittaviksi ja vertailtaviksi.

Automaattisen testauksen osuutta voitaisiin jatkaa tutkimalla mahdollisuuksia toteuttaa testitapaukset niin, että ne vastaavat vielä paremmin manuaalisen testauksen testitapauksia kuin tässä tutkimuksessa käytetyt automaattisen testauksen testitapaukset. Jos ja todennäköisesti kun tällaisia mahdollisuuksia on olemassa, paranneltujen testitapausten tuottamia tuloksia voisi verrata tämän tutkimuksen tuottamiin automaattisen testauksen tuloksiin, jotta nähtäisiin kuinka merkittävä ero näiden tulosten välillä on suhteessa siihen kuinka paljon testitapauksia parannellaan. Tällainen tutkimuksen jatkaminen voisi auttaa pelitestauksen automatisoinnissa siltä osin kuin automaattinen testaus voi korvata manuaalisen testauksen, koska jos tietokone saadaan testaamaan videopelejä samalla tavalla kuin ihminen testaisi niitä, manuaalisen testauksen tarve vähenee.

Lähdeluettelo

[1] M. Kiiskinen, ”Pelitestaamisen teoria ja pelitestaaminen työnä”, Opinnäytetyö, 2020. url: https://urn.fi/URN:NBN:fi:amk-202001181364.

[2] P. Bourque ja R. Fairley, *Guide to the Software Engineering Body of Knowledge*. IEEE, 2014, ISBN: 9780769551661.

[3] K. Naik ja P. Tripathy, *Software Testing and Quality Assurance: Theory and Practice*. Wiley, 2008, ISBN: 9780470382844.

[4] I. Sommerville, *Software Engineering, Global Edition*. Pearson, 2016, ISBN: 9781292096131.

[5] ISO/IEC JTC 1/SC 7, ”ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary”, 2017. url: https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

[6] GeeksforGeeks. ”Configuration Testing - Software Testing”. (2024), url: https://www.geeksforgeeks.org/software-testing-configuration-testing/.

[7] C. Politowski, F. Petrillo ja Y.-G. Gueheneuc, ”A Survey of Video Game Testing”, teoksessa *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, IEEE, toukokuu 2021, s. 90–99. DOI: 10.1109/ast52587.2021.00018.

[8] C. P. Schultz, R. Bryant ja T. Langdell, *Game testing all in one*. Florence, AL: Course Technology, 2005, ISBN: 1592003737.

[9] Game-Ace. ”Your Ultimate Guide to Game Testing”. (2021), url: https://game-ace.com/blog/guide-to-game-testing/ (viitattu 11. 12. 2024).

[10] M. Thakkar. ”Software Testing vs. Game Testing: The Key Differences”. (2020), url: https://www.kiwiqa.com.au/blogpost/software-testing-vs-game-testing-the-key-differences/ (viitattu 11. 12. 2024).

[11] R. E. S. Santos, C. V. C. Magalhães, L. F. Capretz, J. S. Correia-Neto, F. Q. B. da Silva ja A. Saher, ”Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners”, teoksessa *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, sarja ESEM ’18, Oulu, Finland: Association for Computing Machinery, 2018, ISBN: 9781450358231. DOI: 10.1145/3239235.3268923.

[12] Altom Consulting. ”AltTester Documentation”. (2023), url: https://alttester.com/docs/sdk/latest/home.html (viitattu 11. 12. 2024).

[13] GameDriver, Inc. ”GameDriver”. (2022), url: https://www.gamedriver.io/ (viitattu 11. 12. 2024).

[14] OpenJS Foundation. ”Introduction to Appium”. (2024), url: https://appium.io/docs/en/latest/intro/ (viitattu 11. 12. 2024).

[15] T-Plan Ltd. ”T-Plan Game Test Automation”. (2021), url: https://www.t-plan.com/game-test-automation/ (viitattu 11. 12. 2024).

Liite A Manuaalisen testauksen testausohjeet

Yleisiä ohjeita

Aloita lukemalla pelin ohjeet, jotka tulevat näkyviin, kun siirryt pelin aloitusnäkymästä eteenpäin. Siirry eteenpäin yhteensä neljä kertaa, jotta pääset lukemaan kaikki pelin ohjeet. Kun olet lukenut pelin ohjeet, jatka testausohjeiden lukemista. Kun olet lukenut testausohjeet, voit aloittaa pelin.

Jos peli antaa erivärisen pallon kuin mitä testitapauksessa pyydetään heittämään, pyri heittämään pallo niin, että se ei osu mihinkään maalitauluun. Tämä onnistuu heittämällä pallo mahdollisimman vähäisellä voimalla ja suuntaamalla heitto mahdollisimman alas.

Jos et onnistu suorittamaan testitapausta loppuun asti ennen kuin kellosta loppuu aika, aloita peli uudestaan ja jatka testitapauksen suorittamista siitä mihin jäit.

Testitapaus 1

Ohjeet: Aloita peli. Heitä punainen pallo niin, että se osuu mihin tahansa maalitauluun. Toista niin monta kertaa kuin ehdit ennen kuin kellosta loppuu aika, kuitenkin niin, että punainen pallo osuu vähintään kerran punaiseen maalitauluun ja vähintään kerran siniseen tai vihreään maalitauluun.

Odotettu toiminta: Jos punainen pallo osuu punaiseen maalitauluun, pelaaja saa sitä enemmän pisteitä ja lisää sekunteja kelloon, mitä lähempänä maalitaulun keskustaa osumakohta on. Jos punainen pallo osuu siniseen tai vihreään maalitauluun, pelaaja menettää sitä enemmän pisteitä, mitä lähempänä maalitaulun keskustaa osumakohta on.

Testitapaus 2

Ohjeet: Aloita peli. Heitä sininen pallo niin, että se osuu mihin tahansa maalitauluun. Toista niin monta kertaa kuin ehdit ennen kuin kellosta loppuu aika, kuitenkin niin, että sininen pallo osuu vähintään kerran siniseen maalitauluun ja vähintään kerran punaiseen tai vihreään maalitauluun.

Odotettu toiminta: Jos sininen pallo osuu siniseen maalitauluun, pelaaja saa sitä enemmän pisteitä ja lisää sekunteja kelloon, mitä lähempänä maalitaulun keskustaa osumakohta on. Jos sininen pallo osuu punaiseen tai vihreään maalitauluun, pelaaja menettää sitä enemmän pisteitä, mitä lähempänä maalitaulun keskustaa osumakohta on.

Testitapaus 3

Ohjeet: Aloita peli. Heitä vihreä pallo niin, että se osuu mihin tahansa maalitauluun. Toista niin monta kertaa kuin ehdit ennen kuin kellosta loppuu aika, kuitenkin niin, että vihreä pallo osuu vähintään kerran vihreään maalitauluun ja vähintään kerran punaiseen tai siniseen maalitauluun.

Odotettu toiminta: Jos vihreä pallo osuu vihreään maalitauluun, pelaaja saa sitä enemmän pisteitä ja lisää sekunteja kelloon, mitä lähempänä maalitaulun keskustaa osumakohta on. Jos vihreä pallo osuu punaiseen tai siniseen maalitauluun, pelaaja menettää sitä enemmän pisteitä, mitä lähempänä maalitaulun keskustaa osumakohta on.

Testitapaus 4

Ohjeet: Aloita peli, mutta älä heitä yhtään palloa. Odota niin kauan, että kellosta loppuu aika.

Odotettu toiminta: Sama kuin mitä tapahtui aiemmissa testitapauksissa ajan loppuessa kellosta eli kello pysähtyy, pelialueen ulkopuolelle siirtyvien maalitaulujen tilalle ei enää tule uusia maalitauluja, pelaaja ei voi enää saada lisää pisteitä eikä sekunteja kelloon, "Please wait x seconds-teksti tulee näkyviin ja sen tilalle tulee myöhemmin lopullinen pistemäärä, pallot katoavat ja "Press Space to continue-teksti tulee näkyviin.

Testitapaus 5

Ohjeet: Vapaa/Avoin testaus. Siirry edellisestä testitapauksesta tähän testitapaukseen heittämällä minkä tahansa värinen pallo niin, että se osuu mihin tahansa maalitauluun.

Liite B Automaattisen testauksen ohjelmakoodit

Automaattisessa testauksessa käytetyt ohjelmakoodit esitellään tässä liitteessä B testitapausten mukaisessa järjestyksessä:

Ohjelmalistaus 1 Testitapaus 1

```
{csharp}
using NUnit.Framework;
using Altom.AltDriver;

public class TestRedBall
{
    public AltDriver altDriver;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        altDriver =new AltDriver();
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        altDriver.Stop();
    }

    [Test]
    public void Test()
    {
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(0.1f);
        var timeText = altDriver.FindObject(By.NAME, "Time Text");
        while (timeText.enabled && timeText.GetText() != "Time left: 1:00" &&
        timeText.GetText() != "Time left: 0:00")
        {
            var ballFound = false;
            var redBallFound = false;
            var balls = altDriver.FindObjectsWhichContain(By.NAME, "Ball");
            foreach (var ball in balls)
            {
                if (ball.worldX == 0 && ball.worldY == 6.5f && ball.worldZ == 0)
                {
                    ballFound = true;
                    if (ball.name == "RedBall(Clone)")
                    {
                        redBallFound = true;
                    }
                }
            }
            if (redBallFound)
            {
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
            }
            else if (ballFound)
            {
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 0.1f, true);
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
            }
        }
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.B, 0.1f);
    }
}
```

Ohjelmalistaus 2 Testitapaus 2

```
{csharp}
using NUnit.Framework;
using Altom.AltDriver;

public class TestBlueBall
{
    public AltDriver altDriver;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        altDriver = new AltDriver();
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        altDriver.Stop();
    }

    [Test]
    public void Test()
    {
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(0.1f);
        var timeText = altDriver.FindObject(By.NAME, "Time Text");
        while (timeText.enabled && timeText.GetText() != "Time left: 1:00" &&
        timeText.GetText() != "Time left: 0:00")
        {
            var ballFound = false;
            var blueBallFound = false;
            var balls = altDriver.FindObjectsWhichContain(By.NAME, "Ball");
            foreach (var ball in balls)
            {
                if (ball.worldX == 0 && ball.worldY == 6.5f && ball.worldZ == 0)
                {
                    ballFound = true;
                    if(ball.name == "BlueBall(Clone)")
                    {
                        blueBallFound = true;
                    }
                }
            }
            if (blueBallFound)
            {
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
            }
            else if (ballFound)
            {
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 0.1f, true);
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
            }
        }
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.B, 0.1f);
    }
}
```

Ohjelmalistaus 3 Testitapaus 3.

```
{csharp}
using NUnit.Framework;
using Altom.AltDriver;

public class TestGreenBall
{
    public AltDriver altDriver;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        altDriver =new AltDriver();
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        altDriver.Stop();
    }

    [Test]
    public void Test()
    {
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(0.1f);
        var timeText = altDriver.FindObject(By.NAME, "Time Text");
        while (timeText.enabled && timeText.GetText() != "Time left: 1:00" &&
        timeText.GetText() != "Time left: 0:00")
        {
            var ballFound = false;
            var greenBallFound = false;
            var balls = altDriver.FindObjectsWhichContain(By.NAME, "Ball");
            foreach (var ball in balls)
            {
                if (ball.worldX == 0 && ball.worldY == 6.5f && ball.worldZ == 0)
                {
                    ballFound = true;
                    if (ball.name == "GreenBall(Clone)")
                    {
                        greenBallFound = true;
                    }
                }
            }
            if (greenBallFound)
            {
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
            }
            else if (ballFound)
            {
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 0.1f, true);
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
            }
        }
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.B, 0.1f);
    }
}
```

Ohjelmalistaus 4 Testitapaus 4

```
{csharp}
using NUnit.Framework;
using Altom.AltDriver;

public class TestWait
{
    public AltDriver altDriver;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        altDriver =new AltDriver();
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        altDriver.Stop();
    }

    [Test]
    public void Test()
    {
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(60);
        var scoreText = altDriver.FindObject(By.NAME, "Score Text");
        altDriver.SetDelayAfterCommand(0.1f);
        while(scoreText.GetText() == "Score: 0")
        {
            var ballFound = false;
            var balls = altDriver.FindObjectsWhichContain(By.NAME, "Ball");
            foreach (var ball in balls)
            {
                if (ball.worldX == 0 && ball.worldY == 6.5f && ball.worldZ == 0)
                {
                    ballFound = true;
                }
            }
            if (ballFound)
            {
                altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
            }
        }
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.B, 0.1f);
    }
}
```

Ohjelmalistaus 5 Testitapaus 5

```
{csharp}
using NUnit.Framework;
using Altom.AltDriver;

public class TestAllBalls
{
    public AltDriver altDriver;
    //Before any test it connects with the socket
    [OneTimeSetUp]
    public void SetUp()
    {
        altDriver = new AltDriver();
    }

    //At the end of the test closes the connection with the socket
    [OneTimeTearDown]
    public void TearDown()
    {
        altDriver.Stop();
    }

    [Test]
    public void Test()
    {
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.Space, 0.1f);
        altDriver.SetDelayAfterCommand(0.1f);
        var throwDirection = 1;
        var timeText = altDriver.FindObject(By.NAME, "Time Text");
        while (timeText.enabled && timeText.GetText() != "Time left: 1:00" && timeText.GetText() != "Time left: 0:00")
        {
            var ballFound = false;
            var balls = altDriver.FindObjectsWhichContain(By.NAME, "Ball");
            foreach (var ball in balls)
            {
                if (ball.worldX == 0 && ball.worldY == 6.5f && ball.worldZ == 0)
                {
                    ballFound = true;
                }
            }
            if (ballFound)
            {
                if (throwDirection == 1)
                {
                    altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.LeftArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                    altDriver.PressKey(AltKeyCode.RightArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                }
                if (throwDirection == 2)
                {
                    altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                    altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                }
                if (throwDirection == 3)
                {
                    altDriver.PressKey(AltKeyCode.UpArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.RightArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.Space, 0.1f, 1.0f, true);
                    altDriver.PressKey(AltKeyCode.LeftArrow, 0.5f, 0.5f, true);
                    altDriver.PressKey(AltKeyCode.DownArrow, 0.5f, 0.5f, true);
                }
                throwDirection = throwDirection + 1;
                if(throwDirection == 4)
                {
                    throwDirection = 1;
                }
            }
        }
        altDriver.SetDelayAfterCommand(10);
        altDriver.PressKey(AltKeyCode.B, 0.1f);
    }
}
```