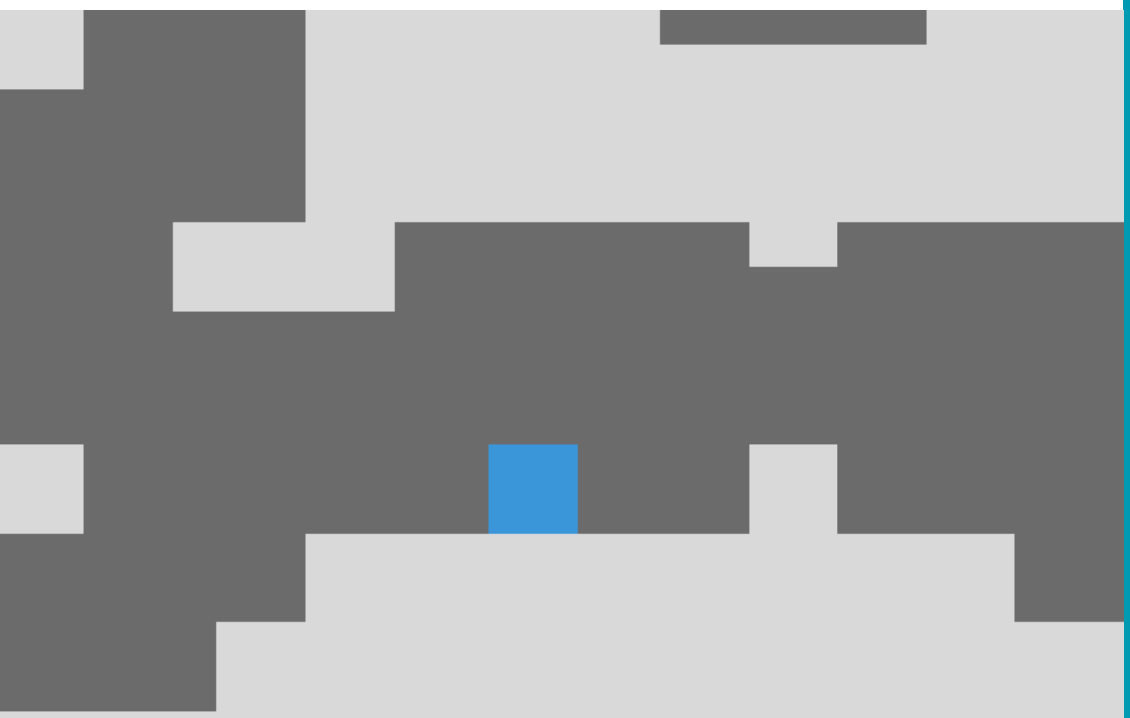
Otto-Pekka Taskinen

2D-tasohyppelypelien erilaiset mekaniikat ja käytänteet



Tietojenkäsittely

Tradenomi

Syksy 2023

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Taskinen Otto-Pekka Aleksi

Työn nimi: 2D-tasohyppelypelien erilaiset mekaniikat ja käytänteet

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: Tasohyppely, pelifysiikat, pelimekaniikat, ohjelmointi, 2D, peliohjelmointi

Työn tavoitteena oli luoda suomenkielinen lähde 2D-tasohyppelypeli genren yleisimmille mekaniikoille ja käytänteille. Työ päätettiin toteuttaa aiheeseen liittyvien suomenkielisten lähteitten puutteen takia ja yleisestä kiinnostuksesta aiheeseen. Aiheeseen liittyviä lähteitä löytyykin paljon, mutta useimmiten vain englanniksi. Luomalla suomenkielisen lähteen aiheelle toivottiin, että mekaniikkojen ja käytänteitten hyödyntäminen peleissä lisääntyisi ja niistä voisi oppia suomeksi. Kyseisten mekaniikkojen ja käytänteitten, etenkin niitten puutteella, on suuret vaikutukset tasohyppelypelien pelaajakokemukseen.

Työssä käytiin läpi 2D-tasohyppelypeleissä hyödynnettyjä mekaniikkoja ja käytänteitä, joilla pyritään parantamaan pelaajan pelikokemusta. Työssä käytiin ensin läpi tasohyppelygenren historia ja siihen, miten se on muuttunut ajan myötä. Tällä kartoitettiin genren kehittymistä ja sen eri vaiheita. Seuraavaksi perehdyttiin genren ominaisiinpiirteisiin ja muutamaan genren peliin, joilla oli merkittävä vaikutus genrelle sen historiassa. Tällä haluttiin selventää genrelle ominaisia piirteitä ja tuoda ilmi genren pelejä. Sen jälkeen käsiteltiin genrelle ominaisia mekaniikkoja ja käytänteitä, miten ne toimivat, mitä niillä haluttaan saavuttaa ja miten ne voitaisiin implementoida.

Työssä kehitettiin 2D-tasohyppelypeli, joka kirjoitettiin C++-ohjelmointikielellä. Projektissa hyödynnettiin OpenGL:ää pelin piirtämiseen. Peli toteutettiin ilman valmiin pelimoottorin hyödyntämistä, jolloin kaikki tekstissä käydyt toiminnallisuudet implementoitiin itse. Peliprojektin tavoitteena oli toimia esimerkkinä, jota hyödyntämällä voitaisiin luoda tasohyppelypeli, riippumatta ohjelmointikielestä, pelimoottorista tai alustasta.

Aikaansaannoksena projektista luotiin työkalu, jolla voidaan testata tekstissä käytyjen mekaniikkojen ja käytänteitten vaikutuksia tasohyppelypelin pelattavuuteen ja kokemukseen. Työkalussa on myös mahdollista muuttaa mekaniikkojen ja käytänteitten piirteitä. Sillä voidaan testata, miten ne toimivat eri tavalla keskenään ja millainen vaikutus niillä on pelattavuuteen.

KAMK • University of Applied Sciences

Abstract

Author(s): Taskinen Otto-Pekka Taskinen

Title of the Publication: Different Mechanics and Practices of 2D Platformers

Degree Title: Bachelor's degree (AMK) of Business Administration, Business Information Technology

Keywords: Platformer, game physics, game mechanics, programming, 2D, game programming

The goal of this thesis was to create a Finnish source for the common mechanics and practices used in 2D platformer games. The subject was chosen for the lack of sources regarding the topic in Finnish and the overall interest towards it. There are a good number of sources regarding the mechanics and practices, but mostly in English. By creating a Finnish source for the topic, it was hoped that the usage of the mechanics and practices covered in the text would increase and there would be a way to learn about them in Finnish. The mechanics and practices in question play a big role in the player experience of a platformer game.

The thesis covered mechanics and practices which are found in 2D platformer games, which are there in attempt to improve player experience. First, the history of the platformer genre was covered and the changes which it faced during those times. With this, the development of the genre was made clearer, along with the changes which that took place. Next, the general characteristic of the genre were explained as well as couple of important games in the genre which were important for the genre's development. With this, the characteristics of the genre were elucidated and some of the genre's games were highlighted. After that, the mechanics and practices were considered how they work, what their goal is, and how they could be implemented.

A 2D platformer game was created as a part of the thesis. It was written with C++ programming language and OpenGL was used for rendering the game. The game was created without using a game engine, so every feature which was covered in the text were implemented. The goal for the project was to work as an example of how a platformer game could be created without depending on the programming language, game engine or platform.

At the end the project, a tool was created, which can be used to test the effects of the covered mechanics and practices on the gameplay and experience of a platformer game. In the tool, it is possible to change the variables of the mechanics and practices. With this, the different interactions and how the changes affect gameplay can be tested.

Sisällys

Symboliluettelo

2.5D	2D:n ja 3D:n elementtien yhdistelmä.
2D	Lyhenne kaksiulotteisuudelle
3D	Lyhenne kolmiulotteisuudelle
AABB	Axis-aligned bounding box (suom. akselin mukainen rajausalue)
Hallittava hyppykorkeus	Variable jump height (suom. Hallittava hyppykorkeus), pelimekaniikka
Hyppypuskurointi	Jump Buffering (suom. hyppypuskurointi), pelimekaniikka
IDE	IDE, Integrated Development Environment (suom. ohjelmointiympäristö)
Indie-kehittäjä	Indie Developer (suom. indie-kehittäjä), yhden tai useamman kehittäjän tiimi.
Kiipeilypeli	Climbing game (suom. kiipeilypeli), peligenre
Kojoottiaika	Coyote time (suom. kojoottiaika), pelimekaniikka
Liukulukuvektori	Float vector, vektori, joka sisältää liukulukuja
OBB	Oriented bounding box (suom. käännettävä rajausalue)
Shader	Shader (suom. varjostin), ohjelma, jolla voidaan vaikuttaa pelin renderöintiin.
Sprite	Sprite, 2D-kuva tai -animaatio
Tasohyppelypeli	Platformer game (suom. tasohyppelypeli)
Törmäys	Collision (suom. törmäys), kahden törmäysalueen keskeinen kohtaaminen
Törmäysalue	Collider (suom. törmäysalue), peliobjektin komponentti, joka reagoi muihin törmäysalueisiin

1 Johdanto

Opinnäytetyön aiheena on 2D-tasohyppelypeleissä hyödynnetyt mekaniikat ja käytänteet, joidenka tarkoituksena on parantaa pelaajan pelikokemusta. Tasohyppelypelit ovat genrenä erittäin vanha ja monen innovaation kokenut genre, olipa tämä pelihalleista siirtyminen kotikonsoleille tai 2D:stä 3D:hen. Tämän pitkän historian aikana genre on kehittynyt suuresti, minkä kautta genreä on parannettu luomalla ja keksimällä uusia genrekohtaisia mekaniikkoja ja käytänteitä.

Aihe valittiin siksi, että kyseisille mekaniikoilla ja käytänteille haluttiin luoda suomenkielinen lähde, joista niistä voisi oppia. Näistä löytyy hyvä määrä informaatiota muilla kielillä, mutta ei suomeksi. Aihe valittiin myös sen takia, että mekaniikat ja käytänteet haluttiin tuoda esille ja näyttää, miten ne voitaisiin implementoida. Käännökset yleisistä tasohyppelypelien termeistä löytyvät liitteestä 1.

Aihe on rajattu muutaman mekaniikkaan ja käytänteeseen, jotka huomattavasti vaikuttavat pelin, varsinkin pelaajahahmon, toimintoihin. Mekaniikkoihin ja käytänteisiin myös perehdytään 2D-ympäristössä, koska ne on helpompi ymmärtää ja selittää näin, mutta ne toimivat miltei samalla tavalla 3D-peleissä.

Tavoitteena tällä työllä on toimia suomenkielisenä lähteenä kyseisille mekaniikoille ja käytänteille, mitä ne ovat, miten ne toimivat ja miten ne voitaisiin implementoida. Teoriaosuudessa perehdytään mekaniikan tai käytänteen toiminnallisuuteen ja siihen, mitä sillä halutaan saavuttaa. Käytännön osuudessa mekaniikka tai käytänne implementoidaan peliprojektiin, joka on kirjoitettu C++-ohjelmointikielellä hyödyntäen OpenGL:ää sen piirtämiseen. Lopullinen projekti tulee toimimaan työkaluna, jossa mekaniikkojen vaikutusta voitaisiin testata peliympäristössä.

2 Tasohyppelypelit genrenä

Tasohyppelypelit, tunnetaan myös nimellä tasoloikkapelit, ovat videopeligenre, jossa pelaaja ohjaa hahmoa kentän alusta kentän määriteltyyn loppupisteeseen hahmon kykyjä hyödyntäen. Tasohyppelypeli genrenä kehittyi toisesta 1980-luvulla suositusta peligenrestä, joka oli kiipeilypeligenre. Kiipeilypeligenreen kuuluu monia tunnettuja kolikkopelejä, joista tunnetuin on Donkey Kong, joka asetti uuden mallin kiipeilypeleille. [1, s. 50–57.]

Genren nimi viittaa peleissä oleviin tasoihin, joita pitkin pelaaja ohjaa hahmoa joko kävelemällä tai hyppimällä. Nämä ovat erittäin yleisiä asioita, joita tehdään miltei kaikissa toimintapeleissä, joten tasohyppelypelejä pidetään toimintapelien alalajina. Liikkuessaan kenttää pitkin pelaajan tulee päihittää vihollisia, kerätä esineitä, väistellä esteitä ja lopuksi päästä kentän loppuun asti. [2.] Kuvassa 1 näkyy, miltä pelihallit näyttivät.

Kuva 1. Kuva pelihallista vuodelta 1982. [3]

2.1 Genren historia

Tasohyppelypelit saivat alkunsa pelihalleissa. Ensimmäiset pelit, joita voidaan ajatella olevan tasohyppelypelien esi-isiä sisältävät joitakin genren piirteitä, mutta ei kaikkia. Ensimmäinen peli, jossa pelaaja ohjaa hahmoa, joka kykenee hyppimään, on Frogs-niminen peli vuodelta 1978. Pelaaja ei tosin pääse hyödyntämään tätä kykyä hyppiäkseen tasolta toiselle. Eri tasoilla liikkuminen oli tärkeä osa Space Panic -peliä, joka julkaistiin 1980. Tosin genren toinen piirre, eli hyppiminen, puuttui Space Panicista kokonaan. Sen sijaan pelaajan piti hyödyntää tikkaita kiivetäkseen toisille tasoille. [4.]

Ei kestänyt pitkään, kunnes julkaistiin peli, joka yhdistäisi nämä kaksi eri mekaniikkaa luodakseen pelin, jota pidetään ensimmäisenä tasohyppelypelinä. Donkey Kong otettiin vastaan hyvin. Se erosi paljon sen aikaisista pelihallipeleistä, joissa pelaaja seikkaili sokkeloissa, kuten Pac-Man-nimisessä pelissä. [5.]

Pelien siirtyessä pelihalleista koteihin kotikonsoleitten ja tietokoneitten avulla, mukana tulivat tasohyppelypelit. Activisionin kehittämä Pitfall peli oli ensimmäisten tasohyppelypelien joukoissa, joka oli pelattavissa kotikonsolilla. [6, s. 16] Julkaistu 1982, Pitfallin maailma koostui 255 ruudusta, joiden välillä pelaaja liikkui toisin kuin aikaisemmissa tasohyppelypeleissä. [7].

Konsoliteknologian kehittyessä myös pelien mekaniikat kehittyivät mukana. Namcon vuonna 1984 julkaisema Dragon Buster [8] hyödynsi uudempien konsoleitten tehoja lisäämällä mekaniikkoja, jotka löytyvät miltei jokaisessa genressä. Kyseiset mekaniikat ovat elämäpistemittari ja tuplahyppy. [9; 10.]

Donkey Kongin suosiosta luotu Super Mario Bros nousi nopeasti parhaiten myydyimmäksi videopeliksi. 1985 julkaistua Super Mario Brosia pidetään tähän päivään parhaimpana videopelinä. Super Mario Brosin suosion takana oli Marion ohjattavuus, Koji Kondon säveltämä muistettava musiikki ja Sienivaltakunnan värikäs maailma. [11.] Shigeru Miyamoto, Super Mario Brossin suunnittelija, on todennut haastattelussa, että he keskittyivät pitämään pelin simppelinä samaan aikaan tekemällä Marion liikkumisesta hauskan ja helposti ohjattavan [12].

Tasohyppelypelien suuri suosio 8-bittisillä kotikonsoleilla jatkui 16-bittisille konsoleille ja oli saapumassa kulta-aikaansa. Nintendon maskotin, Super Marion, suosio aikaan sai kilpavarustelun pelifirmojen välillä. Tämän kilpailun luomista hahmoista tunnetuin on Segan luoma Sonic the Hedgehog, jota pidetään tähänkin päivään asti Marion kilpailijana.

1991 julkaistu Sonic the Hedgehog -peli erottautui muista aikansa tasohyppelypeleistä päähahmon nopeuden ja luonteen takia. Sonic the Hedgehog ei keskittynyt sinänsä niin paljon hyppimiseen kuin Mario ja tämä huomataan pelin kentissä. [4.]

Sonic oli ensimmäinen oikea kilpailija Mariolle. Ensimmäinen Sonic the Hedgehog -peli päihitti Super Mario Worldin, Super Nintendo Entertainment Systemin Mario-pelin, myynneissä kiitos Segan markkinointitaktiikkojen. [13.] Sonicin suosio ei kuitenkaan ollut ikuinen, mutta Sonic pelien nopeus ja konsepti eläinhahmoista jättivät suuret vaikutteet tuleviin tasohyppelypeleihin. [4.] Kuvassa 2 näkyy Sonic the Hedgehog -pelin ensimmäisen kentän alku.

Kuva 2. Sonic the Hedgehog -pelin ensimmäinen kenttä. [14]

Tasohyppelypelit alkoivat laskemaan suosiossa, kun Nintendon Nintendo 64 ja Sonyn Playstation kykenivät pyörittämään kolmiulotteisia pelejä. Siitä huolimatta kaksiulotteisia pelejä vielä tehtiin, myös tasohyppelypelejä, mutta tasohyppelypelien kulta-aika oli tulossa päätökseen. 3D:hen siirtyminen tasohyppelypeligenrelle ei ollut helppoa ja moni genren peleistä pysyi 2D:n maailmassa. Tämä ei tosin estänyt pelinkehittäjiä kokeilemasta yhdistämällä 3D:tä ja 2D:tä, joidenka yhdistäminen tunnetaan käsitteellä 2.5D. [4, 15.] Esimerkkinä tästä testailusta on Sega Saturnille vuonna 1994 julkaistu Clockwork Knight -peli, jossa pelin grafiikat olivat 3D:nä, mutta hahmo liikkui 2D-tilassa. [16.]

Tosin ei mennyt pitkään, kunnes ensimmäiset 3D-pelit, joissa pelaaja kykeni hyppäämään, julkaistiin. Ensimmäinen 3D-tasohyppelypeli oli Exactin kehittämä Jumping Flash, joka julkaistiin 1995. Peli kuvattiin ensimmäisessä persoonassa, mikä eroaa muista tasohyppelypeleistä. [4.]

Nintendon uusi kotikonsoli, Nintendo 64, julkaistiin uuden Super Mario -pelin kanssa, Super Mario 64. Julkaistu 1996, Super Mario 64:sta tuli uusi normi 3D-tasohyppelypeleille. Marion ohjattavuus kolmiulotteisessa tilassa oli sulavaa ja hauskaa, varsinkin ottaen huomioon Marion liikkumiskyvyt, kuten seinähyppiminen ja pelin avoimet kentät, jossa niitä pääsi hyödyntämään. [4.] Sulavuuteen vaikutti myös se, että pelaaja ohjasi Marioa analogisella sauvalle ristiohjaimen sijaan [17].

Super Mario 64 ei tosin ollut ainoa hyvin menestyvä tasohyppelypeli. Sony, halutessaan parantaa uuden PlayStationin myyntiä, kehitti myös oman tasohyppelypelimaskottinsa. Naughty Dogin kehittämää Crash Bandicoot -peliä pidettiin Marion kilpailijana, kuten Sonic vuonna 1991. Crash Bandicootista tuli PlayStationin maskotti. [18.] Kuten Sega aikanaan, Sony markkinoi ja mainosti Crashia ja PlayStationia pitämällä sitä parempana kuin Super Marioa ja Nintendo 64:ä [19].

Nykyään tasohyppelypelien suosio on pienempi kuin 80- ja 90-luvulla. Siitä huolimatta tasohyppelypelejä tehdään, molempia sekä 2D että 3D. [20.] Genre on varsinkin suosittu indie-kehittäjien keskuudessa. Indie-kehittäjien uskallus ja vapaus testata outoja ja mielenkiintoisia ideoita/mekaniikkoja heidän peleissänsä jatkaa genren kehitystä. [21.] Sen lisäksi myös suositut tasohyppelypelisarjat, kuten Super Mario ja Sonic the Hedgehog, vieläkin saavat uusia pelejä. Genren ympärillä on iso ja vahva fanikanta, joka pitää genren hengissä [20].

2.2 Genren yleisiä piirteitä

Tasohyppelypelit, riippuen pelin tavoitteista, sisältävät aina jollain tavalla hahmon liikuttamista kentän alusta loppuun. Genren määrittävät elementit. [22.] Kenttien sisältö ja tavoitteet vaihtelevat pelien välillä, mutta yleisimpiä tavoitteita ovat: esineitten, kuten kolikkojen kerääminen, vihollisten päihittäminen tai väistely, kentän läpäisy tietyssä ajassa tai pulmien ratkaiseminen. [4.] Kenttien ainoa tavoite voi myös olla vain läpäistä kenttä ilman mitään muita sivutavoitteita [23].

2.2.1 Liikkuminen

Pelaajahahmon liikkuvuus on suurin vaikuttaja pelin kulkuun ja siihen, millaisen kuvan pelaaja tulee muodostamaan pelaajahahmosta. Pelaajan liikkumiskykyihin kuuluu kyky kävellä ja/tai juosta vasemmalle tai oikealle ja hyppiminen. Nykyään pelaajalle annetaan muitakin liikkumiskykyjä hyödynnettäväksi, joita pelaaja avaa pelin edetessä. Näistä yleisimpiä ovat tuplahyppy ja syöksy. [4.]

Oikean maailman fysiikoilla on suuri vaikutus genren liikkumisessa. Tasohyppelypelit hyödyntävät useita fysiikan laskukaavoja, kuten painovoimaa ja kiihtymistä, luodakseen luonnollisen liikkeen pelaajahahmolle. [2.]

2.2.2 Pelin ympäristö

Pelaajahahmon liikkumista on täydentämässä pelin maailma. Tasohyppelypelien maailmat ovat useimmiten erittäin interaktiivisia ja sisältävät monia kenttäelementtejä, jotka auttavat tai vaikeuttavat pelaajan liikkumista kentässä. [2.] Esimerkkinä pelaajaa elementeistä voisi olla liikkuvat tasot tai tikkaat, joita pelaaja voi hyödyntää liikkuessaan eteenpäin kentässä. Esimerkkinä taas pelaamista vaikeuttavia elementeistä olisi piikkiansat tai pohjattomat kuilut. [4.] Mitä pidemmälle pelissä edetään, sitä enemmän vaikeuttavien elementtien määrä kasvaa kentissä, mikä luo haastavampia kenttiä. [23.]

2.3 Tärkeitä tasohyppelypelejä

Seuraavassa kappaleessa käydään läpi muutama tasohyppelypeli, jotka ovat vaikuttaneet genreen kokonaisuutena. Kohtaan valittiin neljä eri peliä eri aikakausilta. Pelien vaikutus genreen on voinut olla uusien mekaniikkojen takia tai uuden teknologian hyödyntäminen. Listassa on 2D- ja 3D-tasohyppelypelejä.

2.3.1 Donkey Kong

Donkey Kong julkaistiin 1981 kolikkopelinä pelihalleihin, ensiksi Japanissa ja myöhemmin Pohjois-Amerikassa. Peli oli Nintendon aikaisimpia yrityksiä päästä Amerikan markkinoille ja sanoa, että Donkey Kong onnistuisi tehtävässään, olisi vähättelyä. Kahden ensimmäisen vuoden aikana Donkey Kong -pelikabinetteja myytiin noin 67 000 kappaletta Amerikassa ja vielä enemmän Japanissa. [24.]

Pelin päätehtävä on ohjata pelin protagonistia, Jumpmaniä, pitkin teräspalkkeja pelastamaan tyttöystävänsä, Paulinen, jonka gorilla nimeltään Donkey Kong on kidnapannut. Peli ei kuitenkaan ole vain kiipeilyä, mutta pelaajan tulee samalla väistää Donkey Kongin heittämiä tynnyreitä hyppäämällä niitten yli. [25.]

Donkey Kong on genrelle ja videopeleille yleensäkin tärkeä peli, koska se oli ensimmäinen tasohyppelypeli, joka menestyi suunnattomasti. Sen lisäksi Donkey Kong oli ensimmäisiä pelejä, joissa pelin tarina oli hyvin kerrottu ja pelaajille se oli selkeä ja helppo ymmärtää heti pelin alkaessa. [9, s.3.] Kuvassa 3 näkyy pelin ensimmäinen kenttä.

Kuva 3. Donkey Kong-pelin ensimmäinen kenttä. [24.]

2.3.2 Super Mario Bros.

Pohjois-Amerikassa vuonna 1983 videopelilaman jälkeen videopelimarkkinat olivat vailla uusia, parempia kotikonsoleita. Videopelien liikatuotanto, pelikonsoleitten korkeat hinnat ja pelien vaihteleva laatu ovat vain muutamia syitä videopelilamalle. [26.] Nintendo sillä välin oli julkaissut uuden kotikonsolinsa Japanissa, Famicomin, jonka Nintendo halusi tuoda myös Amerikan markkinoille. Nintendo joutui muokkaamaan Famicomin sopivaksi Amerikan markkinoille. [27.]

Nintendo Entertainment System, tunnetaan myös nimellä NES, julkaistiin Amerikassa vuonna 1985, ja sen mukana julkaistiin useita klassikkopelejä, joista yksi oli Super Mario Bros. Super Mario Bros oli osana kahta erilaista pakettia, jossa itse konsoli myytiin. [28.] Tämä teki Super Mario Brosista konsolin hittipelin. Super Mario Bros -peliä on myyty noin 40 miljoonaa kappaletta. [29.] Pelissä pelaaja pääsee ohjaamaan Marioa, kenenkä tehtävä on pelastaa prinsessa Peach, jonka pahamaineinen Bowser on kidnapannut [30].

Super Mario Brosia pidetään tärkeänä pelinä monen syyn takia. Pelin musiikki ja taide olivat ajalleen erinomaiset ja pelaajanhahmon ohjattavuus oli jotain, mitä ei voinut verrata mihinkään muuhun. [31.] Sisältäen kiihdyttämisen ja hyppykorkeuden määrittämisen [32], Super Mario Bros on tähän päivään asti yksi parhaiten toteutetuista tasohyppelypeleistä. [31.] Kuvassa 4 näkyy yksi pelin monista salaisista alueista.

Kuva 4. Super Mario Bros -pelin salaisesta alueesta. Vasemmalla näkyy Mario. [31.]

2.3.3 Super Mario 64

Konsolien siirtyessä hyödyntämään 32-bittistä ja 64-bittistä teknologiaa, monia uusia ovia avautui pelinkehittäjille. Varmaan vaikuttavin ajalleen oli 3D-grafiikkojen toteuttaminen videopeleihin. Tämä johti siihen, että suurin osa uusista peleistä tehtiin 3D-pelejä. Tasohyppelypelit tulivat hieman jäljessä, mutta nekin siirtyivät 2D:stä 3D:hen. [4.] 1996 julkaistua Super Mario 64:ä pidetään 3D-tasohyppelypelinä, joka suuntasi muita 3D-tasohyppelypelejä oikeaan suuntaan, niin kuin Super Mario Bros teki aikanaan 2D-tasohyppelypeleille. [33.]

Pelissä pelaaja pääse jälleen kerran ohjaamaan Marioa, joka löytää prinsessa Peachin linnan autiona. Toad kertoo Mariolle, että Bowser on jälleen kidnapannut prinsessan ja piilottanut 120 voimatähteä erillisiin maailmoihin, jotka ovat maalausten takana. Marion on kerättävä voimatähdet ja pelastettava prinsessa. [34, s. 4–5.]

Super Mario 64:sen mekaniikkoja ja pelin sulavuutta ei voinut verrata muihin ajan 3D-tasohyppelypeleihin. [35.] Mariolla oli useita liikkumiskykyjä, kuten liukuminen, tolppien kiipeäminen ja tietenkin hyppiminen. Mutta hyppyjäkin oli monia erilaisia: pituushyppyjä, seinähyppyjä, takaperin- ja sivuttain kuperkeikkoja. [34, s. 6–7.] Itse Marion liikuttaminenkin oli ajalleen mullistavaa, kiitokset Nintendo 64:n ohjaimessa olevalle analogiselle sauvaohjaimelle. Tämä antoi pelaajalle paremman mahdollisuuden tehdä tarkkoja pieniä liikkeitä, mutta samalla myös helpotti nopeitten täyskäännösten tekemistä. [35.] Kuvassa 4 nähdään pelin ensimmäinen kenttä.

Kuva 5. Super Mario 64 -pelin ensimmäinen kenttä, Bob-Omb Battlefield [36.]

2.3.4 Celeste

Celeste on Extremely OK Gamesin kehittämä tasohyppelypeli, joka julkaistiin 2018 PC:lle, Playstation 4, Xbox Onelle ja Nintendo Switchille. [37.] Peli nousi nopeasti suosioon ja monen mielestä peli oli juuri se, mitä genre oli kaivannut pitkään. Pelin tarina, taide, musiikki ja varsinkin pelattavuus auttoivat pelin nousussa suosioon. [38.] Pelissä pelaaja ohjaa Madeliniä, kuka on päättänyt kiivetä mystisen Celeste-vuoren huipulle [39].

Madelin kykenee perusliikkumisen lisäksi kiipeämään seiniä pitkin ja syöksymään. Kiivetessä seinää pitkin, pelaaja voi kiivetä seinää hitaasti vai nopeasti ja lopettaakseen seinäkiipeilyn he voivat joko pudota suoraan alas tai tehdä seinähypyn. Syöksyminen on pelaajan toinen työkalu kiivetessä. Syöksyminen on paljon kaoottisempi liikkumisvaihtoehdoista, sillä pelaaja menettää ohjat Madelinistä hetkellisesti. [40.]

Tasohyppelypeli genren suosion ollessa pieni, Celesten menestys oli merkki monille pelinkehittäjille, varsinkin indie-kehittäjille, että genren peleille on vielä kysyntää. Celestessä toteutettiin hyvin pelaajan liikkuminen ja ohjattavuus. [38.] Pelin haastavuutta helpottaa pelin anteeksiantavaisuus pelaajalle usealla erilaisella mekaniikalla. Näistä mekaniikoista tunnetuimmat ovat kojoottiaika ja hyppypuskurointi. Molemmat näistä mekaniikoista tekevät hyppimisestä toiminnon, jota pelaajan ei tarvitse toteuttaa niinkään tarkasti. [40.] Kuvassa 5 näkyy pelin ensimmäisen alueen alkua.

Kuva 6. Kuvankaappaus Celeste-pelin ensimmäisestä haasteesta.

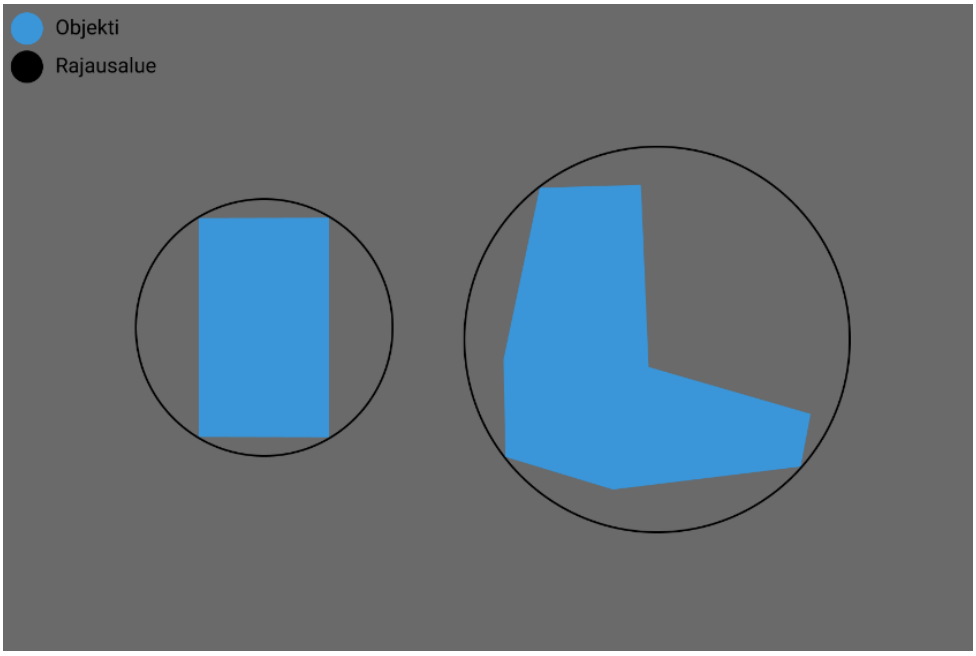
3 Tasohyppelypelien mekaniikat ja käytänteet

Seuraavissa kappaleissa käydään läpi tasohyppelypeleille ominaisia mekaniikkoja ja käytänteitä. Kyseisiä mekaniikkoja ja käytänteitä hyödynnetään molemmissa 2D- ja 3D-tasohyppelypeleissä. Tässä tekstissä tutustutaan tarkemmin, miten ne toimivat 2D-tasohyppelypeleissä.

3.1 Törmäyshavaitseminen

Oikeassa maailmassa kirja ei putoa pöydän läpi, koska fysiikan lait estävät sitä tapahtumasta. Sen sijaan kirja lepää pöydällä, vaikka maan painovoima vetää sitä puoleensa. Tämä sama efekti luodaan videopeleihin hyödyntämällä törmäyksiä ja törmäysalueita. Videopeleissä vuorovaikutuksia objektien välillä tapahtuu suurimmaksi osaksi pelaajahahmon kanssa, esimerkiksi kun pelaajahahmo laskeutuu maahan tai kävelee kerättävään kolikkoon. Törmäys tapahtuu, kun kaksi tai useampi törmäysalue leikkaavat toisensa. [41.]

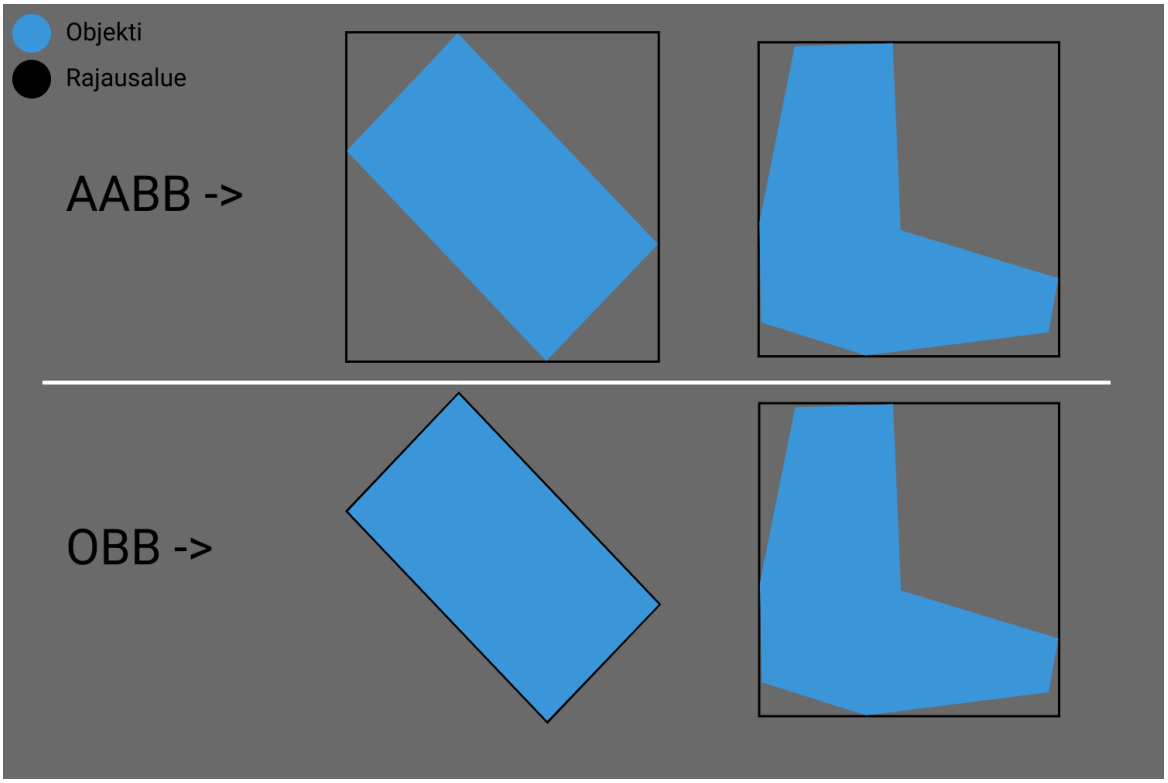
Yksinkertainen ratkaisu tähän olisi luoda ympyrä, joka ympäröi objektin kokonaan luodessaan rajausympyrän. Tämä toimii erinomaisesti ympyränmuotoisille objekteille, mutta voi myös johtaa törmäysalueeseen, joka ei kuvaa objektin muotoa hyvin. [42, s. 486–489.] Kuvassa 7 näkee, kuinka objektin törmäysalue voi olla suurempi kuin itse objekti visuaalisesti.



Kuva 7. Rajausympyrät kärsii tarkkuudessa, jos objekti ei ole pyöreä.

Usein käytetty ratkaisu törmäysalueen luomiseen on tehdä objektille akselin mukainen rajausalue (engl. Axis aligned bounding box, AABB). AABB on objektin ympärille luotu neliö, jonka sivut ovat yhdensuuntaiset maailmanakselien kanssa. [42, s. 486–489.] AABB on hieman parempi luomaan tarkempia törmäysalueita objekteille, jotka eivät ole ympyröitä, mutta sillä on myös omat huonot puolensa. AABB:tä ei voida kääntää, joten objektin grafiikan kääntyessä sen törmäysalue pysyy samana. AABB voi myös luoda suurempia tai vääristyneitä törmäysalueita riippuen objektin muodosta. [42, s. 486–489; 43.]

AABB:n huonoista puolista huolimatta se on silti hyödyllinen, kiitos sen hyvien puolien. AABB on yksinkertainen, minkä takia sen luominen on nopeaa. AABB-törmäysalueen testaaminen toisen AABB-törmäysalueen kanssa on myös erittäin simppeliä ja se voidaan optimoida tehokkaasti. [43.] Jos haluttaisiin luoda törmäysalue, jota voitaisiin kääntää pelinaikana, silloin tulisi hyödyntää käännettävää rajausaluetta (engl. Oriented bounding box, OBB). [42, s. 497–499.] Kuvassa 8 visualisoidaan AABB:n ja OBB:n eroja.



Kuva 8. AABB muodostaa epätarkkoja rajausalueita, jos objekti on kulmassa tai kovera. OBB ei kärsi tarkkuudessa käännettyjen objektien kanssa, mutta kärsii silti koverien objektien kanssa.

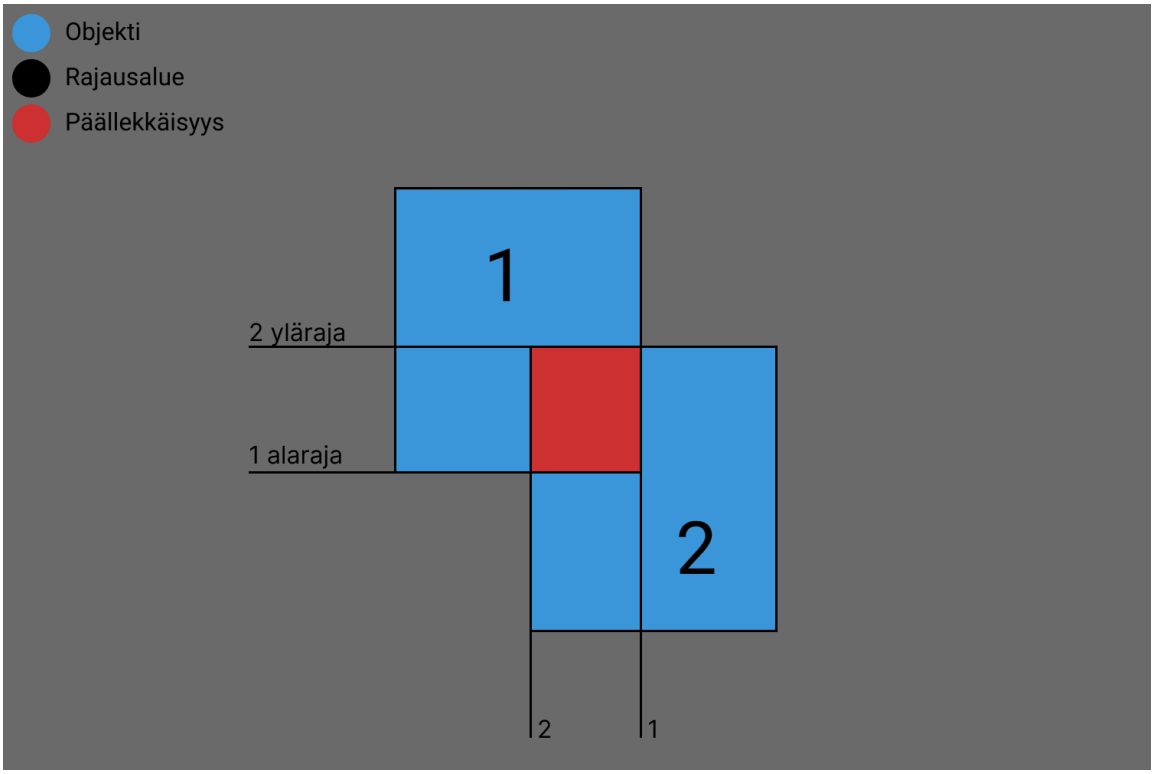
3.2 AABB

AABB-rajausalueitten implementointi on simppeliä, koska niitten toimimiseen vaaditaan vain muutama muuttuja ja hieman logiikkaa. 2D-pelissä tarvitaan vain tietää objektin x- ja y- piste maailmassa ja sen pituus ja korkeus. Näitten avulla voidaan luoda objektille törmäysalue, joka suurin piirtein kuvastaa objektia. [44, 45.]

Jos halutaan luoda AABB:llä paremmin objektia kuvastavamman törmäysalueen, niin voimme jakaa objektin useampaan osaan, ja luoda niille omat AABB-törmäysalueet. Tätä tekniikkaa hyödynnetään yleisesti 2D-tastelupeleissä, joissa törmäysalueitten halutaan olevan mahdollisimman lähellä hahmon muoto. Tämä ratkaisee yhden AABB:n isoimmista ongelmista, mikä on epätarkkuus kompleksien muotojen kanssa. Tosin objektin muodon täydellinen muodostaminen vaatii useita AABB-törmäysalueita. Kuvassa 9 nähdään esimerkki tästä.

Kuva 9. Kuvankaappaus Super Street Fighter 2 -pelistä, jossa törmäysalueet näytetään modia hyödyntäen. [46.]

AABB-törmäysalueitten keskenään testaaminen on laskennallisesti kevyttä, joten se toimii hyvin tilanteissa, joissa on vähän keskenään testattavia objekteja, mutta kärsii kun objekteja on paljon. Kahden AABB:n testaaminen perustuu kahden objektin minimi- ja maksimiarvojen vertaamiseen. [44.] Kuvassa 10 visualisoidaan kahden objektin päällekkäisyyttä. Kuvassa 11 algoritmi on kirjoitettu pseudokoodina.



Kuva 10. Kuvassa objekti 1 ja 2 ovat päällekkäin. Tämä todistetaan vertailemalla objekti 2:n yläreunaa ja objekti 1:n alareunaa. Sama tulos todetaan myös sivuilla.

```
function TörmäysTarkistus(objekti A, objekti B)
    if A oikeareuna < B vasenreuna ja // tarkistetaan onko A:n ja B:n välillä päällekkäisyyttä
       A vasenreuna > B oikeareuna ja
       A alareuna < B yläreuna ja
       A yläreuna > B alareuna
        RatkaiseTörmäys(A, B) // objektien välillä on päällekkäisyyttä
    endif
endfunction
```

Kuva 11. Pseudokoodi funktiosta, jossa tarkastetaan kahden AABB:n omaavan objektin päällekkäisyys.

3.3 Pelaajahahmon mekaniikat ja käytänteet

Tasohyppelypeligenrelle on useita genrekohtaisia mekaniikkoja ja käytänteitä, jotka ovat vielä käytössä suurimmalta osin tasohyppelypeleissä. Iso osa näistä mekaniikoista ja käytänteistä liittyy pelaajahahmon ohjattavuuteen ja vaikuttaa paljon siihen, miltä peliä tuntuu pelata. [47.]

Seuraavissa kappaleissa käydään läpi muutamia tasohyppelypelien pelaajahahmoille yleisiä mekaniikkoja ja käytänteitä. Kappaleissa perehdytään, miten mekaniikka/käytänne toimii, miksi niitä hyödynnetään ja miten ne voitaisiin implementoida.

3.4 Hyppiminen

Hyppiminen on genrelle miltei keskeisin mekaniikka, joten siihen liittyy paljon genrelle ominaisia käytänteitä. Ennen kyseisten käytäntöjen ja mekaniikkojen läpikäyntiä, olisi hyvä käydä läpi, mikä tekee hypystä mukavamman pelaajalle.

Voidaan kuvitella, että hyppy muodostaa paraabelin, joka aukeaa painovoiman suuntaan. Paraabelin voidaan muodostaa vastaavalla toisen asteen yhtälön kaavalla:

$$\boldsymbol{f(x) = ax^2 + bx + c} \qquad (1.)$$

f(x) on pelaajahahmon korkeus verrattavissa maahan, ***x*** on aika, joka on kulunut, ***a*** on painovoima ja ***b*** on hypyn alkunopeus. ***a***:n arvo tulee olla negatiivinen, jotta paraabeli on alaspäin aukeava. Tietämällä eri arvojen merkityksen kaavassa kaava voidaan kirjoittaa uudelleen vastaavasti:

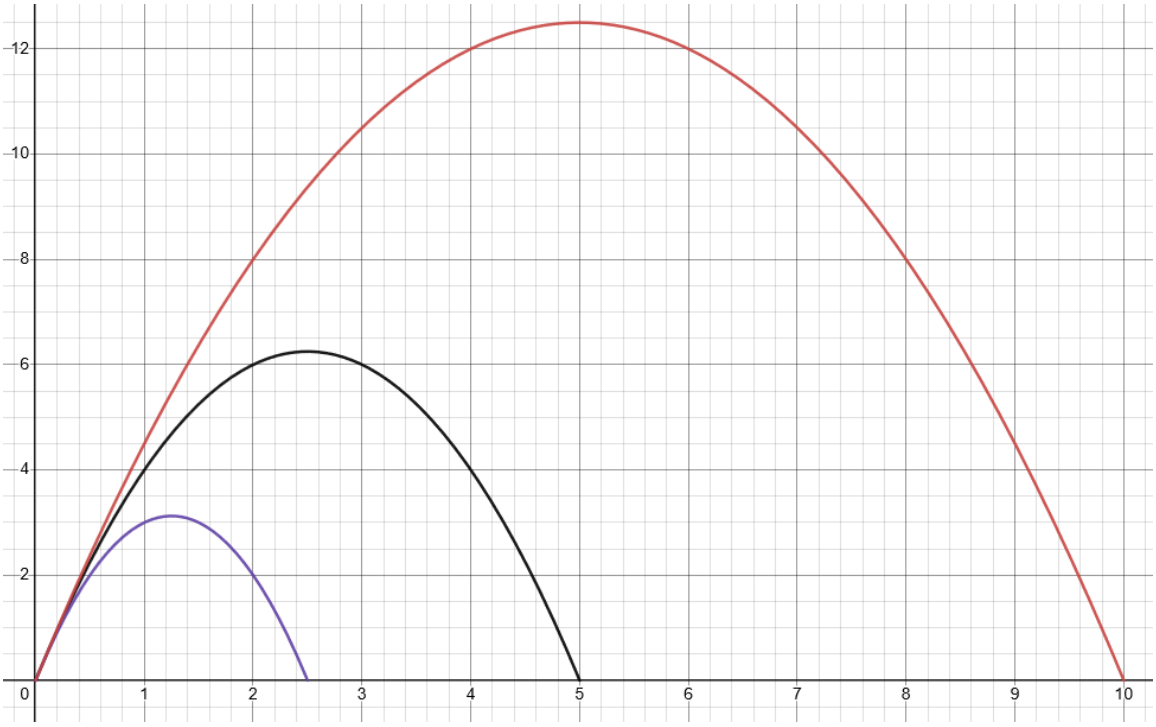
$$\boldsymbol{y = gt^2 + vt} \qquad (2.)$$

Ainoa arvo, joka otettiin kaavasta 1 pois, oli ***c***, koska sen arvo vaikuttaa paraabelin y-sijaintiin. Tässä tapauksessa tätä ei haluta, joten se jätetään pois uudesta kaavasta 2 kokonaan.

Painovoima vaikuttaa siihen, kuinka nopeasti pelaajahahmo palaa takaisin maahan hypättyään. Vaihtamalla painovoiman arvoa paraabelin leveys kasvaa tai pienenee. Alkunopeus taas vaikuttaa paraabelin korkeuteen; mitä pienempi arvo, sitä matalampi paraabeli on ja mitä isompi arvo, sitä korkeampi paraabeli on.

Vaihtamalla näitä kahta arvoa voidaan luoda erilaisia hyppyjä: Pieni painovoima ja iso hyppynopeus johtaa isoon, kevyen tuntuiseen hyppyyn. Sillä välin, jos painovoima on suuri ja hyppynopeus

on pieni, niin hahmo tuntuu paljon painavammalta ja raskaalta. [48.] Kuvassa 12 nähdään kolme eri paraabelia, jotka on piirretty kaavalla 2. Kuvassa 13 nähdään Super Mario Bros -pelissä oleva hyppy ja sen luoma paraabeli.



Kuva 12. Kaavan 2 avulla piirrettyjä paraabeleja. Punaisen paraabelin arvot: ***g*** = -0.5 ja ***v*** = 5. Mustan paraabelin arvot: ***g*** = -1 ja ***v*** = 10. Violetin paraabelin arvot: ***g*** = -2 ja ***v*** = 5

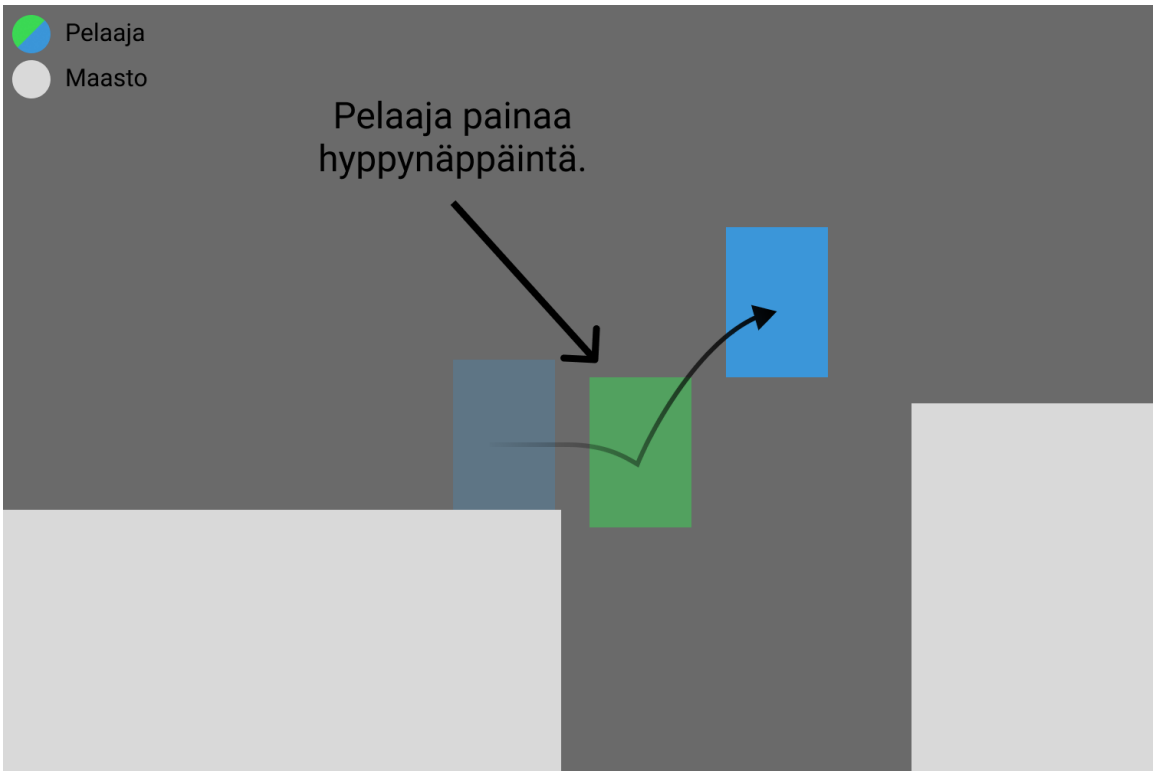
Kuva 13. Super Mario Bros -pelissä Marion hyppy nousee huippuunsa hitaasti, mutta laskeutuu nopeasti. Hyppy kestää noin 0,8 sekuntia. [47.]

3.4.1 Kojoottiaika

Kojoottiaika (engl. Coyote time) on mekaniikka, joka antaa pelaajalle hieman enemmän pelivaraa tehdessä hyppyjä tasojen reunoilta. [47.] Mekaniikan nimi tulee Kelju K. Kojootti ja Maantiekiitäjä -animaatiosarjasta, jossa hahmot jäävät hetkeksi leijumaan paikalleen ilmaan, kunnes tekojen seuraamukset tapahtuvat. [49.] Mekaniikan nimi kuvastaa miltei täydellisesti sen toiminnon.

Kojoottiaika tarkoittaa aikaa, jolloin pelaaja pystyy hyppäämään, vaikka pelaajahahmo ei kosketa maata. Mekaniikka nähdään parhaiten toiminnassa, kun pelaaja joutuu hyppäämään leveän rotkon yli, jolloin pelaajat yleensä jättävät hyppäämisen viime hetkeen. [47.]

Kojoottiajan ansiosta pelaaja ei rankaista, jos he painoivat hyppynäppäintä vähän liian myöhään. Tämä antaa pelaajalle hieman lisää pelivaraa toteuttaa hyppy, mikä tuntuu reilummalta kuin se, että pelaaja painaisi hyppynäppäintä sekunnin murto-osan liian myöhässä ja putoaisi pohjattomaan kuiluun. [50.] Kuvassa 14 nähdään, miten kojoottiaika toimii käytännössä.



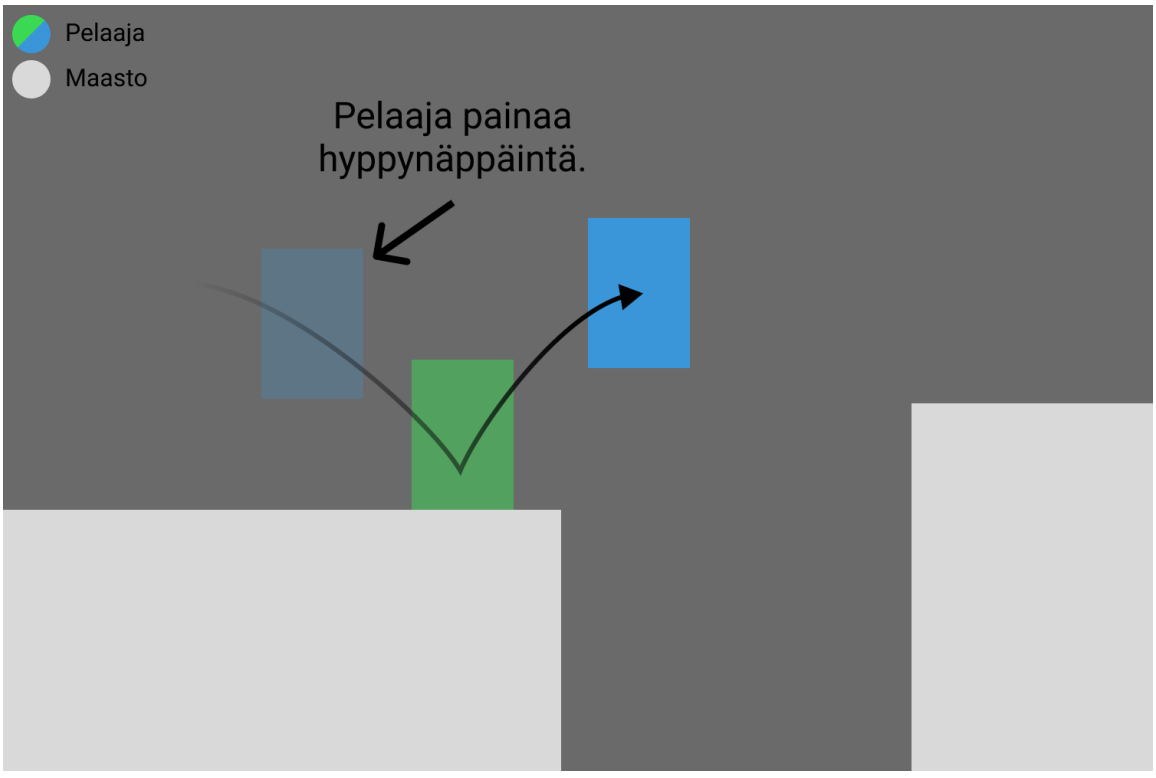
Kuva 14. Kojoottiajan toiminnallisuus pelissä. Pelaaja painaa hyppynäppäintä hieman sen jälkeen, kun hän kävelee reunalta, mutta hyppy tapahtuu silti.

3.4.2 Hyppypuskurointi

Hyppypuskurointi (engl. jump buffering) on hyvin samanlainen mekaniikkana kuin kojoottiaika siinä mielessä, että molemmat mekaniikat antavat pelaajalle enemmän pelivaraa hyppyjen toteuttamiseen. [47.] Mekaniikan nimi, hyppypuskurointi, viittaa mekaniikan toimintaan, jossa pelaajan hyppysyöte säilytetään aktiivisina määritetyn ajan. [51.]

Hyppypuskurointi, kuten kojoottiaika, auttaa pelaajaa toteuttamaan hyppyjä tietyissä tilanteissa. Hyppypuskurointi auttaa pelaajaa tekemään hyppyjä juuri samalla hetkellä, kun pelaajahahmo koskettaa maata. [51.]

Kuten kojoottiajan avulla, hyppypuskuroinnin ansiosta pelaajaa ei rankaista, jos he eivät saa ajoitettua hyppyä täydellisesti. Tämä tekee pelistä reilumman, koska samalla hetkellä hyppääminen, kun pelaajahahmo osuu maahan, jos peli on 60 FPS, tarkoittaisi 0,016 s:n täydellistä ajoitusta. Samalla hyppypuskurointi saa pelin tuntumaan paljon reagoivammalta. [51.]



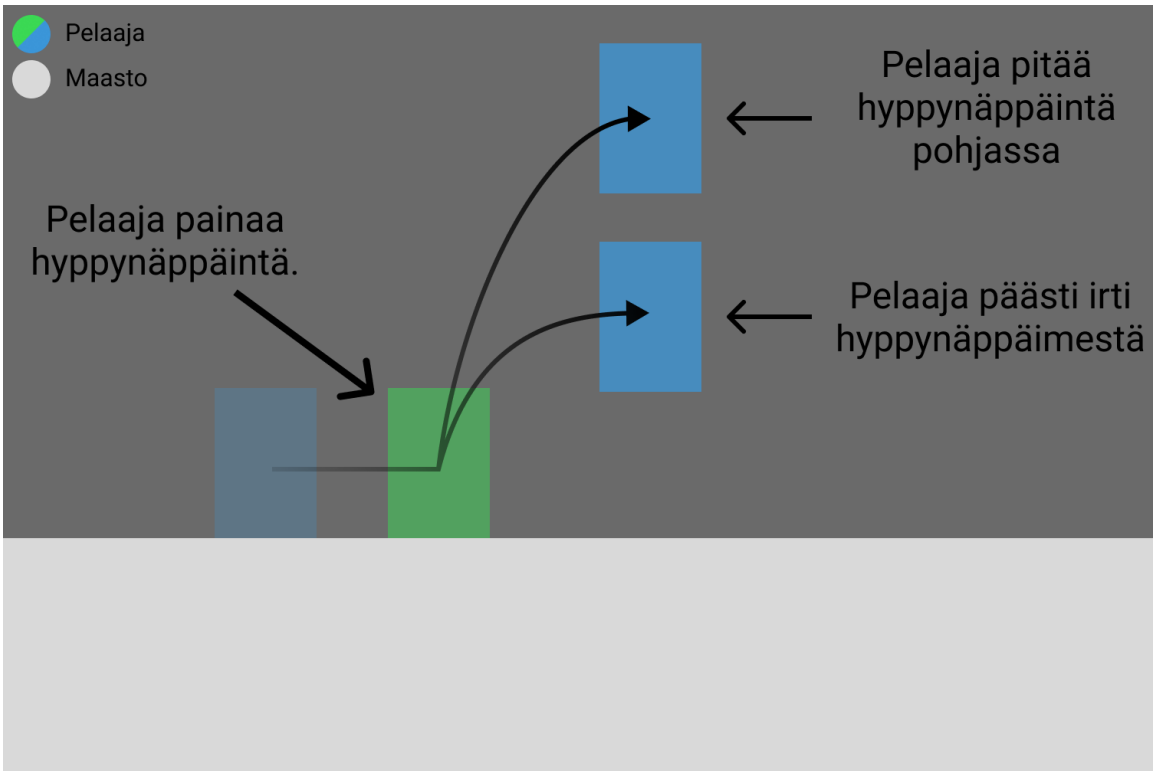
Kuva 15. Hyppypuskuroinnin toiminnallisuus pelissä. Pelaaja painaa hyppynäppäintä jo ilmassa, mutta hyppy tapahtuu silti, kun pelaajahahmo koskettaa maata.

3.4.3 Hallittava hyppykorkeus

Hallittava hyppykorkeus (engl. variable jump height) on mekaniikka, joka antaa pelaajalle mahdollisuuden vaikuttaa pelaajahahmon hypyn korkeuteen. Mekaniikka tunnetaan myös muutamalla muulla nimellä, mutta hallittava hyppykorkeus on yleisin nimitys. [47, 52.]

Hallittava hyppykorkeus vaikuttaa pelaajan hyppynapin toimintaan; jos pelaaja pitää hyppynäppäintä pohjassa, niin pelaajahahmo hyppää korkealle, jos pelaaja napauttaa hyppynäppäintä, tekee pelaajahahmo pienen hypyn. Tämän toiminnan lisäksi on myös yleinen lisäys nopeuttaa pelaajahahmon putoamista, jos hän tekevät lyhyen hypyn. [52.]

Mekaniikan tavoitteena on antaa pelaajalle enemmän hallintaa siitä, kuinka korkeita hyppyjä he voivat tehdä. Tämän avulla pelaaja voi tehdä lyhyitä, tarkkoja hyppyjä ja pitkiä hyppyjä isojen esteitten yli. [52.] Tämä antaa myös kenttäsuunnittelijoille mahdollisuuden luoda kenttiä ja alueita, joissa pelaajan tulee hyödyntää mekaniikkaa päästäkseen kentän loppuun. [47.]



Kuva 16. Hallittavan hyppykorkeuden toiminnallisuus pelissä. Vihreällä värillä kuvattu neliö kuvastaa pelaajahahmoa sillä hetkellä, kun hyppy aloitetaan.

3.4.4 Ohjattavuus ilmassa

Suunnan vaihtaminen ilmassa ei ole realistisesti mahdollista, mutta tasohyppelypeleissä tämä on yleinen mekaniikka. Se, miten paljon pelaaja voi vaikuttaa hahmon suuntaan ilmassa vaihtelee pelistä peliin riippuen siitä, minkälainen pelikokemus halutaan pelaajalle luoda. [53.] Pelaajahahmon vauhti oikealle päin voi vaikuttaa kääntymiseen vasemmalle, jolloin suunnan vaihtuminen tapahtuu hitaammin. Vaihtoehtona on myös se, että pelaajalla ei ole mahdollisuutta vaikuttaa pelaajahahmon suuntaan ilmassa. [54.]

Arvo, joka voidaan lisätä pelaajan liikkeeseen ilmassa ja joka vaikuttaa ohjattavuuteen, on ilmakiihtyvyys. Ilmakiihtyvyys ei saisi olla liian suuri tai pieni; Jos ilmakiihtyvyys on liian suuri, niin pelaajan on vaikeampaa laskeuta haluttuun sijaintiin. Jos taas ilmakiihtyvyys on liian pieni, niin pelaajan on erittäin vaikea tehdä pieniä korjauksia. [54.]

Ilmakiihtyvyys vaikuttaa pelaajan kontrolleihin suoraan, joten arvon saaminen juuri oikeaksi peliin on tärkeää. Miten paljon pelaaja pystyy kontrolloimaan hahmoaan ilmassa vaikuttaa myös siihen minkälaisen kuvan hahmo antaa pelaajalle. [53, 54.] Kun pelaajalle annetaan hieman, mutta ei täyttä, hallintaa hahmon liikkeeseen ilmassa, se lisää hahmon painon tunnetta, mutta samalla se auttaa pelaajaa, koska hän voi keskittyä paremmin pelin muihin osiin. [56]

3.5 Painovoima

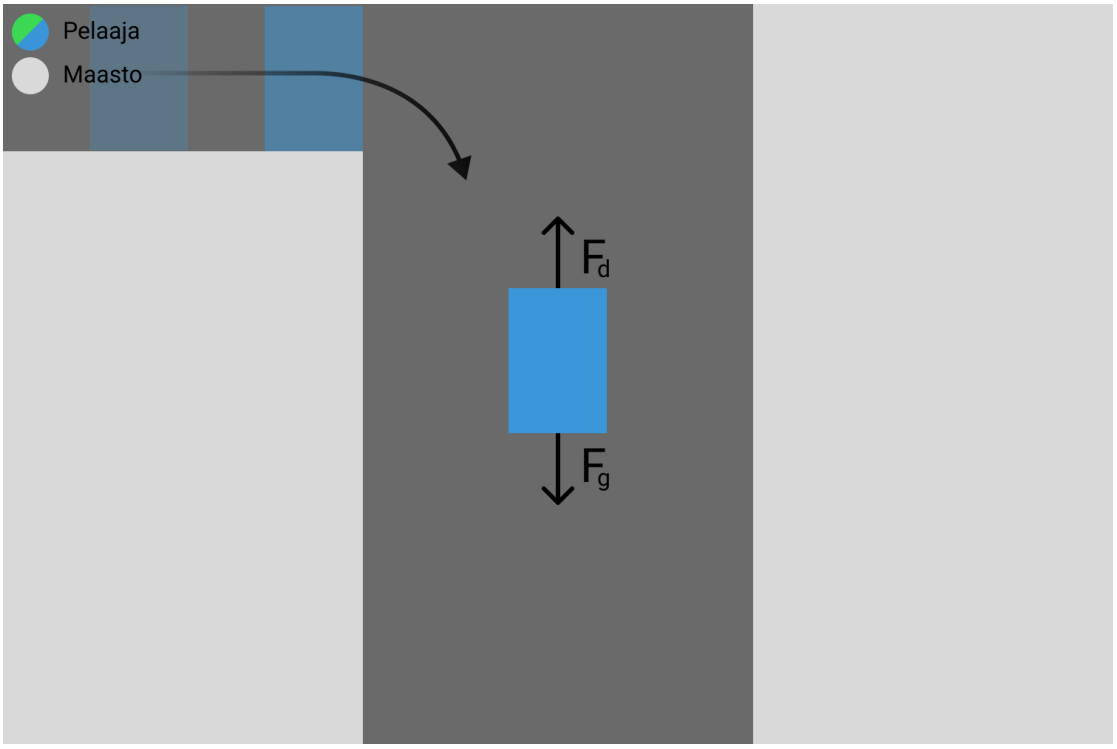
Oikeassa maailmassa painovoima on voima, joka vaikuttaa jokaiseen objektiin universumissa; jokainen objekti vetää toisiaan puoleensa. Maapallon pinnalla tämä ilmiö nähdään, kun esimerkiksi pudotetaan kirja pöydältä. Maapallon painovoima vetää kirjaa puoleensa ja kirja putoaa lattialle. [57.] Painovoima toteutetaan videopeleihin simppelimmällä menetelmällä. Painovoima 2D-tasohyppelyssä on voima, joka vetää pelaajahahmoa näytöllä alaspäin, kun se on ilmassa.

Tasohyppelypeleissä painovoima vaikuttaa erittäin paljon siihen, miltä pelaajahahmon hyppy tuntuu. Useissa tasohyppelypeleissä, kuten Super Mario -pelisarjassa, Mario kykenee hyppäämään useita kertoja oman pituutensa verran ylöspäin. [56.] Tämä voisi johtaa päätökseen, että painovoima Super Mario -peleissä on pieni, mutta se on juuri toisinpäin. Super Mario Bros -pelissä painovoima on 9 kertaa voimakkaampi kuin maanpinnalla. [58.]

3.6 Lukittu putoamisnopeus

Lukittu putoamisnopeus perustuu oikean maailman ilmiöön, rajanopeuteen. Rajanopeus tarkoittaa tilaa, jossa putoavan objektin ilmanvastus kumoaa maapallon painovoiman vetovoiman, jolloin putoava objektin putoamisvauhti on saavuttanut rajanopeuden. [59.] Kuvassa 17 nähdään esimerkki, jossa nähdään putoavaan objektiin vaikuttavat voimat.

Lukitulla putoamisnopeudella imitoidaan rajanopeutta videopeleissä. Tosin ilmanvastuksen lisääminen ja voimien laskeminen veisi laskentatehoa, jota voitaisiin hyödyntää muualla pelissä. Sen sijaan voidaan asettaa maksimiputoamisnopeuden, jonka saavutettua pelaajahahmo alkaa putoamaan tasaista vauhtia. Mekaniikan ansiosta pelaajilla on enemmän aikaa reagoida asioihin, kun he putoavat alaspäin. [47.]



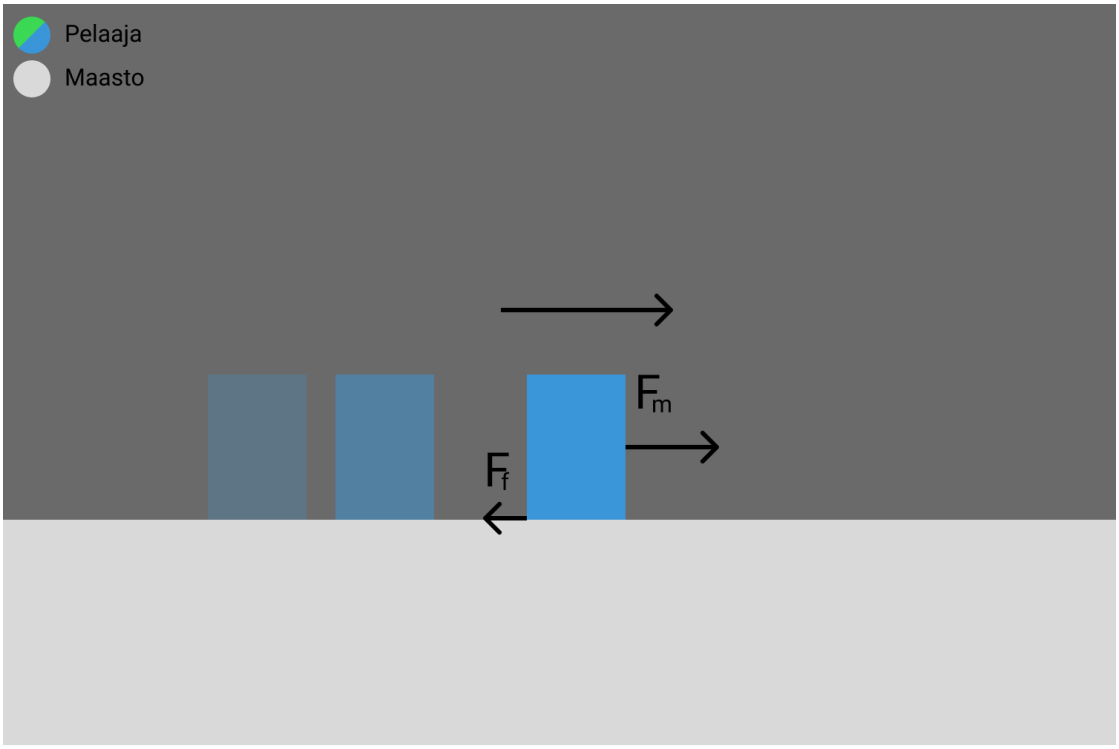
Kuva 17. Kuvassa nähdään voimat F_d ja F_g, jotka kuvastavan ilmanvastusta ja painovoimaa.

3.7 Kiihtyvyys ja kitka

Kiihtyvyys on määrää, jolla objektin nopeus muuttuu ajassa. Koska nopeudella on suuruus ja suunta, kummankaan muuttujan muutos johtaa kiihdyttämiseen. Joten kiihtyvyys on muutos objektin nopeudessa, suunnassa tai molemmissa. Kitka taas on voima, joka vastustaa kahden toistensa pintojaan koskettavien kappaleitten välistä liikettä. [59.] Kuvassa 18 nähdään kiihtyvyyden ja kitkan vaikutus pelaajahahmon liikkeeseen.

Kiihtyvyys ja kitka ovat yleisiä fysiikan ilmiötä oikeassa maailmassa, joten on ymmärrettävää, että kyseiset ilmiöt voidaan löytää videopeleistä, varsinkin tasohyppelypeleistä. Sen sijaan, että pelaajahahmo lähtee heti juoksemaan maksiminopeutta samalla sekunnilla, kun pelaaja painaa oikealla, hän joutuu kiihdyttämään saavuttaakseen maksiminopeuden. Samoin kun pelaaja lopettaa ohjaamisen, pelaajahahmo ei hidastu heti, vain pikkuhiljaa kitkan takia. [50.]

Mekaniikat myös vaikuttavat pelaajan kokemukseen ja kenttäsuunnitteluun. Esimerkiksi paljon tilaa pelaaja tarvitsee, jotta hän voisi juosta maksiminopeutta ja ylittää rotkon pitkällä hypyllä. Liian hidas kiihtyminen ja hidastuminen voivat johtaa huonosti kontrolloitavaan hahmoon, mutta taas liian nopea kiihdytys ja hidastus eivät lisää sitä, mitä mekaniikoilta haetaan. [53.]



Kuva 18. Pelaajahahmo liikkumassa vasemmalle. Kuvassa nähdään voimat F_f ja F_m, jotka kuvastavat kitkaa ja liikkumisvoimaa tässä järjestyksessä.

4 2D-pelihahmon mekaniikkojen toteuttaminen

Käytännön osuudessa luotiin tasohyppelypeli demo, jossa implementoitiin tekstissä aiemmin mainitut mekaniikat ja käytänteet. Projekti toteutettiin C++ -ohjelmointikielellä käyttäen OpenGL:ää renderöimiseen. Projektin toteutus antaa tilaisuuden näyttää, miten 2D-tasohyppelypeli voitaisiin toteuttaa ilman valmiin pelimoottorin hyödyntämistä ja selittää, miten pelimoottoreissa yleiset ominaisuudet, kuten törmäystarkistus, toimivat käytännössä.

4.1 Visual Studio 2022

Visual Studio on Microsoftin kehittämä ohjelmointiympäristö. Visual Studio on laaja ohjelmointiympäristö, joka tukee useita ohjelmointikieliä ja sisältää useita ominaisuuksia. Toteutuksessa hyödynnetään Visual Studio Community 2022 -versiota, joka on uusin versio ja tukee C++ -ohjelmointikieltä. [60.]

4.2 CMake

CMake on työkalu, joka automatisoi ohjelman käännösympäristön generoimisen haluttujen asetusten mukaisesti. CMake ei itse käännä koodia, vaan jättää sen rakennusympäristön tehtäväksi. Projektissa CMakea hyödynnetään kirjastojen lisäämisessä projektiin ja helpottamaan projektin avaamista toisilla tietokoneilla. [61.]

4.3 OpenGL

OpenGL, eli Open Graphics Library, on ohjelmointirajapinta, jolla hyödynnetään laitteen grafiikkalaitteistoa, esimerkiksi tietokoneen grafiikkakorttia. Rajapinta sisältää funktiota, mikä mahdollistaa objektien piirtämisen laitteen näytölle. Projektissa OpenGL:ää hyödynnetään renderöimiseen. [62.]

4.4 Kolmannen osapuolen kirjastot

Kolmannen osapuolen kirjastot (engl. third-party libraries), ovat kokoelmia luokkia ja tai funktiota, joita voidaan lisätä projektiin tuomaan erilaisia toiminnallisuuksia. Projektissa hyödynnetään pääosin neljää eri kirjastoa: GLFW:ää, GLEW:tä, GLM:ää ja ImGui:a.

4.4.1 GLFW

GLFW on C-ohjelmointikielellä kirjoitettu kirjasto, joka toimii ohjelmointirajapintana OpenGL:ää. GLFW yksinkertaiset komennot esim. ikkunoitten luomiseen ja käyttäjän syötteen vastaanottamiseen. Projektissa GLFW:tä hyödynnettiin ikkunoitten luomiseen. [63.]

4.4.2 GLEW

GLEW, eli OpenGL Extension Wrangler Library, on laajennuslatauskirjasto joka ajonaikana päättää, mitkä OpenGL laajennukset toimivat kyseisellä alustalla. GLEW:tä hyödynnetään projektissa helpottamaan oikeiden laajennusten ylläpitämiseen ja oikeitten funktioitten kutsumiseen. [64.]

4.4.3 GLM

GLM, eli OpenGL Mathematics, on matematiikkakirjasto, joka on suunniteltu OpenGL ja sen varjostin-ohjelmakieli, GLSL, mielessä. GLM sisältää matemaattisia luokkia, kuten matriisit ja vektorit, ja funktiota, kuten näitten kahden kertominen. Projektissa GLM:ää hyödynnetään enimmäkseen renderöinnissä vaadittaviin matriisilaskuihin ja vektoriluokkia hyödynnetään esim. hahmon sijainnin seuraamiseen. [65.]

4.4.4 ImGui

ImGui on graafisen käyttöliittymän luomiseen tehty kirjasto C++-ohjelmointikielelle. Se on suunniteltu antamaan ohjelmoijille rajapinta, jolla he voivat nopeasti luoda käyttöliittymiä työkaluille,

datan visualisointia ja virheenjäljitystä varten. Projektissa ImGui:a hyödynnetään luomalla käyttöliittymä, jonka avulla voidaan muokata pelinarvoja ajon aikana. [66.]

4.5 Projektin rakenne

Ilman pelimoottoria, jonka arkkitehtuurin päälle projekti luotaisiin, tarvittavat osat implementoidaan itse. Projektissa käytetään OpenGL:ää, jota hyödyntäen saadaan piirrettyä pelin eri osat näytölle.

Toteutukseen arkkitehtuuriksi valittiin peliobjekti komponentti -systeemi. Molemmilla on omat pohjaluokat, jotka antavat näistä luokista periville luokille pohjatoiminnallisuuden, kuten päivitysfunktion. Komponenttiluokka myös tarvitsee tavan seurata, mikä objekti omistaa sen ja peliobjektiluokka tarvitsee tietää, mitkä komponentit kuuluvat sille.

Peliobjekti komponentti -systeemin avulla voimme helposti päivittää pelin tilaa. Isompien kokonaisuuksien, kuten syötteentarkistukseen ja törmäyshavaitsemiseen, siirretään omiin luokkiin, joita päivitetään myös pelin pääsilmukassa.

4.6 Törmäyshavaitseminen

Törmäyshavaitseminen toteutetaan hyödyntämällä AABB:tä, koska projektissa ei ole tarvetta pyörittää tai kääntää peliobjekteja ja pelin spritet ovat neliötä tai suorakulmioita. AABB on myös erittäin yksinkertainen algoritmi implementoida ja vaatii vähän laskentatehoa, mikä tekee siitä juuri sopivan vaihtoehdon toteutukseen.

4.6.1 Törmäysalueet

AABB-törmäysalue muodostuu sijainnista, korkeudesta ja leveydestä. Toteutus on 2D-peli, joten syvyysakselia ei tarvitse ottaa huomioon. Komponenttiluokka sisältää jo sijainnin, joten perivän luokan tulee vain tallentaa koko liukulukuvektorissa.

Tämän lisäksi lisätään vielä kaksi muuta liukulukuvektoria, jotka pitävät sisällään törmäysalueen maksimi- ja minimiarvot. Minimi ja maksimi selvitetään vähentämällä tai lisäämällä törmäysalueen sijaintiin puolet joko leveydestä tai korkeudesta. Kuvassa 19 nähdään AABB-luokan rakentaja.

```
AABB::AABB(Transform nTransform, Transform* nParent, glm::vec2 nSize, bool isStatic)
    : mSize(nSize), mStatic(isStatic), Component(nTransform, nParent) {
    mSize.x *= transform.GetScale()->x;
    mSize.y *= transform.GetScale()->y;

    min.x = transform.GetPosition()->x - mSize.x * 0.5f;
    min.y = transform.GetPosition()->y - mSize.y * 0.5f;
    max.x = transform.GetPosition()->x + mSize.x * 0.5f;
    max.y = transform.GetPosition()->y + mSize.y * 0.5f;
}
```

Kuva 19. AABB-luokan rakentaja.

4.6.2 Törmäyksiin reagointi

Törmäysten havaitsemiseen ja niihin reagoimiseen tarvittava logiikka kirjoitetaan omaan luokkaansa. Palikat ovat staattisia, joten niitten välisiä törmäyksiä ei tarvitse huomioida, joten tarvitsee vain tarkistaa pelaajan ja palikoitten väliset törmäykset.

Törmäystenhavaitsemiin tarvitaan vain yksi If-ehtolause, jossa verrataan pelaajahahmon ja halutun palikan törmäysalueita keskenään. Siinä hyödynnetään For-silmukkaa, jossa käydään läpi jokainen palikka ja laitetaan sen törmäysalueen tiedot If-lausekkeeseen. Jos If-ehtolauseen ehto toteutuu, kutsutaan toista funktiota, jossa ratkaisemme törmäyksen. Kuvassa 20 nähdään toteutus tästä.

```
for (size_t i = 0; i < level.size(); i++)
{
    slColl = level[i]->GetComponent<AABB>();

    if ((plColl->min.x + plVel.x) < slColl->max.x &&
        (plColl->max.x + plVel.x) > slColl->min.x &&
        (plColl->min.y + plVel.y) < slColl->max.y &&
        (plColl->max.y + plVel.y) > slColl->min.y) {
        ResolveCollision(level[i]);
    }
}
```

Kuva 20. Hyödyntämällä minimejä ja maksimeja voimme vertailla niitä suoraan. Lisäämällä pelaajan nopeuden voimme katsoa, missä pelaaja tulisi olemaan seuraavalla kuvalla.

Jotta törmäys voitaisiin ratkaista, niin tulisi tietää törmäyksen normalisoitu suuntavektori eli se puoli törmäysalueesta, jolla törmäys tapahtui. Normalisoidun suuntavektorin selvittämiseksi tarvitsee tietää, miten paljon pelaajahahmon törmäysalue leikkaa palikan törmäysaluetta jokaiselta suunnalta. Vertaamalla leikkaussyvyyksiä valitsemme sen, mikä on suurin, ja valitsemme sen sivun törmäyksen normalisoiduksi suuntavektoriksi. Tämän selvitettyä normalisoitu suuntavektori ja leikkaussyvyys lähetetään pelaajahahmolle, jotta törmäys voidaan selvittää.

Pelaajahahmon koodissa tarkistamme, onko törmäys tapahtunut x- vai y-akselilla hyödyntämällä törmäyksen normaalia. Erottamalla akselit voimme reagoida törmäykseen eri tavalla, esimerkiksi kun törmäyksen normaali on y-akselilla 1 eli pelaajahahmo on törmännyt palikkaan ylhäältä päin, voimme asettaa totuusarvon todeksi. Molemmissa tilanteissa kuitenkin pelaajahahmon sijaintia siirretään normaalin suuntaisesti ja nollataan pelaajan nopeus kyseisellä akselilla. Kuvassa 21 nähdään törmäyksiin ratkaisufunktio.

```
void Player::OnCollide(CollisionInfo colInfo) {
    glm::vec3 fix = glm::vec3(0.0f);

    if (colInfo.normal.y != 0) { // Vertical collision resolve
        if (colInfo.normal.y == 1) {
            if (!mGrounded) {
                mGrounded = true;
            }
        }
        fix.y = (colInfo.intersectionDepth * 0.35f);
        fix.y *= colInfo.normal.y;

        if (mVelocity.y <= 0 || colInfo.normal.y == -1)
            mVelocity.y = 0;
    }
    else if (colInfo.normal.x != 0) { // Horizontal collision resolve
        fix.x = (colInfo.intersectionDepth * 0.35f);
        fix.x *= colInfo.normal.x;

        mVelocity.x = 0;
    }

    *transform->GetPosition() += fix;
    transform->Translate();
}
```

Kuva 21. Pelaajaluokan törmäyksen ratkaisufunktio.

4.7 Pelaajahahmo perusliikkuminen

Pelaajahahmon perusliikkuminen koostuu kahdesta erillisestä osasta: Pelaajan nopeusvektorin muokkaaminen ja pelaajan sijainnin muokkaaminen nopeusvektorin mukaisesti. Nopeusvektoria muokataan pelaajan syötteen mukaan ja myös silloin, kun pelaaja ei kosketa maata. Kuvassa 22 nähdään koodi, jolla saadaan pelaaja liikkumaan.

Kun muutokset nopeusvektorin x-arvoon on tehty, tarkistetaan törmäykset pelaajan ja kentän välillä. Kun törmäystarkistus toteutetaan ennen kuin sijaintia on muokattu, voidaan varmistaa, että pelaajahahmo ei ilmesty kentän palikoitten sisällä ja voidaan siirtää sijaintia ennen kuin pelaajahahmo piirretään uudelleen ruudulle.

Törmäystarkistuksen jälkeen pelaajan sijaintiin voidaan lisätä pelaajan nopeusvektori. Tämän pohjan päälle implementoidaan teoriaosuudessa käydyt mekaniikat ja käytänteet, kuten kiihtyvyys ja kitka. Nopeusvektori kerrotaan muuttujalla deltaTime, joka on kulunut aika siitä, milloin peli viimeksi päivittyi. Tämän avulla pelaaja liikkuu samalla nopeudella, riippumatta pelin päivitysnopeudesta.

```
mVelocity.x = inputMgr->KeyHeld(GLFW_KEY_A) ? -mMoveSpeed * deltaTime : 0;
mVelocity.x = inputMgr->KeyHeld(GLFW_KEY_D) ? mMoveSpeed * deltaTime : 0;

if (inputMgr->KeyPressed(GLFW_KEY_SPACE) && mGrounded) {
    mVelocity.y = mJumpPower;
    mGrounded = false;
}
if (!mGrounded) {
    mVelocity.y += mGravity * deltaTime;
}
```

Kuva 22. Yksinkertainen toteutus pelaajahahmon liikkumisesta.

4.8 Kiihtyvyys ja kitka

Kiihtyvyyden ja kitkan lisääminen vaatii kahden muuttujan lisäämistä pelaajahahmoon; maksimivertikaalisen liikkumisnopeuden ja kitkan. Sen sijaan että pelaajan nopeus asetetaan olemaan pelaajahahmon liikkumisnopeus silloin, kun pelaaja haluaa liikkua vasemmalle tai oikealle, lisätään pelaajahahmon liikkumisnopeuden nopeusvektoriin. Tässä tilanteessa liikkumisnopeus toimii pelaajahahmon kiihtyvyytenä.

Seuraavaksi implementoidaan maksiminopeus. Maksiminopeuden implementointi lisää koodiin muutaman If-ehtolauseen: Pelaajan nopeuteen lisätään silloin, jos pelaaja ei ole vielä saavuttanut maksiminopeutta, muuten pelaajan nopeus tulisi olla maksiminopeus. Kitka toimii vastaavasti, mutta päinvastoin; jos pelaaja ei ohjaa hahmoa liikkumaan sivuille, niin pelaajahahmon nopeudesta vähennetään, kunnes se on nolla. Jotta hidastumista voidaan paremmin kontrolloida, hyödynnetään sille luotua kitkamuuttujaa. Kuvassa 23 nähdään päivitetty pelaajan liikkumiskoodi.

```
if (inputMgr->KeyHeld(GLFW_KEY_A)) {
    mVelocity.x -= mMoveSpeed * deltaTime;
    if (mVelocity.x < -mMaxMoveVelocity) {
        mVelocity.x = -mMaxMoveVelocity;
    }
}
if (inputMgr->KeyHeld(GLFW_KEY_D)) {
    mVelocity.x += mMoveSpeed * deltaTime;
    if (mVelocity.x > mMaxMoveVelocity) {
        mVelocity.x = mMaxMoveVelocity;
    }
}
if (!inputMgr->KeyHeld(GLFW_KEY_D) && !inputMgr->KeyHeld(GLFW_KEY_A)) {
    if (mVelocity.x > 0 && mVelocity.x != 0) {
        mVelocity.x -= mFriction * deltaTime;
        if (mVelocity.x < 0)
            mVelocity.x = 0;
    }
    else if (mVelocity.x < 0 && mVelocity.x != 0) {
        mVelocity.x += mFriction * deltaTime;
        if (mVelocity.x > 0)
            mVelocity.x = 0;
    }
}
```

Kuva 23. Samoin kuten liikkumisnopeus, kerromme kitkan myös ajan muutoksella, jotta pelaaja hidastuu yhtä nopeasti riippumatta päivitysnopeudesta.

4.9 Hyppiminen ja painovoima

Hypyn ja painovoiman implementointia varten lisätään kaksi liukulukumuuttujaa; hyppyvoima ja painovoima. Näiden lisäksi lisätään myös yksi totuusarvo, jolla seurataan, onko pelaajahahmo maassa. Kun pelaaja ei kosketa maata, pelaajan nopeusvektorin y-arvoon lisätään painovoima. Kun pelaaja koskettaa maata, niin nopeusvektorin y-arvo nollataan ja asetetaan totuusarvon todeksi.

Painovoiman jälkeen hyppimisen implementointi tarvitsee vain hieman logiikkaa; kun pelaajahahmo on maassa ja pelaaja painaa hyppynäppäintä, pelaajan nopeusvektorin y-arvon asetetaan olemaan hyppyvoima ja pelaaja ei enää koske maata. Hypyn piirteitä, kuten kestoa, voidaan muokata vaihtamalla hyppyvoimaa ja painovoimaa. Kuvassa 24 nähdään kyseinen logiikka koodissa.

```
if (inputMgr->KeyPressed(GLFW_KEY_SPACE) && mGrounded) {
    mVelocity.y = mJumpPower;
    mGrounded = false;
}

if (!mGrounded) {
    mVelocity.y += mGravity * deltaTime;
}
```

Kuva 24. Kun painovoima lisätään pelaajan nopeusakselin y-arvoon, se kerrotaan myös ajan muutoksella. Kun pelaajan nopeusakselin y-arvon asetetaan suoraan hyppyvoimaksi, sitä ei kerrota ajan muutoksella, koska muutos tapahtuu yhdellä päivityksellä.

Jotta hypynpiirteitä voidaan määrittää tarkemmin, lisätään kaksi uutta liukulukua; hypyn maksimikorkeus ja hypynaika. Jotta haluttu maksimikorkeus ja hyppyaika saavutettaisiin, tulee selvittää pelaajan hyppyvoima ja painovoima asetetulle maksimikorkeudelle ja hyppyajalle. Ensiksi selvitetään pelaajan hyppyvoima hyödyntämällä kaavaa, jolla voidaan selvittää pelaajahahmon nopeus hypynaikana:

$$\boldsymbol{f(t) = gt + v_0} \qquad (3.)$$

Tämän jälkeen selvitetään painovoima. Painovoiman selvittämisessä hyödynnetään pystysuoran heittoliikkeen kaavaa:

$$\boldsymbol{f(t) = \frac{1}{2}gt^2 + v_0 + P_0} \qquad (4.)$$

Selvittämällä kaavasta 3 hyppyvoiman ja kaavasta 4 painovoiman voimme laskea ne sijoittamalla halutun hyppykorkeuden ja hypynajan. Kuvassa 25 nähdään kaavat koodissa.

```
mGravity = -(2 * maxJumpHeight) / (jumpDuration * jumpDuration);
mJumpPower = -mGravity * (jumpDuration);
```

Kuva 25. Kaavojen hyödyntäminen koodissa. Tarkempi selostus kaavojen selvittämisessä nähdään liitteessä 2.

4.9.1 Hallittava hyppykorkeus

Hallittavan hyppykorkeuden implementointiin on muutamia erilaisia ratkaisuja. Tähän projektiin valittiin simppeli ratkaisu, joka vaatii hieman logiikkaa toimiakseen ja ei vaadi uusia muuttujia. Jos pelaajahahmon nopeusvektorin y-arvo on suurempi kuin nolla, eli pelaaja on juuri hypännyt, ja hän päästää irti hyppynäppäimestä, y-arvo puolitetaan. Kuvassa 26 nähdään toiminnon toteutus projektissa.

Tämä ratkaisu toimii, mutta jos pelissä olisi tuplahyppy tai muita tapoja, millä pelaaja pystyy vaikuttamaan nopeusvektorin y-arvoon, esimerkiksi hyppyalustat, tämä toteutus vaatisi hieman lisää logiikkaa. Silloin tulisi ottaa huomioon, jos pelaaja hyppää ja tehdä muutokset nopeusvektoriin vain silloin.

```
if (inputMgr->KeyPressed(GLFW_KEY_SPACE) && mGrounded) {
    mVelocity.y = mJumpPower;
    mGrounded = false;
}
if (inputMgr->KeyReleased(GLFW_KEY_SPACE) && mVelocity.y > 0) {
    mVelocity.y *= 0.5f;
}
```

Kuva 26. Tällä toteutuksella saadaan vastaava toiminto kuin vanhemmissa tasohyppelypeleissä, kuten Capcom yrityksen kehittämässä Megaman-pelisarjassa.

4.9.2 Lukittu putoamisnopeus

Lukitun putoamisnopeuden implementointi on miltei sama kuin kiihtyvyyden ja kitkan, mutta x-akselin sijaan se toteutuu y-akselilla. Lisäämällä uuden liukulukumuuttujan, maksimiputoamisnopeuden ja hieman logiikkaa saadaan implementoitua lukittu putoamisnopeus pelaajahahmoon. Maksimiputoamisnopeus, kuten painovoima, on negatiivinen luku, joka tekee näiden kahden muuttujan vertailemisesta helppoa.

Kun pelaaja on putoamassa ja nopeusvektorin y-arvoon lisätään painovoima, tarkistetaan, onko pelaajan nopeus y-akselilla pienempi kuin sallittu eli maksimi putoamisnopeus. Jos ehto toteutuu, nopeuden y-arvon asetetaan olemaan maksimi putoamisnopeus, minkä jälkeen pelaajahahmo jatkaa putoamista samalla nopeudella maahan asti.

4.9.3 Kojoottiajan lisääminen

Kojoottiajan implementointiin tarvitaan kaksi liukulukumuuttujaa, kojoottiaika ja laskuri. Kojoottiaika on se aika, jolloin pelaaja voi vielä hypätä käveltyään reunalta. Laskuria käytetään taas ajanlaskemiseen hyppimislogiikassa.

Kun pelaaja on maassa, niin laskuri palautetaan alkutilaan, muuten laskurista vähennetään ajan muutos. Jos laskurin arvo on vähemmän kuin nolla, niin siitä ei tarvitse enää vähentää. Viimeisenä muutoksena hyppylogiikkaan tehdään pieni muutos; sen sijaan että seurataan, onko pelaaja maassa hyödyntämällä totuusarvoa, niin katsotaan, onko kojoottiaika laskurin arvo suurempi kuin nolla. Jotta pelaaja ei pystyisi hyppäämään uudestaan ilmassa, laskuri asetetaan nollaan. Kuvassa 27 nähdään päivitetty hyppytoiminnallisuus ja kojoottiajan laskurin logiikka.

```
if (mGrounded) {
    mCoyoteTimeCounter = mCoyoteTime;
} else {
    if (mCoyoteTimeCounter > 0) {
        mCoyoteTimeCounter -= deltaTime;
    } else {
        mCoyoteTimeCounter = 0;
    }
}

if (inputMgr->KeyPressed(GLFW_KEY_SPACE) && mCoyoteTimeCounter > 0) {
    mVelocity.y = mJumpPower;

    mCoyoteTimeCounter = 0;
    mGrounded = false;
}
```

Kuva 27. Päivitetty hyppäämiskoodi.

4.9.4 Hyppypuskuroiniin lisääminen

Hyppypuskurointi toimintona on teoriassa erittäin samanlainen kuin kojoottiaika. Hyppypuskuroinnin logiikka vaatii kaksi liukulukumuuttujaa, hyppypuskurointi ajan ja laskurin. Kuten kojoottiajan kanssa, laskuria hyödynnetään hyppimislogiikassa. Hyppypuskuroinnin implementointi vaatii myös hyppimislogiikan muuttamista.

Hyppypuskurointi, kuten kojoottiaika, tarvitsee logiikkaa laskurin toiminnalle. Logiikka on miltei samanlainen, laskuri palautetaan alkutilaan silloin, kun pelaaja pitää hyppynäppäintä pohjassa. Hyppimislogiikan toinen ehto, eli onko pelaaja painanut hyppynäppäintä, vaihdetaan hyppypuskurointi laskurin arvon vertaamiseen.

4.10 Ohjattavuus ilmassa

Mekaniikan implementointi tarvitsee kaksi uutta liukulukumuuttujaa, ilmakiihtyvyyden ja ilmanvastuksen. Ilmakiihtyvyys on kerroin, jolla pelaajan liikkumisnopeus kerrotaan, kun pelaaja on ilmassa. Tämä vaatii hieman logiikan lisäämistä pelaajan liikkumiskoodiin. Ilmanvastus toimii kuten kitka, mutta silloin kun pelaaja on ilmassa.

Jos pelaaja on ilmassa, niin liikkumisnopeus kerrotaan ilmakiihtyvyyskertoimella ennen kuin se vähennetään/lisätään pelaajan nopeusvektorin x-akseliin. Koska ilmakiihtyvyys on kerroin, se voi joko parantaa tai huonontaa sitä, kuinka hyvin pelaaja pystyy hallitsemaan hahmon liikettä ilmassa. Kuvassa 28 nähdään logiikka lisättynä pelaajan liikkumiskoodiin.

Jos pelaaja on ilmassa ja lopettaa liikkumisen, pelaajan nopeuden hidastaminen kitkamuuttujalla vaihtuu hidastumiseksi ilmanvastuksella. Toisin kuin ilmakiihtyvyys, ilmanvastus ei ole tässä tilanteessa kerroin, joten sen arvoa voi muokata ilman kitkan huomioon ottamista. Kuvassa 29 nähdään logiikka lisättynä kitkalogiikkaan.

```
if (mGrounded)
    mVelocity.x -= mMoveSpeed * deltaTime;
else
    mVelocity.x -= mMoveSpeed * deltaTime * mAirControlMult;
```

Kuva 28. Käyttämällä kerrointa pelaajan liikkumisnopeus ei vaihdu liikaa, jolloin pelaajan kontrolli hahmosta pysyy miltei samana.

```
if (mGrounded)
    mVelocity.x -= mFriction * deltaTime;
else
    mVelocity.x -= mAirFriction * deltaTime;
```

Kuva 29. Käyttämällä kahta eri arvoa liikkumista voidaan hallita paljon vapaammin.

5 Pohdinta

Projektista saatiin aikaiseksi sovelias työkalu, jolla pystytään perehtymään eri mekaniikkojen ja käytänteitten vaikutukseen 2D-tasohyppelypelin pelaajakokemukseen ja millaisen kuvan se antaa pelistä ja pelinmaailmasta pelaajalle. Projektissa löytyy heikkoja puolia, kuten AABB:llä toteutettu törmäyksenhavaitseminen, joka suurilla nopeuksilla ei onnistu pysäyttämään pelaajahahmoa.

Näistä ongelmista huolimatta projekti pääosin toimii ilman ongelmia. Projektiin saatiin lisättyä yllättävän paljon muokattavia osia, lähtien pelaajan nopeudesta ja hyppykorkeudesta itse testikentän asetelmaan. Linkki projektin versionhallintaan löytyy liitteestä 3.

Projektin pohjan luominen, jotta tekstissä käydyt mekaniikat ja käytänteet päästiin implementoimaan, osoittautui hankalaksi ja tuotti ongelmia varsinkin projektin alkupäässä. Tämä tosin mahdollisti asioiden, kuten törmäyshavaitsemisen implementoinnin, jossa olisi muuten käytetty pelimoottorin valmista ratkaisua. Tämä myös tarkoitti sitä, että projektin lähdekoodia voidaan käyttää referenssinä riippumatta alustasta, ohjelmointikielestä ja pelimoottorista.

6 Yhteenveto

Tekstin tavoitteena oli luoda suomenkielinen lähde valituille mekaniikoille ja käytänteille, jossa kyseiset mekaniikat ja käytänteet implementoitaisiin projektiin, jonka lähdekoodi olisi kaikille luettavissa. Tekstissä perehdyttiin tasohyppelypeligenreen, seitsemään valittuun mekaniikkaan ja käytänteeseen, jotka lopuksi implementoitiin projektiin. Mekaniikkoihin ja käytänteisiin perehdyttiin toiminnallisuuden, niitten tavoitteitten ja teoreettisiin implementointiratkaisuihin.

Projektin toteutus jakautui kahteen vaiheeseen: moottorin luominen ja mekaniikkojen implementointi. Ensiksi luotiin tarvittavat toiminnot, kuten peliobjektien piirtäminen näytölle ja käyttäjän syötteen vastaanottaminen. Seuraavaksi projektiin lisättiin törmäyshavaitseminen, minkä jälkeen mekaniikat ja käytänteet lisättiin projektiin. Vaiheittaisesta toteutuksesta huolimatta toisessa vaiheessa jouduttiin tekemään paljonkin muutoksia itse moottoriin, kuten pelaaja syötteen vastaanottamisen luominen uudestaan.

Tämänhetkinen versio projektista suorittaa tehtävänsä hyvin tarjoamalla pohjan, josta voidaan ottaa mallia tai kehittää sen päälle uusia toimintoja. Projektin lisenssin ansiosta kuka tahansa voi hyödyntää lähdekoodia omiin tarpeisiin. Odotetaan mielenkiinnolla, miten tasohyppelypelit genrenä tulevat kehittymään ja minkälaisia uusia mekaniikkoja käytänteitä genrelle kehitetään, joilla parannettaisiin pelaajakokemusta.

Lähteet

1 The Player's Guide to Climbing Games. Electronic Games, 1983, vuosikerta 1, numero 11, s. 50–57. Saatavilla: https://archive.org/details/Electronic_Games_Volume_01_Number_11_1983-01_Reese_Communications_US/page/n47/mode/2up?view=theater Viitattu: 30.1.2023

2 MasterClass. Learn About Platform Game: 7 Examples of Platform Games. Masterclass.com [Internet]. 2021. Saatavilla: https://www.masterclass.com/articles/platform-game-explained Viitattu: 31.1.2023

3 Lance Cartelli. Throwback photos from old video game arcades. cnet.com. [Internet]. 2017. Saatavilla: https://www.cnet.com/pictures/throwback-photos-from-old-arcades/ Viitattu: 21.3.2023

4 racketboy. Platforming Games 101: Running, Jumping & More. racketboy.com [Internet]. 2011. Saatavilla: https://www.racketboy.com/retro/platforming-games-101-all-you-need-to-know Viitattu: 31.1.2023

5 Steve Bloom. Video Invaders, 1983, s. 42. Saatavilla: https://archive.org/details/book_video_invaders/page/n61/mode/2up Viitattu: 2.2.2023

6 Jason Begy. The History and Significance of Jumping in Games. [Internet]. 2010. Saatavilla: https://www.jasonbegy.com/uploads/5/0/7/7/50772065/begy_jumping.pdf Viitattu: 8.2.2023

7 David Crane, GDC. Classic Game Postmortem – PITFALL! [Video]. 2011. Saatavilla: https://www.gdcvault.com/play/1014632/Classic-Game-Postmortem-PITFALL Viitattu: 8.2.2023

8 Arcade Museum. Dragon Buster. arcade-musuem.com. [Internet]. Saatavilla: https://www.arcade-museum.com/game_detail.php?game_id=7641 Viitattu: 8.2.2023

9 GamesRadar_US. Gaming's most important evolutions. gamesradar.com. [Internet]. 2010. Saatavilla: https://web.archive.org/web/20160118203108/http://www.gamesradar.com/gamings-most-important-evolutions/?page=4 Viitattu: 8.2.2023

10 Ben Reeves. An Ode to The Most Important Power-Up: Double Jump. gameinformer.com. [Internet]. 2014. Saatavilla: https://www.gameinformer.com/b/features/archive/2014/11/04/an-ode-to-the-double-jump.aspx Viitattu: 8.2.2023

11 Super Mario Bros. Review. allgame.com. [Internet]. 2014. Saatavilla: https://web.archive.org/web/20141114120755/http://www.allgame.com/game.php?id=1320&tab=review Viitattu: 8.2.2023

12 Super Mario Bros.' 25th: Miyamoto Reveals All. ugo.com. [Internet]. 2010. Saatavilla: http://www.ugo.com/games/super-mario-bros-25th-miyamoto-reveals-all.html Viitattu: 8.2.2023

13 Sam Kennedy. Sonic the Hedgehog. 1up.com. [Internet]. 2004. Saatavilla: https://web.archive.org/web/20040822083659/http://www.1up.com/do/feature?cId=3134008 Viitattu: 8.2.2023

14 IMDb. Sonic the Hedgehog. imdb.com. [Internet]. Saatavilla: https://www.imdb.com/title/tt0291490/mediaindex/?ref_=tt_mv_sm Viitattu: 22.3.2023

15 Ross Bramble. 2D Games Vs. 3D Games: What's the Difference?. gamemaker.io. [Internet]. 2022. Saatavilla: https://gamemaker.io/en/blog/2d-games-vs-3d-games Viitattu: 9.2.2023

16 Elko Lance. ng alphas – Clockwork Knight. Next Generation, s. 81. 1995. Saatavilla: https://archive.org/details/nextgen-issue-001/page/n85/mode/2up Viitattu: 9.2.2023

17 Nebojsa Radakovic. Super Mario 64 Review. gamerevolution.com. [Internet]. 2004. Saataavilla: https://www.gamerevolution.com/review/32564-super-mario-64-review Viitattu: 14.2.2023

18 Ziff Davis. Welcome to the Jungle. Electronic Gaming Monthly, 1996, s. 56-59. Saatavilla: https://archive.org/details/ElectronicGamingMonthly_201902/Electronic%20Gaming%20Monthly%20Issue%20085%20%28August%201996%29/page/n57/mode/2up Viitattu: 14.2.2023

19 Crash Bandicoot visits Nintendo – US TV Commercial. [Video]. 1996. Saatavilla: https://www.youtube.com/watch?v=Kmpl1_aZTbY Viitattu: 14.2.2023

20 Andrew Donovan. The Lasting Relevance of the Platformer Genre. dbltap.com. [Internet]. 2021. Saatavilla: https://www.dbltap.com/posts/the-lasting-relevance-of-the-platformer-genre-01fca11jfsks Viitattu: 15.2.2023

21 Melika Jeddi. Indie Game Sub-genres Explained. indiegameculture.com. [Internet]. Saatavilla: https://indiegameculture.com/indie-culture/indie-game-sub-genres-explained/ Viitattu: 15.2.2023

22 iD Tech. 10 Type of Platforms in Platform Video Games. idtech.com. [Internet]. 2012. Saatavilla: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games Viitattu: 1.3.2023

23 Michael Klappenbach. "What is a Platform Game?". lifewire.com. [Internet]. 2021. Saatavilla: https://www.lifewire.com/what-is-a-platform-game-812371 Viitattu: 1.3.2023

24 Pierre Bienaimé. Square Roots: Donkey Kong (NES). nintendodojo.com. [Internet]. 2012. Saatavilla: https://www.nintendojo.com/features/columns/square-roots/square-roots-donkey-kong-nes Viitattu: 2.3.2023

25 Donkey Kong. silverballmuseum.com. [Internet]. Saatavilla: https://silverballmuseum.com/product/donkey-kong/ Viitattu: 3.3.2023

26 Donkey Kong Play Guide. classicgaming.cc. [Internet]. Saatavilla: http://www.classicgaming.cc/classics/donkey-kong/play-guide Viitattu: 6.3.2023

27 N. R. Kleinfield. Video Games Industry Comes Down to Earth. nytimes.com. [Internet]. 1983. Saatavilla: https://www.nytimes.com/1983/10/17/business/video-games-industry-comes-down-to-earth.html Viitattu: 6.3.2023

28 Evan Narcisse. How Nintendo Made the NES (And Why They Gave It a Gun). kotaku.com. [Internet]. 2015. Saatavilla: https://web.archive.org/web/20151016230249/http://kotaku.com/an-insiders-memories-of-making-the-nintendo-entertainme-1737014878 Viitattu: 7.3.2023

29 ClassicGaming Museum. Nintendo Entertainment System (NES). classicgaming.gamespy.com. [Internet]. 2012. Saatavilla: https://web.archive.org/web/20121029033423/http://classicgaming.gamespy.com/View.php?view=ConsoleMuseum.Detail&id=26&game=5 Viitattu: 7.3.2023

30 Keith Stuart. Super Mario Bros: 25 Mario facts for the 25th anniversary. theguardian.com. [Internet]. 2010. Saatavilla: https://www.theguardian.com/technology/games-blog/2010/sep/13/games-gameculture Viitattu: 7.3.2023

31 Nintendo. Mario through the years. nintendo.com. [Internet]. Saatavilla: https://mario.nintendo.com/history/ Viitattu: 7.3.2023

32 Geoffrey Douglas Smith. Super Mario Bros. Review. allgame.com. [Internet]. 2014. Saatavilla: https://web.archive.org/web/20141114120755/http://www.allgame.com/game.php?id=1320&tab=review Viitattu: 7.3.2023

33 Chris Scullion. The Nintendo 64 and Super Mario 64 turn 25 years old today. videogameschronicle.com. [Internet]. 2021. Saatavilla: https://www.videogameschronicle.com/news/the-nintendo-64-and-super-mario-64-turn-25-years-old-today/ Viitattu: 11.3.2023

34 Nintendo Power. Super Mario 64 Player's Guide, s. 4-5. archive.org. [Internet]. Saatavilla: https://archive.org/details/SuperMario64OfficialPlayersGuide/mode/2up Viitattu: 13.3.2023

35 Niall O'Donoghue. How Super Mario 64 changed the face of the games industry – 25th Anniversary. vg247.com. [Internet]. 2021. Saatavilla: https://www.vg247.com/super-mario-64-changed-face-games-industry-25th-anniversary Viitattu: 13.3.2023

36 Martin Watts. "Super Mario 64 review – how does it play today?". n64today.com. [Internet]. 2018. Saatavilla: https://n64today.com/2018/09/01/super-mario-64-review/ Viitattu: 13.3.2023

37 Madeline Thorson. Maddy Makes Games-kotisivut. maddymakesgames.com. [Internet]. Saatavilla: https://www.maddymakesgames.com/ Viitattu: 14.3.2023

38 Daniel R. Miller. Review Round-Up: Celeste is being called a "surprise masterpiece". gamezone.com. [Internet]. 2018. Saatavilla: https://www.gamezone.com/news/review-round-celeste-called-surprise-masterpiece/ Viitattu: 14.3.2023

39 Oscar Dayus. Celeste Review: More Than Just a Great Platformer. gamespot.com. [Internet]. 2018. Saatavilla: https://www.gamespot.com/reviews/celeste-review-more-than-just-a-great-platformer/1900-6416843/ Viitattu: 14.3.2023

40 Game Maker's Toolkit. "Why Does Celeste Feel So Good to Play?". youtube.com. [Video]. 2020. Saatavilla: https://www.youtube.com/watch?v=yorTG9at90g Viitattu: 22.3.2023

41 Fangkai Yang. Collision Detection in Computer Games. kth.se. [Internet]. Saatavilla: https://www.kth.se/social/files/574684e8f2765458722b5565/DH2323%20Collision%20Detection%20I.pdf Viitattu: 7.4.2023

42 James M. Van Verth, Lars M. Bishop. Essential Mathematics for Games and Interactive Applications Third Edition. Boca Raton, Florida, USA: CRC Press; 2016. Viitattu: 7.4.2023

43 Miguel Casillas. AABB to AABB. miguelcasillas.com. [Internet]. Saatavilla: http://www.miguelcasillas.com/?p=30 Viitattu: 7.4.2023

44 Elliot Forbes. AABB Collision Detection Tutorial. tutorialedge.net [Internet]. 2017. Saatavilla: https://tutorialedge.net/gamedev/aabb-collision-detection-tutorial/ Viitattu: 19.4.2023

45 Luu the Vinh. Using Swept AABB to detect and process collision. amanotes.com. [Internet]. 2021. Saatavilla: https://www.amanotes.com/post/using-swept-aabb-to-detect-and-process-collision Viitattu: 19.4.2023

46 Xenozip. Super Street Fighter 2 Hitboxes: Claw and Chun. youtube.com. [Video]. 2011. Saatavilla: https://www.youtube.com/watch?v=SB2CvPt87X8 Viitattu: 19.4.2023

47 David Strachan. Tips and Tricks for good platforming games (with examples). davetech.co.uk. [Internet]. Saatavilla: http://www.davetech.co.uk/gamedevplatformer Viitattu: 26.7.2023

48 Bruno Guedes. Physics for Game Dev – A Platformer Physics Cheatsheet. medium.com. [Internet]. 2019. Saatavilla: https://medium.com/@brazmogu/physics-for-game-dev-a-platformer-physics-cheatsheet-f34b09064558 Viitattu: 3.8.2023

49 Coyote time. wiktionary.org. [Internet]. 2022. Saatavilla: https://en.wiktionary.org/wiki/coyote_time Viitattu: 6.8.2023

50 Jlett12. Advanced Platformer Movement. instructables.com. [Internet]. Saatavilla: https://www.instructables.com/Advanced-Platformer-Movement/ Viitattu: 6.8.2023

51 Kyle Pulver. Jump Input Buffering. kpulv.com. [Internet]. 2013. Saatavilla: http://kpulv.com/106/Jump_Input_Buffering/ Viitattu: 6.8.2023

52 Pav. Jumping control in 2D pixel-perfect platformers. pavcreations.com. [Internet]. 2021. Saatavilla: https://pavcreations.com/jumping-controls-in-2d-pixel-perfect-platformers/ Viitattu: 7.8.2023

53 Yoann Pignole. Platformer controls: how to avoid limpness and rigidity feelings. gamedeveloper.com. [Internet]. 2014. Saatavilla: https://www.gamedeveloper.com/design/platformer-controls-how-to-avoid-limpness-and-rigidity-feelings Viitattu: 9.8.2023

54 Pawel. Guide to Jumping in platformers. defold.com. [Internet]. 2022. Saatavilla: https://forum.defold.com/t/guide-on-jumping-in-platformers/71882 Viitattu: 9.8.2023

56 Mohan Rajagopalan. Designing a 2D jump. gamedeveloper.com. [Internet]. 2014. Saatavilla: https://www.gamedeveloper.com/design/designing-a-2d-jump Viitattu: 10.8.2023

57 Charlie Wood. ”What is gravity?”. space.com. [Internet]. 2023. Saatavilla: https://www.space.com/classical-gravity.html Viitattu: 10.8.2023

58 Adam Lefky, Artem Gindin. Acceleration Due to Gravity: Super Mario Brothers. hypertextbook.com. [Internet]. 2007. Saatavilla: https://hypertextbook.com/facts/2007/mariogravity.shtml Viitattu: 10.8.2023

59 Samuel J. Ling, Jeff Sanny, William Moebs. University Physics Volume 1. openstax.com. [Internet]. 2012. Saatavilla: https://openstax.org/books/university-physics-volume-1/pages/preface Viitattu: 10.8.2023

60 Microsoft. Visual Studio -kotisivut. visualstudio.microsoft.com. [Internet]. Saatavilla: https://visualstudio.microsoft.com/vs/ Viitattu: 31.8.2023

61 CMake-kotisivut. cmake.org. [Internet]. Saatavilla: https://cmake.org/ Viitattu: 31.8.2023

62 Mark Segal, Kurt Akeley. The OpenGL Graphics System: A Specification. registry.khronos.org. [Internet]. 2010. Saatavilla: https://registry.khronos.org/OpenGL/specs/gl/glspec40.core.pdf Viitattu: 31.8.2023

63 GLFW-kotisivut. glfw.org. [Internet]. Saatavilla: https://www.glfw.org/ Viitattu: 31.8.2023

64 GLEW-kotisivut. glew.sourceforge.net. [Internet]. Saatavilla: https://glew.sourceforge.net/ Viitattu: 31.8.2023

65 GLM:n github sivu. github.com. [Internet]. Saatavilla: https://github.com/g-truc/glm Viitattu: 31.8.2023

66 ImGui:n github sivu. github.com. [Internet]. Saatavilla: https://github.com/ocornut/imgui Viitattu: 31.8.2023

Liitteet

axis-aligned bounding box (AABB) akselin mukainen rajausalue

collision törmäys

collision detection törmäyshavaitseminen

collider törmäysalue

coyote time kojoottiaika

jump buffering hyppypuskurointi

oriented bounding box (OBB) käännettävä rajausalue

platform taso

platformer tasohyppelypeli, tasoloikkapeli

timer ajastin

variable jump height hallittava hyppykorkeus

Pelaajahahmon nopeus hypyn aikana

$f(t) = gt + v_0$, jossa

g on painovoima,

t on kulunut aika

v_0 on hyppyvoima

Pelaajan nopeus on nolla, kun hypyn huippukorkeus h on saavutettu, jolloin

$$f(t_h) = 0$$

Tästä voimme selvittää hyppyvoiman, jolloin

$$0 = gt_h + v_0 \qquad | -gt_h$$

$$v_0 = gt_h$$

Painovoiman selvittäminen pystysuoran heittoliikkeen kaavasta

$f(t) = \frac{1}{2}gt^2 + v_0 t + P_0$, jossa

g on painovoima,

t on aika,

v_0 on hyppyvoima,

P_0 on pelaajan sijainti y-akselilla hypyn alussa

Toisesta kaavasta tiedämme, että $v_0 = -gt_h$ hypyn huippu ajalla

Sen lisäki pelaajan sijainti y-akselilla oletetaan olevan nolla, jolloin $P_0 = 0$

Pelaaja on saavuttanut huippukorkeuden h hypyn puolessa välissä, jolloin $t = t_h$

Sijoittamalla arvot kaavaan saamme

$h = \frac{1}{2}gt_h^2 + (-gt_h)\,t_h + 0$, josta voimme selvittää painovoiman

$$h = \frac{1}{2}gt_h^2 + (-gt_h^2)$$

$$h = -\frac{1}{2}gt_h^2 \qquad | \cdot 2$$

$$2h = -gt_h^2 \qquad | : t_h^2$$

$$-g = \frac{2h}{t_h^2} \qquad | \cdot (-1)$$

$$g = -\frac{2h}{t_h^2}$$

https://github.com/Hippopotaska/2DCharacterController